UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

WAGNER GUIMARÃES AL ALAM

A ABSTRAÇÃO DE CONTRATOS CONTEXTUAIS PARA ALOCAÇÃO DE RECURSOS DE SISTEMAS DE COMPUTAÇÃO PARALELA ORIENTADOS A COMPONENTES EM NUVENS

FORTALEZA

2019

WAGNER GUIMARÃES AL ALAM

A ABSTRAÇÃO DE CONTRATOS CONTEXTUAIS PARA ALOCAÇÃO DE RECURSOS DE SISTEMAS DE COMPUTAÇÃO PARALELA ORIENTADOS A COMPONENTES EM NUVENS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Computação de Alto Desempenho

Orientador: Prof. Dr. Francisco Heron de Carvalho Junior (Orientador)

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A1a Al Alam, Wagner Guimarães.
A abstração de contratos contextuais para alocação de recursos de sistemas de computação paralela orientados a componentes em nuvens / Wagner Guimarães Al Alam. – 2019.
213 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2019.
Orientação: Prof. Dr. Francisco Heron de Carvalho Junior.

1. computação em nuvem. 2. programação paralela. 3. contratos contextuais. 4. CBSE. 5. HPC. I. Título.
CDD 005

WAGNER GUIMARÃES AL ALAM

A ABSTRAÇÃO DE CONTRATOS CONTEXTUAIS PARA ALOCAÇÃO DE RECURSOS DE SISTEMAS DE COMPUTAÇÃO PARALELA ORIENTADOS A COMPONENTES EM NUVENS

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciência da Computação. Área de Concentração: Computação de Alto Desempenho

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Francisco Heron de Carvalho Junior (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Antonio Tadeu Gomes de Souza
Laboratório Nacional de Computação Científica (LNCC)

Prof. Dr. André Rauber Du Bois
Universidade Federal de Pelotas (UFPel)

Prof. Dr. Lincoln Souza Rocha
Universidade Federal do Ceará (UFC)

Profa. Dra. Rossana Maria de Castro Andrade
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Agradeço a todos que de alguma maneira ajudaram o desenvolvimento deste trabalho, de maneira direta ou indiretamente.

Especialmente ao prof. Francisco Heron de Carvalho Junior, agradeço por orientar o desenvolvimento deste trabalho.

À minha família, agradeço pelo apoio no decorrer do curso e me incentivar a concluí-lo.

Agradeço também aos meus colegas de grupo de pesquisa, que motivaram diariamente a execução do trabalho.

E à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) e à Universidade Federal do Ceará, deixo meus agradecimentos por contribuir e investir nas pesquisas desenvolvidas.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

A Computação de Alto Desempenho (HPC[1]) vem empregando técnicas e ferramentas para construção de soluções para execução de tarefas que demandam alto poder de processamento, tanto nos programas quanto nas plataformas de execução. Nesse contexto, o uso de nuvens computacionais como plataforma de serviços de HPC ainda é um desafio, pois muitos usuários mostram-se receosos em adotar esse modelo, permanecendo com a execução de suas aplicações em seus próprios parques de recursos computacionais. Dentre os problemas relatados por parte desses usuários, destacam-se as restrições quanto às escolhas que os usuários podem fazer nos seus recursos, como a localização geográfica das plataformas de execução, uso de acelerador computacional específico, além da sobrecarga adicionada pela execução virtualizada de hardware e rede de comunicação. A fim de promover o acesso aos ambientes de nuvens computacionais de modo a satisfazer os requisitos dos usuários dessa área, bem como prover acesso aos recursos heterogêneos sob demanda, foi proposto pelo grupo de HPC da Universidade Federal do Ceará, um ambiente de computação em nuvens chamado HPC Shelf. A HPC Shelf tem como base o paradigma de orientação à componentes, utilizando a abstração de componentes tanto para representar os programas quanto as plataformas de execução. Nesse cenário, esta tese introduz o Alite, um arcabouço para seleção de componentes de sistemas de computação paralela, notadamente com requisitos HPC, onde tais componentes representam tanto o software quanto o hardware que o hospeda para execução, as plataformas de computação paralela. O Alite permite a abstração de componentes através de contratos contextuais, os quais determinam propriedades que influenciam na implementação de componentes. Além de requisitos da aplicação e características da plataforma de execução alvo, são tratados requisitos de qualidade de serviço (QoS) e de custo de execução, considerados importantes dentro do novo contexto imposto pela HPC em nuvens. Neste trabalho, são aplicados técnicas de tomadas de decisão baseadas em múltiplos critérios para validação do arcabouço. Um protótipo de implementação do Alite foi desenvolvido com propósito de avaliar o sistema de contratos contextuais com respeito a aspectos de expressividade, desempenho e eficácia, evidenciando a sua aplicabilidade em cenários reais. Com relação à expressividade, foi projetado um arcabouço de plataformas virtuais para a HPC Shelf, as quais constituem plataformas heterogêneas de computação em *clusters* com algumas características consideradas comuns no projeto dessas plataformas. A partir desse arcabouço, é apresentado um arcabouço de plataformas virtuais sobre

[1] *High Performance Computing*

o serviço EC2 da Amazon. Finalmente, foi desenvolvido um arcabouço de componentes de computação para multiplicação de matrizes com base na interface de nível 3 da biblioteca BLAS, demonstrando como funcionalidades de bibliotecas científicas de uso disseminado podem ser encapsuladas e reutilizadas por meio de componentes na HPC Shelf. O estudo experimental mostra que o tempo de resolução de contratos dos usuários foi considerado satisfatório para uma quantidade extensa de plataformas registradas, assim como o resultado da classificação mostrou-se alinhado com os objetivos delimitados pela HPC Shelf.

Palavras-chave: Computação em nuvem. Componentes. Computação paralela. Programação paralela. CBSE. Contratos contextuais. CAD.

ABSTRACT

High performance computing has been employing techniques and tools to build solutions for performing tasks that demand high performance in both programs and execution platforms. In the evolution of these techniques, the use of high performance computational clouds is still a challenge, since several users of this area are being reluctant to adopt this model, remaining with the execution of their applications in their own computational resource parks. Among the problems reported by these users are restrictions on the choices users can make in their resources, such as the geographic location of execution platforms, the use of specific computational accelerator, and the added overhead of virtualized hardware execution and communication network. In order to promote access to cloud computing environments to meet the requirements of users in this area, as well as to provide access to heterogeneous resources on demand, it was proposed by the HPC group of the Universidade Federal do Ceará, an environment of cloud computing called HPC-Shelf. HPC-Shelf is based on the component orientation paradigm, using component abstraction to represent programs and execution platforms. In this scenario, this thesis introduces Alite, a method for choosing Computer Systems, formed by pairs of components that represent the programs and their respective execution platforms. Alite, through a series of abstractions, is able to treat the universe of components, registered in a database, using its characteristics that can be represented hierarchically, or through numerical values that can be set by users or automatically calculated from the valuations of other numerical characteristics. An Alite implementation prototype was developed with the purpose of evaluating the contextual contracting system with respect to aspects of expressiveness, performance and effectiveness, evidencing its applicability in real scenarios. Regarding expressiveness, a virtual platform framework was designed for HPC Shelf, which constitute heterogeneous cluster computing platforms with some features considered common in modern cluster design. From this framework, a virtual platform framework on Amazon's EC2 service is presented. Finally, a matrix multiplication computing component framework was developed based on the BLAS library's level 3 interface, demonstrating how features of widespread scientific libraries can encapsulated and reused through components at HPC Shelf. User contract resolution time was considered satisfactory for a large number of registered platforms, as well as the result of the classification was aligned with the objectives defined by HPC-Shelf.

Keywords: Cloud computing. Components. Parallel computing. Parallel programming. CBSE.

Contextual contracts. HPC.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Computadores modernos baseados na arquitetura de John Von Neumann, aceleram, ou mesmo viabilizam, a solução de problemas desde as últimas 8 décadas, frequentemente substituindo o papel de especialistas humanos de uma respectiva área, incapazes de resolver manualmente problemas que o computador é capaz de resolver de maneira muito rápida e com alto grau de precisão.

Para os propósitos deste trabalho, é conveniente distinguir as aplicações de interesse computacional em dois grupos: as de interesse comercial e as de interesse científico. As aplicações de interesse comercial apresentam como característica principal o provimento de serviços para vários usuários clientes, de modo que a carga de trabalho depende da quantidade de usuários atendidos simultaneamente, além da alta disponibilidade dos serviços (TANENBAUM; STEEN, 2007). O interesse e grau de complexidade dessas aplicações tem sido fortemente alavancadas de acordo com a consolidação e evolução da infraestrutura global de comunicação da internet. Já as aplicações científicas, de busca de conhecimento, são geralmente desenvolvidas para resolver problemas computacionalmente árduos a partir de um conjunto de dados fornecidos como entrada, seja antes ou durante a execução. Na solução de tais problemas, programas que implementam a solução podem executar durante dias para atingir o resultado desejado. Aplicações científicas surgem tanto no âmbito industrial, por exemplo em pesquisa aplicada com o propósito da criação de novos produtos baseados em inovação tecnológica (e.g. fármacos, automóveis, aeronaves, edificações, etc), quanto no âmbito acadêmico, tanto na pesquisa básica (e.g. ciências computacionais) quanto na pesquisa aplicada (e.g. simulação de fenômenos naturais tais como furacões, tornados e terremotos). Mais recentemente, novas aplicações de busca e inferência de conhecimento tem ganho importância nas áreas de BigData e aprendizagem de máquina[1].

Este trabalho concentra-se nas aplicações ditas científicas, notadamente aquelas que demandam grande poder de processamento, cujas instâncias de problema que interessam exigem o uso de técnicas particulares para aceleração da computação ou aumento da quantidade de dados processados, mesmo quando executando nos mais modernos e rápidos computadores. Essas aplicações são aquelas que interessam à área de computação chamada Computação de Alto Desempenho (HPC[2]). A HPC relaciona técnicas e ferramentas para construção e programação

[1] *machine learning*.
[2] *High Performance Computing*

de computadores com máximo de poder de processamento possível, bem como capacidade de memória. Além do *hardware* especializado, explora ao máximo o seu desempenho através de desenvolvimento de *softwares* otimizados para tais arquiteturas. Sua existência é motivada por aplicações computacionais intensivas em processamento e/ou uso de memória e que simulam ou buscam respostas sobre o comportamento do mundo real (VECCHIOLA *et al.*, 2009). No restante deste trabalho, essas aplicações serão tratadas apenas por aplicações HPC.

Nos computadores, o sistema operacional (SO) gerencia a interação entre as aplicações dos usuários (*software*) e o *hardware*, de modo que uma aplicação qualquer, incluindo as aplicações de HPC, compartilha recursos do *hardware* tanto com aplicações do sistema operacional como com as demais aplicações. Assim, define-se contexto de uma aplicação em determinado computador, como o conjunto das características do equipamento, tais como a quantidade de memória livre, quantidade de núcleos disponível, etc, respeitando o compartilhamento de recursos proporcionado pelo SO.

Além disso, para as aplicações de HPC, a busca por maior eficiência computacional não possui limites. Logo, uma maior eficiência computacional em relação a existente levará à cobertura de instâncias de problemas de interesse ainda maiores que as instâncias cobertas atualmente. Devido a isso, é conveniente para tais aplicações a disponibilidade da máxima eficiência computacional disponível, frequentemente a qualquer custo, apenas limitadas por barreiras tecnológicas.

O processamento paralelo é a principal técnica utilizada atualmente para viabilizar as aplicações de HPC. Por conta disso, os computadores utilizados para suas finalidades apresentam características diferenciadas em relação a computadores convencionais, apresentando alto custo de aquisição e implantação, bem como maior complexidade para o desenvolvimento das aplicações, uma vez que se faz imprescindível que usem de maneira eficiente os recursos de computação disponíveis, frequentemente heterogêneos.

Diversas técnicas vem sendo utilizadas para lidar com a complexidade no desenvolvimento de programas paralelos. Tradicionalmente, desde os anos 1990, os destaques tem sido as interfaces portáveis de desenvolvimento, tais como bibliotecas de passagem de mensagens, especialmente o *Message Passing Interface* (MPI) (DONGARRA *et al.*, 1996), para plataformas de computação paralela de memória distribuída, e diretivas de compilação combinadas com subrotinas de biblioteca, como o OpenMP (BOARD, 1997), para plataformas de memória compartilhada. Entretanto, poucas ferramentas que conciliem tal nível de portabilidade e eficiência

com alto nível de abstração e expressividade para especificação simples de padrões de computação paralela tem se consolidado universalmente. A conciliação entre alto nível de abstração, eficiência e portabilidade tem sido alcançada através de bibliotecas de subrotinas científicas de propósito especial, muitas das quais apresentam versões pré-paralelizadas para uma ou mais arquiteturas de plataformas de computação paralela. No entanto, devido ao seu caráter de propósito especial, apresentam expressividade limitada a um conjunto restrito, embora representativo, de padrões de paralelismo pré-determinados. Por outro lado, técnicas que oferecem maior nível de expressividade e abstração, como bibliotecas de esqueletos de programação, não tem alcançado o nível esperado de aceitação, a despeito do intenso trabalho de pesquisadores no aprimoramento dessa técnica (COLE, 1989; RABHI; GORLATCH, 2002; COLE, 2004; KUCHEN; COLE, 2006).

A principal dificuldade de oferecer interfaces de programação paralela que conciliem os requisitos de portabilidade, eficiência, grau de abstração e expressividade tem relação com a complexidade arquitetural das plataformas de computação paralela, em especial motivada pela sua heterogeneidade. Nos últimos anos, essas plataformas começaram a ser utilizadas sob a forma de prestação de serviço, através do formato de nuvem computacional. Nas nuvens computacionais os recursos de execução são providos sob demanda (*utility computing*), semelhante à distribuição de energia elétrica nas cidades. Esse serviço pode ser provido em diversos níveis de abstração e em diversos modelos de gerência e serão objeto de estudo da Seção 2.2.

A HPC Shelf é uma plataforma que vem sendo desenvolvida por nosso grupo de pesquisa para a convergência de tecnologias de nuvens computacionais e componentes de *software* com o propósito da oferta de serviços HPC. A HPC Shelf , portanto, uma das contribuições deste trabalho, muito embora em um sentido coletivo, sendo também parte da contribuição de trabalhos de doutoramento de outros pesquisadores do grupo de pesquisa dentro do qual esse projeto emergiu (SILVA, 2016; ALENCAR, 2017; REZENDE, 2017; DANTAS, 2017).

A HPC Shelf visa prover um ambiente de nuvem computacional expressivo, composto por várias aplicações voltadas aos domínios de interesse de usuários finais, ditos *especialistas*, e eficiente, suportando multiplos níveis de abstração a fim de explorar as características de infraestruturas de computação paralela. A portabilidade de componentes entre diferentes infraestruturas é consequência natural do emprego de conceitos de Computação de Alto Desempenho

Baseado em Componentes (CBHPC[3]) (STEEN, 2006).

Para que especialistas possam resolver problemas dos seus interesses por intermédio das aplicações, é necessário que a aplicação crie e gerencie o ciclo de vida de *sistemas de computação paralela*, voltados à sua solução. Esses sistemas representam a composição de um conjunto de componentes paralelos cuja execução de suas ações computacionais é guiada através de um programa *workflow*. O responsável pela construção de sistemas de computação paralela são usuários chamados *provedores de aplicações*, responsáveis, em um sentido mais geral, pela construção de aplicações na HPC Shelf.

Os componentes de sistemas de computação paralela são compatíveis com o modelo Hash, um modelo de componentes inerentemente paralelos (CARVALHO-JUNIOR; LINS, 2008). Esses componentes são referenciados, em sistemas de computação paralela, através da abstração de *contratos contextuais*, os quais determinam interfaces de componentes para o ambiente, bem como as suposições sobre seus respectivos contextos de execução que guiam a forma como são implementados (definição de algoritmos, estruturas de dados, linguagens, bibliotecas de comunicação, padrões de comunicação, etc). O *contexto* de um contrato contextual modela os requisitos de uma aplicação hospedeira para com o componente e as características arquiteturais de uma plataforma de computação paralela alvo sobre o qual o componente instanciado para o determinado contrato executará. Dentre os componentes, estão as próprias plataformas de computação paralela que fazem parte do sistema de computação paralela, chamadas de *plataformas virtuais*,.

Este trabalho propõe o Alite, uma alternativa para o problema da escolha de componentes de sistemas de computação paralela de forma sensível ao conjunto de recursos e restrições à sua utilização que caracterizam o ambiente de nuvem computacional sob o qual o sistema deverá executar. Para isso, o Alite gerencia o conjunto de contratos contextuais oferecidos a aplicações da HPC Shelf, suportando um modelo de seleção de componentes, que sejam apropriados para atender os requisitos desses contratos, de acordo com o contexto onde devem ser instanciados em componentes.

Há outras iniciativas de trabalhos relacionados que buscam prover o uso de ambientes de nuvens computacionais para HPC. No entanto, de acordo com os relatórios do projeto UberCloud (GENTZSCH; YENIER, 2013; GENTZSCH; YENIER, 2014; GENTZSCH; YENIER, 2015; GENTZSCH; YENIER, 2016; GENTZSCH; YENIER, 2018), os dois principais proble-

[3] *Component-Based High Performance Computing.*

mas que dificultam a adoção desse modelo pelos usuários de HPC são a sobrecarga adicionada pela virtualização e a transparência de localização. Além dessas dificuldades, diversos participantes do projeto citam dificuldade para acessar os resultados, pois geralmente essas aplicações retornam grandes quantidades de dados e é demasiadamente custoso fazer o *download* desses dados. Nesse caso, seria mais vantajoso quer fosse provida uma ferramenta de visualização e pós tratamento dos dados.

As questões de pesquisa que nortearam a concepção deste trabalho estão elencadas a seguir:

1. É viável utilizar ambientes de nuvens computacionais para executar aplicações de alto desempenho, pelos usuários convencionais de HPC?

   Essa questão foi dividida em outras questões:

   a) Quais os fatores que atualmente dificultam a adoção das nuvens computacionais pelos usuários de HPC ?
   b) Como minimizar ou evitar os fatores identificados?
   c) Como proporcionar uma escolha de componentes que respeite todas as restrições dos envolvidos no processo de execução, avaliando as métricas de desempenho e custo para guiar as escolhas?
2. Como representar os recursos de diferentes provedores de recursos para serem disponibilizados aos usuários?
3. É possível, no processo de escolha dos componentes, sobrepor os problemas identificados pelos usuários de HPC?

1.1 Trabalhos Relacionados

O desafio de alocação automática de recursos computacionais de nuvens, bem como o processo de escolha automática de componentes, de caráter mais geral, vem sendo objeto de estudo de outros trabalhos de pesquisa em desenvolvimento.

A escolha de componentes de *software* vem sendo estudada desde a década de 1990, e diversas soluções já foram propostas (PANDE *et al.*, 2013). Posteriormente, com o surgimento das grades e nuvens computacionais, esse problema tornou-se mais complexo, pois além de escolher os componentes de *software*, também se faz necessária a escolha da plataforma onde irão executar.

Em Hall *et al.* (2008), os autores exploram os desafios da programação orientada a

componentes para aplicações de Computação Alto Desempenho, tendo grades computacionais como ambiente de execução. São identificados os principais problemas da escolha otimizada dos componentes, onde é discutido uma variedade de trabalhos que podem ser usados como base para o desenvolvimento de um ambiente de desenvolvimento que faça automaticamente melhorias nos componentes. Ambientes de nuvens computacionais apresentam os mesmos problemas identificados em ambientes de grades, além dos custos e degradação de desempenho devido à virtualização, recurso essencial nesses ambientes.

Diversos trabalhos estão sendo desenvolvidos objetivando a alocação automática de aplicações em ambientes de nuvens computacionais, de modo a maximizar algumas características do ambiente de execução, ou ainda, minimizar outras. Nesse contexto, Garg *et al.* (2011a) oferecem uma importante contribuição, pois apresentam um modelo de classificação para serviços de nuvens computacionais com critérios valoráveis e não valoráveis, utilizando a técnica AHP para essa classificação. O método AHP escolhido funciona bem com poucas alternativas, mas, como esse método calcula o peso de cada critério e faz a comparação par a par depois de aplicá-los aos valores das alternativas, resulta em uma grande sobrecarga ao método de medição.

Li *et al.* (2010) também propõem um *framework* para escolha de qual ambiente de nuvem computacional utilizar. Nesse trabalho, são explorados os critérios de avaliação, bem como quais utilizar, e ainda, a normalização dos valores. Para identificar a percepção do usuário como relação ao *framework*, é feita uma abordagem da percepção que o usuário tem dos critérios, para que os resultados estejam de acordo com o definido. Como resultado, são apresentadas as avaliações dos ambientes de nuvem Amazon S3, Rackspace Cloud, Google App Engine e Microsoft Azure.

Por sua vez, Niehorster *et al.* (2010) resolvem um problema mais específico. Dado um ambiente de nuvem segundo o modelo SaaS (*Software as a Service*), para HPC, é possível alocar uma aplicação de acordo com seu SLA, e, caso seja possível, gerenciar sua execução para que o SLA não seja violado. Na ferramenta que propõem, utilizam modelos *fuzzy* para predição de *deadline*. Para que seja possível a manutenção do *deadline*, deve ser permitida a reconfiguração dinâmica de *hardware*. Supõem-se que não estejam disponíveis detalhes arquiteturais das plataformas. Dessa forma, as aplicações devem permitir o acesso ao seu *status*. O custo de orçamento é elevado, pois o ambiente de nuvem realiza uma execução parcial da aplicação a fim de identificar o desempenho do *hardware* e a complexidade da aplicação.

Além do trabalho de Niehorster *et al.* (2010), outros trabalhos resolvem problemas relativos à alocação de componentes em um determinado conjunto de recursos, como o trabalho de Shi *et al.* (2011), o qual apresentam um estudo sobre a granularidade das aplicações para serem executadas em ambientes de nuvens computacionais. É utilizada uma multiplicação de matrizes como estudo de caso, onde, através do modelo de avaliação de tempo proposto, permite-se ajustar a granularidade das tarefas.

Buyya *et al.* (2011b) apresentam uma estratégia de alocação de recursos aplicada ao *framework* Aneka, onde o ambiente realiza a alocação dinâmica dos recursos a partir do tempo total de execução previsto. Conforme a avaliação efetuada, realiza a manutenção dos recursos disponibilizados. Trabalhos que sucederam Buyya *et al.* (2011b) buscaram propor modelos de alocação dinâmica de recursos. Toosi *et al.* (2018) estendeu o ANEKA para fazer a alocação de recursos baseada em *deadline*, através da técnica de *cloud bursting*, quando os recursos são executados em uma nuvem privada e os recursos de nuvens públicas são usadas sob demanda na situação onde os recursos da nuvem privada tornam-se insuficientes. Para isso, utiliza uma nuvem privada como base e um ambiente de nuvem pública homogêneo para complementar os recursos de *hardware*. Como avanço em relação aos trabalhos anteriores, os autores citam a inclusão do tempo de transferência de dados entre os nós e o tempo de instanciação de máquinas virtuais. Neste ano, Sandhu *et al.* (2018) estendeu o trabalho anterior, de modo a possibilitar aos desenvolvedores a personalização dos algoritmos de alocação através de uma API específica.

Garg *et al.* (2011b) apresentam um serviço de alocação de recursos baseado em SLA's, orientada tanto para sistemas comerciais quanto para aplicações de Computação Alto Desempenho. O serviço ainda apresenta um modelo de custo e penalização.

Younge *et al.* (2010) apresentam um *framework* de alocação de máquinas virtuais que tem como objetivo a alocação de VM/Host com o consumo energético menor que os mecanismos nativos dos VMM. Para alcançar esse objetivo, identificaram que, nos processadores atuais, a diferença do consumo energético para utilização de um núcleo, dois núcleos, até a quantidade total de núcleos, é menor que se utilizassem mais processadores com somente um núcleo. Essa estratégia garantiu que conseguissem uma diminuição do consumo principalmente quando desligados os computadores físicos que estão ociosos, e, caso necessário, sua reativação ocorre através do recurso *WAKE-ON-LAN*. Outro trabalho que explora a eficiência energética é apresentado por Buyya *et al.* (2010), o qual apresenta um sistema de gerenciamento de migração de máquinas virtuais de modo a diminuir o consumo de energia. Utiliza o controle

de frequência/tensão dos processadores para identificar a tensão e, baseada nisso, a estratégia identifica os processadores que devem ser migrados ou devem receber novas máquinas virtuais. A eficiência energética tem ganhado cada vez mais importância no processo de escolha de recursos computacionais, sendo um dos possíveis critérios de escolha de componentes a ser aplicado pela HPC Shelf.

Outros trabalhos propõem a escolha dos componentes de *software* com diferentes objetivos, como o trabalho de (PANDE *et al.*, 2013), o qual define um modelo de seleção de componentes ótimo, em relação à custo/benefício, baseado em programação inteira. É definida uma métrica chamada *pliability*, a qual considera atributos qualitativos ponderadas dos componentes, cada uma com seu peso definido conforme sua importância. Os atributos de qualidade considerados são: segurança, confiabilidade, desempenho, tolerância a falhas, disponibilidade, testabilidade e manutenibilidade. O modelo busca maximizar a métrica *pliability*, que é definida pela soma das qualidades de um componente, subtraída do custo do mesmo. Esse modelo é apropriado para aplicações comerciais, onde diferentes perfis de aplicações resultam em diferentes combinações de pesos entre os atributos de qualidade.

Alkhamisi e Qureshi (2013) fez uma proposta de estratégia de escolha automática de componentes em CBSE usando a técnica de CBR (*Case Based Reasoning System*. Fizeram uma pesquisa por questionário para avaliar a necessidade de uma ferramenta que possibilitasse a automação na escolha de componentes quando a quantidade de opções é de dezenas ou centenas, concluindo que, para poucos componentes, os usuários não se importam em fazer as escolhas manualmente. Entretanto, para grandes quantidades, essa escolha torna-se difícil. Apresenta uma série de trabalhos relacionados à escolha automática de componentes de *software* que utilizam técnicas de algoritmos genéticos e algoritmos evolutivos. Por sua vez, Khan *et al.* (2014) propõem um algoritmo para escolha de componentes armazenados em um repositório que ocorre em duas fases. Na primeira fase, os componentes são desenvolvidos e armazenados, etiquetando-os para diferenciar as características. Na segunda fase, os componentes são buscados para fins de reuso.

Kwong *et al.* (2010) desenvolveram um modelo para escolha de componentes, para cada módulo de componentes, buscando maximizar o desempenho funcional, maximizar a coesão e minimizar o acoplamento dos módulos de *software*. Para alcançar esses objetivos, os autores propõem um algoritmo genético para otimizar o processo de escolha ótima de componentes. Embora esse trabalho permita a seleção ótima, ele parte de critérios subjetivos para definir os

pesos de cada característica, definidos pelos desenvolvedores.

Ainda no sentido de predição de desempenho, etapa usada geralmente em ambientes de escolha de componentes, Lee *et al.* (2015) apresentam um protótipo de arcabouço para modelagem e predição de desempenho automática de *software* a partir de análise estática de códigos escritos em C, através do compilador OpenARC (*Open Accelerator Research Compiler*) e de ferramentas de predição de desempenho de modelos especificados na linguagem Aspen. As ferramentas utilizadas permitem desde a modelagem de desempenho de aplicações executadas puramente em CPU, como também aplicações executadas em GPUs. Uma vez identificado o modelo de desempenho, o resultado é combinado com um modelo abstrato de máquina (*Abstract Machine Model* - AMM) que descreve funcionalidades e parâmetros de uma arquitetura de computador alvo, a qual pode ser definida manualmente ou identificada através da execução de *benchmarks*. É importante ressaltar um recurso adicionado pelo COMPASS que permite inserir modelos de desempenho através de anotações no código. Com base no COMPASS, Obaida *et al.* (2018) apresentam o *PyPassT* um *framework* capaz de modelar o desempenho de computação considerando aspectos de comportamento das memórias, custos de comunicação e execução em aceleradores computacionais.

Calegari *et al.* (2019) apresentam um estudo sobre portais *web* voltados à Computação de Alto Desempenho, onde identificam requisitos divididos em três grupos: funcionalidades obrigatórias, funcionalidades chaves e funcionalidades desejáveis. Dentre as funcionalidades obrigatórias, são listadas: o gerenciamento de tarefas, com a utilização de agendadores de tarefas (*job schedulers*) para controlar as filas de tarefas a serem executadas, possibilitando o controle de ciclo de vida das tarefas; e o gerenciamento de dados, possibilitando serviços para controle do envio e gerenciamento dos dados da aplicação. Por sua vez, dentre as funcionalidades chaves, apresentam-se as seguintes características: *multi-tenancy*, possibilitando compartilhamento de recursos computacionais de forma segura; *multi-cluster*, onde o portal deve gerenciar múltiplos *clusters* simultaneamente; *multi-scheduler*, permitindo suporte a diversos escalonadores de tarefas; multi-diretório, permitindo que vários usuários usem diretórios de arquivos em ambientes compartilhados de forma segura; visualização remota, permitindo que os usuários possam visualizar os resultados remotamente, sem a necessidade de fazer *download* de grandes quantidades de dados; uso de HTTP RESTful API[4], para que o portal permita que a aplicação seja controlada totalmente por serviços *web* através de uma Interface de Programação

[4] do inglês, Application Programming Interface (API)

Tabela 1 – Comparativo de características dos trabalhos relacionados.

Trabalho	Escolha de Software	Escolha de Hardware	Avaliação Qualitativa	Suporte a Heterogeneidade de Hardware	Suporte à Parâmetros Numéricos	Orientação à Componentes
(PANDE *et al.*, 2013)	SIM	NÃO	SIM	NÃO	SIM	SIM
Garg *et al.* (2011a)	NÃO	SIM	SIM	NÃO	SIM	NÃO
Li *et al.* (2010)	NÃO	SIM	SIM	NÃO	SIM	NÃO
Niehorster *et al.* (2010)	NÃO	SIM	SIM	SIM	SIM	NÃO
Buyya *et al.* (2011b)	NÃO	SIM	NÃO	SIM	SIM	SIM
Toosi *et al.* (2018)	NÃO	SIM	SIM	SIM	SIM	SIM
Sandhu *et al.* (2018)	NÃO	SIM	NÃO	SIM	SIM	SIM
Garg *et al.* (2011b)	NÃO	SIM	NÃO	NÃO	SIM	NÃO
Younge *et al.* (2010)	NÃO	SIM	NÃO	NÃO	SIM	NÃO
Buyya *et al.* (2010)	SIM	SIM	SIM	SIM	SIM	NÃO

Fonte: Próprio Autor

de Aplicação sem a necessidade de um navegador, o que permite o controle da aplicação através de um *workflow*. Dentre as funcionalidades desejáveis, são apresentadas: motor de *workflows*, para permitir o controle de execução de múltiplas tarefas com suas dependências e condições; arquivamento de dados dos usuários, possibilitando uma API de armazenamento dos dados resultantes da execução das aplicações dos usuários; gerenciamento de nuvem, possibilitando o uso de recursos de diferentes provedores de nuvens para adicionar recursos à computação. Apresentam também requisitos não funcionais, como segurança, usabilidade, performance e confiabilidade.

A Tabela 1 sumariza os trabalhos relacionados que propõem soluções para escolha de *software* ou *hardware*, eventualmente com suporte à heterogeneidade de *hardware*, tratamento de características qualitativas, valoração numérica de características e se são aplicados em ambientes de computação baseados em componentes. Nessa tabela foram ocultados os trabalhos que apresentam avaliações de alguma das áreas envolvidas neste trabalho, ou os trabalhos de visão geral.

1.1.1 Trabalhos Anteriores

A HPC Shelf foi concebida com base em esforços de pesquisa anteriores, notadamente relativos ao modelo Hash de componentes (JUNIOR *et al.*, 2007) e sua implementação de referência, a plataforma HPE (*Hash Programming Environment*) (CARVALHO-JUNIOR; REZENDE, 2013).

O modelo Hash de componentes busca conciliar as técnicas modernas de engenharia de *software* com as técnicas tradicionais de desenvolvimento de programas paralelos voltados a aplicações com requisitos de HPC. O HPE, sua implementação de referência, é um ambiente de

construção de programas paralelos segundo os conceitos de Programação Paralela Orientada a Componentes (COPP[5]), tendo como alvo a execução sobre plataformas de *cluster computing*. O HPE foi construído de forma a obedecer os padrões de projeto propostos pelo CCA (*Common Component Architecture*), um modelo de componentes desenvolvido para atender os requisitos de aplicações de HPC (CARVALHO-JUNIOR; REZENDE, 2013). De fato, uma contribuição importante do HPE é a definição de um conceito totalmente expressivo de componente CCA paralelo. Embora a HPC Shelf tenha sido concebida visando o modelo Hash, ela também pode ser compatibilizada com outros modelos de componentes como o CCA ou GCM, eventualmente com um conjunto reduzido de funcionalidades. Tanto o modelo Hash quanto o HPE serão melhor detalhados nos capítulos posteriores.

Como já dito anteriormente, a HPC Shelf é um projeto em desenvolvimento, englobando contribuições de vários trabalhos de pesquisa publicados em dissertações de mestrado e teses de doutorado concluídas por membros do grupo de pesquisa ParGO do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará (MDCC/UFC). Anteriores às contribuições introduzidas nesta Tese de Doutorado, as seguintes contribuições à HPC Shelf já foram publicadas:

- A Tese de Doutorado de Jefferson Carvalho Silva (SILVA, 2016), defendida no mês fevereiro do ano 2016, possui contribuições relacionadas ao SAFe (*Shelf Application Framework*), o arcabouço para construção de aplicações da HPC Shelf, posicionando-o como um sistema gerenciador de *workflows* científicos (SWfMS[6]) com características peculiares em relação ao estado-da-arte (SILVA; CARVALHO-JUNIOR, 2016);
- Por sua vez, a Tese de Doutorado de João Marcelo Uchôa de Alencar, defendida no mês agosto do ano 2017, propõe um arcabouço de componentes de computação elásticos para a HPC Shelf, com o propósito de permitir que componentes sejam capazes de reduzir ou aumentar a quantidade de recursos computacionais utilizados da infraestrutura de computação hospedeira, tais como número de processadores, quantidade de memória, largura de banda da rede de interconexão, etc, a fim de conciliar o cumprimento de metas de qualidade estabelecidos em seus contratos e o atendimento a restrições de custos impostos pelos fornecedores dos recursos, representados por componentes das diversas espécies;
- No mesmo mês de agosto do ano 2017, foi defendida a Tese de Doutorado de Cenez Araújo Rezende, introduzindo o Gust, um arcabouço orientado a componentes para processamento

[5] *C*omponent-Oriented Parallel Programming
[6] *Scientific Workflow Management System.*

paralelo em larga escala de grafos grandes sobre a HPC Shelf, o qual toma por base o modelo MapReduce, porém capaz de suportar outros modelos que tem sido propostos especificamente para os requisitos do processamento de grafos, tais como o Pregel, GAS e GraphX;

- Finalmente, a Tese de Doutorado de Allberson Bruno de Oliveira Dantas, defendida no mês outubro do ano 2017, propõe um arcabouço de certificação de componentes para a HPC Shelf, instanciando, como estudos de caso, componentes certificadores específicos para componentes de computação (C4[7]) e componentes *workflow* (SWC2[8]).

1.2 Objetivos

O objetivo principal deste trabalho é a concepção e desenvolvimento de um sistema de contratos contextuais, possibilitando em um ambiente de execução orientado a componentes, a escolha dos componentes de uma aplicação, os quais representam os recursos de *software* e *hardware* que compõem os sistemas de computação paralela oferecidos como serviços, tudo isso sob a abstração de nuvem computacional. Os critérios de escolha são baseados em características de qualidade e desempenho representadas por acordos de nível de serviço.

Para alcançar esse objetivo, alguns objetivos específicos foram traçados, os quais estão elencados a seguir:

- Levantamento das características de *hardware* presentes nos computadores que mais se aproximam dos recursos onde a HPC Shelf será executada, ou seja, em um grupo dos supercomputadores mais poderosos;
- Definição da estratégia de escolha de sistemas computacionais, bem como a classificação da lista resultante;
- Definição dos critérios de desempenho a serem considerados na definição dos acordos de nível de serviço entre os intervenientes da HPC Shelf e o próprio ambiente da nuvem, bem com o levantamento dos objetivos de nível de serviço que devem ser considerados;
- Implementação do componente *Core* da HPC Shelf, além da definição e implementação dos serviços que estarão disponíveis ao *Frontend*;
- Estudos experimentais de avaliação do mecanismo de alocação de recursos proposto, através do protótipo desenvolvido.

[7] *Computation Component Certifier Component*.
[8] *Scientific Workflow Certifier Component*.

1.3 Estrutura da Tese

Além deste capítulo introdutório, esta Tese de Doutorado possui cinco outros capítulos, cujos propósitos são explicados nos próximos parágrafos.

O Capítulo 2 apresenta o referencial teórico deste trabalho, começando pela introdução à área de Computação Alto Desempenho, seguido pelas áreas de computação em nuvens e engenharia de *software* baseada em componentes. Uma das principais preocupações neste capítulo é a discussão a cerca das interseções entre essas áreas, a fim de contextualização e delimitação do escopo em que o Alite foi concebido. Neste capítulo, é apresentado ainda o modelo Hash de componentes e seu sistema de tipos, o HTS (*Hash Type System*), o qual é a base para o desenvolvimento deste trabalho.

Por sua vez, o Capítulo 3 introduz a plataforma HPC Shelf, definindo suas espécies de componentes, seus intervenientes e seus elementos arquiteturais, bem como as relações entre eles.

O Capítulo 4 é apresenta o Alite, principal contribuição desta Tese, introduzindo-se seus conceitos e abstrações, bem como a estratégia geral e algoritmos responsáveis pela seleção de componentes.

No Capítulo 5 são definidos os estudos de caso para a avaliação do Alite. Nesse capítulo também é definida a metodologia de validação do protótipo e os resultados da validação são discutidos.

Por fim, no Capítulo 6 são apresentadas as considerações finais da Tese e são apresentados os trabalhos identificados para serem desenvolvidos no futuro.

2 CONTEXTO E REFERENCIAL TEÓRICO

O sistema de contratos contextuais da HPC Shelf, produto deste trabalho, pode ser visto como o mecanismo responsável pela seleção de que componentes estão disponíveis para solução computacional de problemas, notadamente com requisitos de computação intensiva, incluindo as próprias plataformas de computação paralela que serão empregadas para execução dessa solução computacional. O objetivo é maximizar a eficiência do uso dessas plataformas, bem como atender a cronogramas de execução, considerando critérios definidos de maneira formal para o ambiente da HPC Shelf.

A plataforma HPC Shelf fundamenta-se na convergência de três métodos de desenvolvimento de software e computação:

- Computação paralela;
- Computação em nuvens;
- Engenharia de software baseado em componentes (CBSE[1]).

Tais métodos são reunidos dentro de um mesmo arcabouço a fim de oferecer uma plataforma para desenvolvimento de aplicações de HPC cujos requisitos levam a demanda por técnicas de desenvolvimento de larga escala e o emprego de múltiplas plataformas de computação paralela, notadamente provenientes das ciências e das engenharias.

Neste capítulo, são apresentados os principais conceitos por trás de plataformas de computação paralela, nuvens computacionais e desenvolvimento baseado em componentes, com o ênfase especial em posicioná-los dentro do contexto dos requisitos de aplicações de HPC.

2.1 Computação Paralela

A Computação de Alto Desempenho (HPC) é a área da computação dedicada ao estudo e desenvolvimento de aplicações e tecnologias que permitem a resolução dos problemas que demandam por alto poder de processamento, para que, em tempo hábil, o seu resultado possa ser útil. Os ambientes de execução das aplicações de interesse de HPC utilizam tecnologias que aglomeram unidades de processamento, sejam computadores completos, processadores ou núcleos de processamento de processadores, sendo tais aplicações especificamente desenvolvidas para explorar o paralelismo potencial nesses ambientes, de forma a entregar grande capacidade de processamento.

[1] Component-Based Software Engineering

Um fator chave para a HPC nos dias atuais é a diversidade de plataformas específicas para a execução de aplicações com requisitos de computação intensiva. Portanto, a escolha da plataforma tem grande peso na qualidade do sistema computacional que irá resolver o problema, cuja solução motiva o interesse do usuário por técnicas de HPC. Devido à diversidade do *hardware* para se obter o melhor desempenho para cada caso, é importante compreender a evolução da arquitetura de plataformas de computação paralela dentro do contexto de HPC, ao longo do tempo.

2.1.1 Plataformas de Computação Paralela (Histórico)

O computador CDC 6600, inicialmente desenvolvido pela empresa *Control Data Corporation* (CDC), de Seymour Cray, considerado o precursor da supercomputação, é considerado o primeiro computador comercial com capacidade de processamento paralelo, através de técnicas de processamento superescalar, ou seja, a execução de várias instruções simultaneamente (NULL; LOBUR, 2010). Foi desenvolvido, na primeira metade dos anos 1960, para atender aplicações de interesse científico, inicialmente implantado para atender o CERN (*European Organization for Nuclear Research*), em 1965, e, posteriormente, o *Lawrence Radiation Laboratory* da Universidade de Berkely, nos EUA.

Embora muitos considerem o CDC 6600 como o primeiro supercomputador, a implantação do computador Cray-1 no LANL (Los Alamos National Laboratory), em 1976, é considerada o marco inicial mais significativo para a supercomputação moderna. O Cray-1 foi fabricado pela empresa Cray Research, também de propriedade de Seymour Cray, sob contrato com o governo dos EUA. Essa plataforma inaugurou o uso efetivo da computação vetorial para os propósitos da supercomputação, somando-se às técnicas superescalares, levando-a a um nível de desempenho que superava outros computadores da época em uma ordem de magnitude, notadamente para aplicações científicas às quais foi destinada. Isso era possível pelo fato de que tais aplicações poderiam usufruir de técnicas de computação vetorial, de arquitetura SIMD (*Single Instruction Multiple Data*), onde uma mesma instrução era aplicada sobre registradores que representavam conjuntos homogêneos de dados, ou seja, vetores ao invés de valores escalares.

Ainda sob contratos com o governo dos EUA, a Cray Research inaugurou uma linha de supercomputadores de grande sucesso durante os anos 1980, os quais exploravam o processamento vetorial combinado com o multiprocessamento, através do emprego de várias unidades de processamento vetorial, equipadas com memória *cache*, as quais compartilhavam um

espaço de memória comum. Plataformas que seguiam essa arquitetura, tais como X-MP (1983) e Y-MP (1988), ficaram conhecidas como *Crayettes*. Durante os anos 1980, a NEC, empresa japonesa, constituiu a principal concorrente da Cray no mercado global de supercomputação vetorial, levando a uma intensa disputa comercial.

Em paralelo ao desenvolvimento de supercomputadores para aplicações científicas, o desempenho dos processadores de propósito geral, comumente usados para aplicações comerciais, crescia rapidamente em capacidade de processamento, seguindo a previsão de Gordon E. Moore. Segundo Moore, co-fundador da *Intel*, o número de transistores de um processador, que mostra-se proporcional ao seu desempenho, teria um aumento de 100%, pelo mesmo custo, a cada período de 18 meses (NULL; LOBUR, 2010; MOORE, 1965; MONTEIRO, 2002). De fato, as indústrias que fabricavam esses processadores seguiam investindo esforço de pesquisa e desenvolvimento para oferecer produtos com desempenho crescente, em escala exponencial, motivadas pela demanda das novas aplicações e a concorrência entre si. Desse modo, a Lei de Moore continua a vigorar até os dias atuais (STALLINGS, 2003), a despeito das dificuldades físicas em aumentar o desempenho de processadores sequenciais. Inicialmente, o aumento de desempenho era perseguido através do incremento no número de transístores em um único processador, bem como do aumento da sua frequência (*clock*), o que motivava a pesquisa voltada ao desenvolvimento de técnicas para que os processadores fossem capazes de executar múltiplas instruções de uma mesma sequência de instruções ao mesmo tempo (paralelismo implícito) e obter proveito dessas características (NAKAJIMA; PALLIPADI, 2002). Assim, técnicas como o uso de *pipelines* e execução *superescalar*, antes restritas a supercomputadores de alta tecnologia, tornaram-se onipresentes na arquitetura de processadores de uso geral (STALLINGS, 2003).

Durante os anos 1980, a rápida evolução do mercado de processadores sequenciais, onde novos produtos, de desempenho muito superior aos anteriores, eram lançados em espaços de tempo muito curtos, bem como a evolução das tecnologias de interconexão de processadores em rede, levou a situação na qual tornou-se viável a construção de supercomputadores de memória distribuída, construídos com processadores de propósito geral, de prateleira, interligados por meio de interconexões proprietárias de alta velocidade. Tais arquiteturas, ditas MPPs (*Massive Parallel Processors*) passaram a rivalizar com supercomputadores vetoriais no restrito mercado de supercomputação, com a vantagem de serem arquiteturas mais escaláveis, por serem de memória distribuída. Dessa forma, uma MPP poderia ter seu desempenho aumentado acrescentando-se mais unidades de processamento de maneira mais simples do que no caso de supercomputadores

vetoriais de arquitetura Cray. No final dos anos 1980, as MPPs já dominavam o mercado da supercomputação, relegando os supercomputadores vetoriais a uma posição secundária no mercado até praticamente desaparecerem durante os anos 1990.

Também no final dos anos 1980, segundo Anderson *et al.* (1995), por motivos semelhantes a aqueles que levaram ao surgimento de MPPs, laboratórios científicos enxergaram a viabilidade do emprego de redes de estações de trabalho (NOWs[2]) para finalidades de aplicações de HPC. Isso foi possível devido a rápida evolução dos processadores empregados em estações de trabalho individuais, bem como das tecnologias de prateleira para a sua interconexão em redes de computadores. Uma vez que a construção de MPPs capazes de usar as últimas versões de um certo processador mais moderno demorava um certo tempo (*lag time*), e tendo em vista o tempo necessário para desenvolvimento de tecnologias de integração, como as interconexões proprietárias, redes de estações de trabalho, formadas com os mais recentes processadores e interconectadas por redes Ethernet, as NOWs mostravam-se viáveis para muitas aplicações menos intensivas em comunicação.

Além das NOWs, surgiram os *clusters* formados por computadores pessoais (PCs) interligados por rede Ethernet, inaugurando a classe de arquiteturas de computação paralela que ficou conhecida como *cluster computing*. Destaca-se como marco da computação em *clusters* o projeto Bewoulf, desenvolvido pela equipe de Donald Becker na agência espacial NASA, quando um conjunto de computadores pessoais fora de uso foram empregados para aplicações de dinâmica dos fluidos (CFD[3]), de interesse da NASA, após o desenvolvimento de *software* de suporte pela sua equipe Becker *et al.* (1995).

Desde o final dos anos 1990, plataformas de *cluster computing* lideram o principal *ranking* de arquiteturas de computação paralela, seguido por MPPs. Esse *ranking*, que lista os 500 computadores paralelos de maior desempenho, segundo o *benchmark* LINPACK, é conhecido como Top500, sendo acessível por meio do link <http://www.top500.org>. A iniciativa Top500 é mantida por instituições acadêmicas e indústrias, recebendo atualizações desde 1993. A lista permite observar a evolução da arquitetura dos supercomputadores desde o início dos anos 1990, quando os supercomputadores vetoriais ainda dominavam o contexto da supercomputação, até os dias atuais, quando predominam *clusters* e MPPs.

Com o objetivo de garantir maior eficiência energética dos processadores, bem como tornar ainda possível o incremento do desempenho dos processadores apesar da exaustão das

[2] *Networks of workstations*.
[3] *C*omputational Fluid Dynamics.

Tabela 2 – Núcleos de Processamento (cores) por processador nos supercomputadores do Top 500 (Novembro/2018), soma-se 100% o total dessas quantidades, evidenciando a ausência de computadores com processadores de apenas um núcleo.

Quantidade de Núcleos por soquete	% da quantidade de sistemas
6	1,2
8	5,6
10	5,8
12	13,4
14	8,8
16	19,8
18	12,2
20	22,2
22	1,4
24	4,4
28	0,2
32	1,2
64	1,4
68	2,2
260	0,2

Fonte: Próprio autor.

técnicas tradicionalmente empregadas, a indústria de processadores optou pela exploração do processamento paralelo explícito baseado em *threads* dentro do mesmo processador. Essa tecnologia, proposta inicialmente pela *Intel*, foi denominada *Hyper Threading*, onde são adicionados registradores adicionais para armazenar o contexto completo de um processo em execução no processador, e assim, promovendo a visão lógica de dois processadores por parte do sistema operacional. Porém, de fato, só existe uma única unidade de execução das instruções, disputada pelos dois processadores lógicos. Essa duplicação otimiza o uso do processador por permitir que em momentos que ocorram situações de ociosidade no *pipeline*, o contexto do processo seja substituído pelo contexto do outro processador lógico (NAKAJIMA; PALLIPADI, 2002).

Posteriormente, as unidades lógicas e aritméticas do processador foram replicadas totalmente, em duas ou mais, constituindo núcleos de processamento independentes, capazes de executar *threads* em paralelismo real. Atualmente, a grande maioria dos processadores de prateleira, mesmo de dispositivos móveis simples, já possuem de 4 a 68 núcleos de processamento, sendo portanto uma tecnologia já consolidada.

Atualmente, todas as arquiteturas modernas de computadores paralelos empregam processadores com vários núcleos, introduzindo múltiplas hierarquias de paralelismo e memória que não podem ser trivialmente utilizadas com eficiência pelas aplicações. A Tabela 2 apresenta

o número de núcleos de processamento por processador nos computadores listados recentemente no Top500, onde nota-se uma predominância de 8 núcleos ou mais, o que é refletido pelo maior desempenho combinado dos computadores que possuem mais núcleos, ou seja, os de arquitetura mais recente e que ocupam as primeiras posições da lista.

Uma outra tendência no cenário das arquiteturas de computadores, com impacto igualmente significativo em plataformas de computação paralela, são os aceleradores computacionais, representados por FPGAs (*Field Programmable Gateway Arrays*) (HERBORDT *et al.*, 2007), MICs (*Many Integrated Core*) (DURAN; KLEMM, 2012) e GPUs (*Graphic Processor Units*) (FAN *et al.*, 2004).

GPUs são unidades de processamento desenvolvidas para processamento gráfico, mas que mostraram-se úteis para aceleração de computações em aplicações típicas de HPC, especialmente de computação numérica sobre vetores e matrizes. Para isso, empregam uma grande quantidade de núcleos de processamento de pequena capacidade capazes de executar instruções de forma síncrona, como em arquiteturas SIMD tradicionais. Tais núcleos de processamento operam sobre várias hierarquias de memória, sendo crucial para obter um melhor desempenho um controle fino sobre a localidade dos dados em processamento.

Por sua vez, a arquitetura MIC foi anunciada pela *Intel* em Maio de 2010, como alternativa às GPUs. Esse tipo de acelerador difere das GPUs por ser composto por muitos núcleos limitados da arquitetura *x*86 em uma única pastilha de silício (POHL, 2012). Recebeu o nome comercial de *Xeon Phi*, e suporta linguagens comumente utilizadas em processadores de uso geral, como C, C++ e Fortran. Por consequência, argumenta-se sobre a sua compatibilidade com programas paralelos legados de memória compartilhada, escritos nessas linguagens (DOERNER, 2012). Além do *Xeon Phi*, uma empresa Japonesa chamada Pezy Computing tem desenvolvido uma família de aceleradores computacionais chamados Pezy-SC (Pezy Super Computing) (Pezy Super Computing, 2015). Em sua primeira versão, lançada em 2014, era composto por 1024 núcleos, capaz de alcançar um desempenho teórico de $1,5$ TFlops em operações de ponto

Tabela 3 – Participação de Aceleradores Computacionais no Top 500

	Nov/2013	Nov/2014	Nov/2015	Nov/2016	Nov/2017	Nov/2018	Jun/2019
GPU	7,8%	9,8%	13,8%	12,2%	17,0	25,2%	25,2%
MIC	2,4%	4,0%	5,8%	4,2%	2,0	1,2%	1,0%
IBM Cell	0,0%	0,0%	0,0%	0,0%	0,0	0,0%	0,0%
PEZY-SC	0,0%	0,2%	0,4%	0,2%	1,0	0,4%	0,0%
MATRIX-2000	0,0%	0,0%	0,0%	0,0%	0,0	0,2%	0,2%
Hibrido	0,4%	1,0%	0,8%	0,6%	0,4	0,4%	0,2%
Total	10,6%	15,0%	20,8%	17,2%	20,4	27,4%	26,6

Fonte: Próprio autor.

flutuante de precisão dupla e $3,0$ TFlops em precisão simples. Suas versões posteriores, Pezy-SC2 (2017) e Pezy-SC3 (2019), suportam respectivamente 2048 e 8192 núcleos. Essa última versão tem sido desenvolvida para equipar supercomputadores da série ZettaScaler-3.

A Tabela 3 apresenta a evolução da presença de aceleradores de arquitetura MIC e GPU nos computadores listados no Top500 nos últimos anos, notando-se uma crescente utilização de aceleradores MIC em contraposição a uma estabilização no crescimento do uso das GPUs até Nov/2015, seguido de um declínio até seu quase desaparecimento de acelereadores MIC, devido a descontinuidade do desenvolvimento do Xeon Phi pela Intel. Os dados também sugerem o forte impacto de aceleradores GPU e MIC no desempenho das plataformas, a julgar pela diferença entre a participação em termos de número de computadores do ranque que os suportam e em termos de desempenho combinado. Nota-se que esse fato é bastante exacerbado a partir da lista de Novembro de 2013 com relação aos aceleradores MIC. Em 2014, surge na lista os aceleradores Pezy-SC, com uma participação modesta até o momento e em 2018 surge o acelerador Matrix-2000, também com participação modesta.

Aceleradores computacionais introduzem técnicas peculiares para o desenvolvimento das aplicações, demandando conhecimento específico para que um programador seja capaz de explorar o seu potencial de desempenho. Por exemplo, GPUs devem ser programadas com interfaces de programação específicas, tais como CUDA (COOK, 2012) ou OpenCL (STONE *et al.*, 2010). Por outro lado, MICs adotam interfaces de programação semelhantes àqueles adotados para programação paralela de memória compartilhada, como OpenMP, aplicado tanto em multiprocessamento quanto em processamento de múltiplos núcleos dentro de um mesmo processador. Entretanto, a grande quantidade de processadores e hierarquias de memória presentes nesse tipo de acelerador faz com que técnicas de programação paralela específicas devam ser conhecidas e aplicadas pelos programadores.

Além da complexidade inerente à programação e distribuição de computações que devem ser executadas no acelerador, há a complexidade do emprego de múltiplos núcleos de processamento em um mesmo processador, bem como múltiplos processadores em um mesmo nó de processamento, ao qual uma ou mais GPUs e/ou MICs podem estar conectados. Além disso, há a necessidade de uso de interfaces de programação diferentes para sincronização eficiente entre os nós de processamento distribuídos e entre os núcleos de processamento que compartilham memória dentro de um mesmo nó. Devido a isso, o uso extensivo de aceleradores em plataformas de computação paralela de alta tecnologia tem motivado a ênfase no conceito

de *computação heterogênea*, referindo-se à heterogeneidade arquitetural dessas plataformas. Tal heterogeneidade reflete-se nas diversas técnicas de programação que precisam ser aplicadas para que o poder de processamento de plataformas de computação paralela modernas possa ser utilizado na sua plenitude.

As aplicações de HPC costumam ser fortemente dependentes das plataformas de computação paralela sobre as quais executam, devido às otimizações feitas pelos desenvolvedores para extrair o seu potencial máximo, explorando as suas características arquiteturais e heterogeneidade. Assim, o desenvolvimento dessas aplicações costuma não utilizar técnicas de programação que envolvam maior nível de abstração, as quais são incentivadas pela engenharia de *software* moderna, a fim de evitar sobrecargas na aplicação (SANDOVAL; COOPER, 2009).

A heterogeneidade em plataformas de computação paralela acarreta uma grande variedade de linguagens e técnicas de programação. Portanto, torna-se pertinente qualquer esforço de disseminar o uso de novas plataformas de computação paralela, bem como os diversos tipos de aceleradores computacionais, por usuários que não sejam especializados em coprocessamento ou processamento específico dessas novas plataformas. Os objetivos a serem alcançados nesse trabalho de pesquisa estão engajados nesses esforços, através de contribuições na convergência de técnicas discutidas nas próximas seções: componentes (Seção 2.5) e nuvens computacionais (Seção 2.2).

2.2 Computação em Nuvens

A computação em nuvens é um modelo de provimento de execução de aplicações sob demanda, inicialmente desenvolvido para aplicações comerciais, e que só foi viabilizado após uma sucessão de técnicas que permitiram o compartilhamento de computadores de forma fácil, segura e relativamente eficiente. É o principal representante do conceito de computação como utilidade, na qual recursos que oferecem poder de processamento e armazenamento de plataformas de computação são disponibilizados de forma elástica por provedores de serviços, de modo que recursos podem ser adicionados ou removidos a qualquer momento, mantendo ainda transparência de localização. Assim, como no caso da energia elétrica, provedores de recursos são pagos somente pelo que foi consumido pelo usuário. Potencialmente, essa abordagem torna o custo dos recursos de computação mais acessível aos usuários (FOSTER *et al.*, 2008), uma vez que, se utilizassem equipamentos próprios, teriam de lidar com períodos de ociosidade que gerariam gastos de manutenção desnecessários, além dos custos empregados na aquisição. O

modelo de tarifação de nuvens computacionais é conhecido como *pay-as-you-go* e torna-se atrativo pelo fato de não se pagar pelos custos de manter equipamentos próprios (CHOO, 2010).

A computação em nuvens tem se tornado foco de diversos trabalhos de pesquisa de origem tanto na academia quanto na indústria, muitos dos quais buscam desenvolver um ambiente de nuvem computacional que se qualifique como um padrão de fato. No entanto, a comunidade científica não chegou a um consenso nem mesmo no próprio conceito de nuvem computacional, existindo diferentes definições. Ciente disso, Vaquero *et al.* (2008) pesquisou e classificou definições de nuvens computacionais, limitando-se a vinte definições, dentre as quais julgamos como mais relevante a seguinte:

> Nuvens são um largo conjunto de recursos virtualizados (como *hardware*, plataformas de desenvolvimento e/ou serviços) de modo fácil e acessível. Esses recursos podem ser reconfigurados dinamicamente para se ajustar a uma variação de carga (escala), também permitindo uma utilização ótima dos recursos. Esse conjunto de recursos é tipicamente explorado no modelo de pagamento sob demanda, onde garantias são oferecidas por um provedor de infraestrutura através de um Acordo de Nível de Serviço (SLA) personalizado.

De acordo com essa definição, há uma abstração de virtualização que oculta algumas especificidades do *hardware*, para tornar o uso dos recursos mais fácil, em comparação com o gerenciamento do *hardware* nativo. Além disso, trata a utilização dos recursos como ótima, pois através da variação de carga, pode-se variar a quantidade de recursos disponível, eliminando, ou pelo menos reduzindo em muito, o tempo de ociosidade do equipamento. Adiciona-se a este último fator, a escolha do *hardware* virtual mais adequado ao perfil da aplicação. Ainda definem que o pagamento será feito somente pela quantidade de recurso realmente utilizada, e esse modelo financeiro sendo regido por um contrato chamado SLA, onde são definidos todos os requisitos que o provedor de nuvem deve cumprir na disponibilização dos recursos computacionais.

Além desta definição, existem outras características que são relacionadas ao conceito de nuvem computacional, conforme Sousa *et al.* (2009) apresentam, tais como:

- *Self-service*: permite que usuários adicionem ou removam recursos computacionais de maneira fácil e de modo automático no lado da nuvem.
- *Amplo Acesso*: refere-se à facilidade de acesso aos recursos que estão sendo executados na nuvem, de modo que sejam acessíveis por um *web browser*.

- *Pooling*: permite o compartilhamento dos recursos entre os usuários de forma suficiente e transparente, permitindo, no máximo, escolhas no nível mais abstrato, como a localização do centro de dados.
- *Elasticidade Rápida*: a capacidade do usuário de adicionar ou remover recursos de sua aplicação de forma automática e rápida.
- *Serviço Medido*: permite a tarifação de forma transparente para que o usuário tenha conhecimento dos recursos utilizados, e ainda considerando a avaliação da qualidade de serviço, quando disponibilizada.

Em especial, a elasticidade é um requisito que torna o uso de nuvens computacionais atrativo, pois o ambiente de execução pode se adequar a carga da aplicação no momento atual da execução. E elasticidade pode ocorrer de duas maneiras distintas e não exclusivas, a elasticidade vertical e a horizontal. Na elasticidade vertical, os recursos aumentados ou reduzidos referem-se à quantidade de recursos em um único nodo. Já na elasticidade horizontal, criam-se novas instâncias de nodos para dividirem os trabalhos ou para aumentar a disponibilidade de um sistema (HUANG *et al.*, 2013).

As nuvens computacionais podem ter restrições de acesso aos recursos ou ainda restrição à visão do usuário na sua utilização. Na analogia feita com a distribuição de energia elétrica, é caracterizado o modelo de implementação de nuvem chamado de nuvem pública. Além dessas, existem também as nuvens privadas, em que o poder computacional está disponível dentro dos limites do proprietário da nuvem; as nuvens comunitárias, operadas por um conjunto de instituições; e as nuvens híbridas, que se caracterizam pela composição de duas ou mais nuvens (privadas, públicas, comunitárias) (ZHANG *et al.*, 2010).

A classificação apresentada refere-se à visibilidade dos recursos, restringindo o público alvo dos serviços. Porém, em uma nuvem computacional, existem diversos níveis de abstração disponíveis para os usuários, ficando a cargo de cada provedor de recursos especificar quais tipos são oferecidos. No mais alto nível de abstração, o usuário somente utiliza as aplicações disponibilizadas em um ambiente de nuvem computacional. No nível inferior, o usuário trabalha no nível de máquinas virtuais. Já no nível intermediário, o usuário abstrai as características da plataforma de execução e desenvolve suas aplicações a partir de uma biblioteca específica. Esses níveis de abstração são classificados pela maioria dos autores como *Software*-como-Serviço (SaaS), Plataforma-como-Serviço (PaaS), Infraestrutura-como-Serviço (IaaS), e Dados-como-Serviço (DaaS) (SOUSA *et al.*, 2009).

No modelo SaaS, o usuário não desenvolve ou instala nenhum tipo de aplicação, somente utiliza as aplicações disponibilizadas. No modelo PaaS, o usuário tem parte do trabalho simplificada, abstraindo-se do sistema básico e das características da máquina virtual (infraestrutura) e concentrando-se na utilização de linguagens e APIs próprias do provedor de serviço de nuvem para desenvolver aplicações (SOUSA *et al.*, 2009). No modelo IaaS, o usuário interage com computadores virtuais, como se estivesse utilizando um computador local, necessitando instalar e configurar todo o *software* presente no computador, incluindo o sistema operacional (SOUSA *et al.*, 2009). Por fim, no modelo DaaS, que é o tipo de nuvem mais utilizado atualmente, as aplicações na internet utilizam serviço de armazenamento de arquivos em seus próprios *data-centers*, servindo como um serviço de nuvem de armazenamento pela visão dos usuários (SOUSA *et al.*, 2009).

Recentemente, uma nova classificação surgiu, com a mesma visão do IaaS, no entanto com diferenciação de tarifação e alocação de recursos, sendo chamada de Recurso-como-Serviço (RaaS) (BEN-YEHUDA *et al.*, 2012). Nesse novo modelo, os recursos serão tarifados somente pela parcela em que realmente são usados, por exemplo pela quantidade de memória e de dispositivos de I/O, além do processador específico, e a unidade de tempo para cobrança passará de horas para ciclos de processador, ou até mesmo segundos, de modo a não haver desperdício de recursos. Além da maneira de tarifar os recursos, a alocação dos recursos deve prever que facilmente recursos possam ser adicionados ou removidos, preferencialmente na mesma máquina física. Para possibilitar essa afinidade na alocação, SLAs devem permitir que sejam definidas implicitamente as prioridades entre os processos dos usuários, podendo suspender ou reduzir processos em execução de um usuário com menor prioridade. Esse modelo foi definido de maneira teórica, não existindo até o momento a implementação do mesmo, ficando como uma área de pesquisa em aberto, com uma série de desafios identificados.

A plataforma de nuvem computacional disponível sob o modelo de nuvem pública que apresenta maior destaque é a *Amazon Web Services* (AWS), que, dentre um grande conjunto de serviços, disponibiliza um serviço pioneiro de IaaS chamado *Amazon Elastic Cloud Computing* (EC2). Além do EC2[4], outros exemplos de serviços IaaS são o Google Cloud Platform (GCP)[5], o IBM Cloud[6], o Rackspace[7], o CloudSigma[8], o Numergy[9], dentre outros. Já no modelo de PaaS,

[4] http://ec2.amazon.com
[5] http://cloud.google.com
[6] http://cloud.ibm.com
[7] http://www.rackspace.com
[8] http://www.cloudsigma.com
[9] https://www.numergy.com

destacam-se os provedores Google App Engine[10], Microsoft Azure e SalesForce[11] (KäCHELE *et al.*, 2011). Nos modelos SaaS e DaaS, exitem muitas aplicações, mas, de modo geral, são aplicações disponibilizadas para os usuários finais, tais como o Dropbox e as diversas aplicações disponibilizadas na nuvem da Google, tais como Google Documents[12], Google Drive[13], etc, e Amazon[14].

Além desses serviços, onde o *hardware* utilizados é terceirizado, ainda há a possibilidade de se utilizar as tecnologias envolvidas nos ambientes de nuvens computacionais públicas em equipamentos próprios, ou seja, a instalação de nuvens privadas. Para nuvens privadas, destacam-se Open Nebula [15], OpenStack [16] e Eucalyptus [17], dentro do modelo IaaS (PEPPLE, 2011; WOLSKI, 2012). Já para o modelo PaaS, destacam-se Aneka (BUYYA *et al.*, 2011a), Stackato [18] e CloudBees[19].

Em geral, os ambientes de nuvens privadas de infraestrutura apresentam compatibilidade com nuvens públicas de uso mais disseminado, tais como o AWS e GCP.

No contexto desta Tese, utliliza-se a nomenclatura HPCaaS (*HPC-as-a-Service*) para caracterizar a disponibilização, em nuvem, de recursos de HPC de maneira simplificada aos usuários finais, de modo que as particulares arquiteturais das plataformas não necessitem ser conhecidas pelos usuários, os quais geralmente não estão capacitados a decisões com base em tais particulares (HUANG *et al.*, 2013). Geralmente, em ambientes de execução de aplicações de HPC, os usuários preocupam-se em informar a quantidade de recursos que gostariam de utilizar na execução, mas tomar essa decisão não é uma tarefa simples, pois depende de como a aplicação foi modelada e desenvolvida. Para esses usuários, o importante é que, o quanto antes a execução terminar, melhor. O problema se agrava quando adiciona-se a heterogeneidade trazida por equipamentos de diferentes centros computacionais e dos *hardware* de aceleração computacional.

Embora as nuvens computacionais estejam cada vez mais ocupando espaço antes dominado por *clusters* de computadores nas próprias instituições, ainda existe uma resistência

[10] https://appengine.google.com
[11] http://www.salesforce.com
[12] http://docs.google.com
[13] http://drive.google.com
[14] https://aws.amazon.com
[15] http://opennebula.org
[16] http://www.openstack.org
[17] http://www.eucalyptus.com
[18] http://www.activestate.com/stackato
[19] http://www.cloudbees.com

por parte dos usuários de HPC (AHUJA; MANI, 2012). Parte dessa aversão se explica pela sobrecarga de desempenho, inerente da virtualização de recursos físicos de *hardware* bem como ao fato dos usuários das aplicações de alto desempenho, de modo geral, estarem acostumados com controle total sobre as máquinas.

2.3 Computação de Alto Desempenho em Nuvens

Nesta seção, apresenta-se mais detalhes a respeito da HPC como serviço, chamada de *HPC-as-a-Service*, onde é feita uma descrição das suas características, associado com um levantamento do estado-da-arte desses tipos de serviço. Por fim, serão apresentadas as classificações quanto aos critérios que são usados para escolhas de recursos em um serviço de HPCaaS.

Aplicações de computação podem ser classificadas de acordo com diversas dicotomias, como determinísticas ou não-determinísticas, síncronas ou assíncronas, sequenciais ou concorrentes, no entanto, no contexto da diferenciação de aplicações comerciais e de HPC, é relevante a introdução da dicotomia definida por aplicações transformacionais ou reativas. As aplicações transformacionais recebem dados no início da execução e resultam em uma saída ao final de sua execução, já as reativas não tem por objetivo obrigatoriamente gerar uma saída, nesse caso, as interações da aplicação são o objetivo da mesma. Nas aplicações reativas é comum que nunca terminem, pois os resultados são obtidos em interações breves (HAREL; PNUELI, 1985). Assim, podemos classificar as aplicações do cenário comercial, aplicações que geralmente são caracterizadas por processos interativos entre os usuários e o ambiente de execução como reativas. Um exemplo é um servidor de *web site*, no qual a carga varia de acordo com o número de acessos simultâneos, e a execução do programa nunca termina. Já as aplicações científicas, são transformacionais, pois caracterizam-se de forma semelhante ao processamento em lote (*batch*), no qual uma aplicação começa sua execução e o tempo total para retornar o resultado depende inteiramente da entrada do programa (GARG *et al.*, 2011b).

Em geral, as aplicações de HPC são executadas em plataformas de computação paralela, as quais apresentam *hardware* de alta capacidade de processamento e armazenamento, bem como redes de interconexão de baixa latência, tais como Infiniband e Myrinet, evitando ao máximo a incorporação de custos computacionais de tarefas de gerenciamento da execução paralela (MASUD, 2011). Atualmente, segundo o site Top500.org (2019), as principais classes de plataformas de computação paralela são as MPP (*Massively Parallel Processors*) e os clusters de classe *I* e *II*.

O *hardware* de alto desempenho apresenta custo relativamente elevado, o que no caso de entidades que possuem a necessidade da execução da aplicação de HPC em um tempo satisfatório e possuem grandes orçamentos, não é problema. Apesar do custo dos *clusters* de classes *I* e *II* ser relativamente baixo, devido à utilização de *hardware* de prateleira, esse custo ainda é elevado para uma quantidade considerável de instituições.

As aplicações executadas nos *clusters* são necessariamente paralelas, a fim de poder otimizar a exploração do poder computacional do conjunto de computadores. Logo, há a necessidade de sincronização de dados de uma unidade de processamento (nodo/nó) para outra, e em geral, isso ocorre através de passagem de mensagens. Além da comunicação entre os nodos, a comunicação pode ser feita em um único nodo que possua vários núcleos de processamento. Nesse caso, a troca de dados costuma ser através da memória compartilhada entre os núcleos (MASUD, 2011; NAPPER; BIENTINESI, 2009).

Os ambientes de nuvens computacionais voltados para aplicações científicas, diferentemente daqueles voltados a aplicações comerciais, possuem requisitos diferenciados, onde destacam-se os requisitos mais severos de desempenho e, consequentemente, de uso eficiente dos recursos computacionais. Os usuários desse perfil geralmente adquirem um *cluster* próprio apenas para executar o seu processamento, fator que dificulta a migração para as nuvens devido às potenciais penalidades de desempenho advinda do uso desses ambientes (AHUJA; MANI, 2012).

Quando se trata de aplicações de HPC, a elasticidade pode ser usada para antecipar o resultado de uma execução, que pode chegar a dias e também pode ser usada para efetuar a manutenção de contrato (SLA), de modo a suprir flutuações de carga dos ambientes de nuvens computacionais.

Com base nas diferenças entre aplicações comerciais e as de HPC, Alencar (2017), em sua Tese de Doutorado, assim definiu a elasticidade para sistemas HPC:

> *Elasticidade em Computação de Alto Desempenho é a capacidade de um ambiente que garante a gerência adaptativa de recursos computacionais com o objetivo de aprimorar o desempenho de uma aplicação, considerando as limitações de escalabilidade dos algoritmos utilizados e os requisitos não funcionais do usuário.*

Como as aplicações científicas tem o seu funcionamento interno paralelizado, não é possível simplesmente reduzir ou aumentar a quantidade de nós para se beneficiar dos ambientes

de execução paralela, pois geralmente a divisão das tarefas é feita estaticamente antes do início da execução. De mesmo modo, a escalabilidade horizontal não é nativamente explorada, pois muitas aplicações são conscientes das quantidade de recursos de um nó antes do início da execução para otimizar o uso de hierarquias de memórias, e, caso haja aumento de recursos, essas otimizações serão possivelmente perdidas. Por exemplo, considere uma tarefa já está dividida para executar usando 10 nós. De nada adiantará adicionar 90 nós, pois ainda serão somente 10 tarefas sendo executadas paralelamente. De forma análoga, se diminuir a quantidade de nós, ocorrerá a serialização de algumas tarefas, penalizando a execução, e, em alguns casos, causando a ociosidade de unidades de processamento.

Para permitir a elasticidade de recursos em um ambiente de nuvem de HPC, devem ser definidos quais os mecanismos de comunicação são compatíveis com a elasticidade. Por exemplo, o uso do padrão de computação paralela *Bag of Tasks*, que se adapta bem à elasticidade, necessita somente da adequação da granulosidade das tarefas.

A definição de nuvem computacional, vista na Seção 2.2, quando vista sob a ótica de usuários de HPC necessita ser adaptada, pois esses usuários tem requisitos ligeiramente diferentes dos demais. Partindo-se da definição anterior de nuvem computacional, e das generalizações identificadas como necessárias para satisfazer os requisitos dos usuários de HPC, a definimos no contexto desta teste, como sendo: um conjunto de recursos computacionais de *software* e *hardware*, disponível sob diferentes camadas de abstração aos usuários, permitindo a ideia de um sistema paralelo, de fácil utilização, possivelmente transparente, e escalável dinamicamente, visando a utilização ótima dos recursos físicos e com garantia de desempenho através de acordos de nível de serviço.

Das características descritas nessa definição, a transparência de *hardware* trata da visão do usuário, a qual deve ser totalmente transparente em ambientes comerciais. No entanto, para usuários de HPC, o conhecimento das especifidades das plataformas de execução é importante, pois as implementações são otimizadas para serem executadas em computadores com características específicas, de modo a obter o máximo desempenho do computador (AHUJA; MANI, 2012).

Já as camadas de abstração referem-se ao nível de visibilidade dos usuários, que deve variar conforme o nível de conhecimento e o interesse de utilização dos usuários. A escalabilidade dinâmica refere-se ao *framework* fazer as escolhas dos recursos automaticamente, de modo a alocar os recursos mais otimizados para as respectivas plataformas, e assim obter o

máximo desempenho do *hardware*. Por fim, a garantia de qualidade deve ser respeitada, pois os usuários de HPC tem interesse na resposta rápida de sua aplicação, e contratar um serviço que não respeite os contratos pode levar a grandes prejuízos.

Em um estudo chamado UberCloud Gentzsch e Yenier (2013), Wolfgang Gentzsch e Burak Yenier lançaram um desafio para um grupo de empresas e usuários individuais, buscando identificar os maiores desafios encontrados na adoção de ambientes de nuvens computacionais para aplicações científicas e de engenharia com requisitos de HPC. Foram formados 80 times, cada um composto por um usuário final da indústria, um provedor de *software*, e um especialista em HPC.

No relatório UberCloud, foram apresentados os 25 times melhores posicionados, onde de modo geral, foram identificadas as seguintes deficiências nos ambientes de nuvens utilizados pelas equipes:

- **Predição de Custos:** Uma das dificuldades encontradas por parte dos usuários foi a falta da predição de custo da execução das aplicações, pois o custo de um recurso por uma hora não é muito baixo, principalmente quando se fala em vários recursos paralelos, ou seja, alocados e usados simultaneamente.
- **Localização da Computação:** Um problema recorrente a muitos usuários de HPC é a transparência de localização dos recursos da nuvem, onde os usuários se responsabilizam legalmente pela computação e poderiam estar infringindo leis locais dos países onde o centro de dados está instalado. Além desse aspecto legal, a distância influencia na capacidade de comunicação, como latência e taxa de transferência de arquivos.
- **Desempenho de Envio das Entradas e *Download* dos Resultados:** Muitos times reportaram a dificuldade no envio dos dados de entrada e *download* dos resultados, devido à baixa qualidade das conexões de internet, muitas vezes agravado pela distância entre o usuário e os recursos da nuvem.
- **Segurança:** No quesito segurança, os usuários buscaram o uso de *Virtual Private Network* (VPN) para adicionar criptografia no envio e recebimento de dados, mas muitos usuários ainda são resistentes à utilização das nuvens por conta da segurança.
- **Desempenho:** O desempenho não foi o foco da primeira etapa do projeto UberCloud. No entanto, os usuários que se dispuseram a avaliar o desempenho nos ambientes de nuvens utilizados identificaram que o uso de escalabilidade horizontal, através de nodos distribuídos, apresenta uma sobrecarga adicional causada pela virtualização da rede. Devido a isso,

grande parte dos times adotaram a escalabilidade vertical, adicionando recursos presentes em uma mesma máquina física, evitando assim o uso da rede de interconexão.

- **Gerenciamento de Licenças:** Para os *softwares* proprietários, um problema adicional foi identificado, o gerenciamento de licenças. Esses *softwares* apresentam licenças para uma quantidade pré determinada de nodos, o que restringe a escalabilidade da aplicação.
- **Resultados das Computações:** Um dos pontos já apresentados foi a dificuldade de *download* dos resultados das aplicações, os quais podem passar por um pré ou pós tratamento para facilitar o *download* dos mesmos. Uma alternativa utilizada por muitos times foi a visualização remota, fazendo inclusive o uso de nodos equipados com GPUs.

Já no relatório UberCloud Gentzsch e Yenier (2018), os problemas com o uso dos ambientes de nuvens computacionais reduziram, principalmente devido à introdução no ano de 2015 do uso de *containers* do Docker pré-configurados para HPCGentzsch e Yenier (2015). Com o uso de *containers*, pôde ser observado um aumento na quantidade de execuções em *bare metal*, aproveitando ao máximo o desempenho dos computadores hospedeiros. Algumas equipes ainda relataram algumas dificuldades pontuais, como é o caso de gerenciamento de licença, interface não amigável a usuários finais, ou ainda na configuração da granularidade das tarefas para prever o tempo de execução.

A seguir, são apresentados alguns dos principais trabalhos e ambientes comerciais de computação em nuvens com foco em alto desempenho.

A Amazon foi uma das empresas pioneiras no serviço de nuvens computacionais, através do serviço denominado EC2 (*Elastic Compute Cluster*), e, no que se refere à proposta de nuvens para HPC, também inovou ao disponibilizar instâncias de maquinas virtuais específicas para HPC. As instâncias que fazem parte dos recursos de HPC apresentam características distintas das demais instâncias. Possuem melhor rede de interconexão, com tecnologia 10 *Gigabit Ethernet*, além da presença de GPU's NVIDIA de última geração, semelhantes aquelas presentes em grande parte dos supercomputadores que apresentem aceleração computacional no ranque Top500. Essas instâncias ainda permitem o agrupamento chamado de grupo de colocação, onde são formados *clusters* de computadores com otimizações na rede de interconexão (TOP500.ORG, 2019; SERVICES, 2019a). Na Seção 5.2, uma descrição detalhada das instâncias de máquinas virtuais EC2 é apresentada, para fins de um dos estudos de caso reportados neste trabalho.

Em Zaspel e Griebel (2011), é apresentada uma avaliação das instâncias de HPC do Amazon, onde foi identificado que, mesmo objetivando aplicações de alto desempenho, a rede

de interconexão ainda é um gargalo, se comparada com o desempenho nativo de *clusters* locais.

Masud (2011) apresenta um *framework* chamado *Penguin on Demand - POD*, com características diferenciadas das demais propostas, baseada em execução não-virtualizada e na utilização de um *cluster* de computadores com rede de interconexão Infiniband. Foi feita uma comparação com o Amazon EC2, utilizando uma aplicação biomédica de uma empresa que trabalha em cooperação com os autores. O desempenho observado foi muito superior que no EC2. No entanto, os autores não apresentam maiores explicações do ambiente disponibilizado e da aplicação, que é identificada como intensiva em comunicação e processamento. A abordagem assemelha-se a tarifação de uso de grades computacionais, onde evidentemente há o desempenho melhor que o de uma nuvem como a Amazon para as instâncias comuns. Talvez por isso a diferença de desempenho tenha sido tão significativa para a referida aplicação utilizada. O POD executou a tarefa em $31,2$ minutos, contra $18,5$ horas no EC2 convencional de maior capacidade.

O SARA [20] é um centro de computação voltado para a pesquisa, o qual oferece atualmente um ambiente de IaaS voltado para aplicações de HPC, possibilitando que os usuários criem *clusters* de máquinas virtuais. Uma funcionalidade interessante que disponibilizam consiste em permitir a utilização de computadores que a organização já possua, desde que seja possível executar uma imagem de disco compatível com o SARA (JHA *et al.*, 2011).

O *CloudBroker Platform*[21], em conjunto com o projeto SCI-BUS (*SCIentific gateway Based User Support*), proporciona um ambiente de nuvem de *software* como serviço, o qual possui diversas aplicações científicas disponibilizadas em uma loja. Esse serviço possui a particularidade de tornar possível que a infraestrutura de execução da aplicação seja fornecida pelo usuário. Assim, o usuário pode optar por usar a sua infraestrutura, sua conta de um serviço de IaaS ou o ambiente do próprio *CloudBroker Platform*. Possui compatibilidade com vários ambientes de IaaS, tais como o Amazon EC2, o IBM SmartCloud e adaptadores para utilização com o Eucalyptus e OpenStack.

Além dos ambientes de nuvens públicas, uma série de trabalhos acadêmicos e iniciativas da indústria buscam o desenvolvimento de tecnologia de infraestrutura de nuvem para HPC, transformando *clusters* de computadores em provedores de nuvens, como é o caso dos trabalhos que seguem.

O Eucalyptus é um *framework* para nuvens computacionais comumente utilizado para execução de aplicações de HPC. Ele implementa o modelo de infraestrutura como serviço

[20] https://www.sara.nl/services/cloud-computing
[21] http://www.cloudbroker.com/

(IaaS), e tem por objetivo ser portável, modular e de fácil utilização (WOLSKI, 2012).

O projeto OpenStack é um ambiente de nuvem de IaaS desenvolvido inicialmente pela *Rackspace Cloud*, uma grande empresa norte americana de hospedagem, e a Agência Espacial Norte Americana (NASA), juntando o Nebula da NASA com a plataforma de nuvem de arquivos da Rackspace. O projeto OpenStack busca criar uma plataforma de computação de código aberto para nuvens públicas e privadas, possuindo escalabilidade sem complexidade adicional (PEPPLE, 2011).

O Aneka é um *framework* que provê o desenvolvimento com um rico conjunto de APIs para explorar de maneira transparente os recursos heterogêneos de *hardware* expressando a lógica da aplicação com uma variedade de abstrações de programas (BUYYA *et al.*, 2011a). Usando Aneka, Vecchiola *et al.* (2009), efetuaram dois testes com aplicações de biologia molecular. O primeiro consiste na classificação da medição de atividade de genes, criando uma imagem da função celular. Já a segunda é um fluxo funcional de imagem de ressonância magnética. Essas imagens, uma vez processadas, visam determinar que parte do cérebro reage em resposta a um estímulo aplicado. Comparando-se o desempenho do *framework* Aneka executando na nuvem do Amazon EC2 com o GRID'5000, um ambiente de grade computacional consolidado, foi identificada uma pequena diferença quando executando com 10 nodos. Aumentando-se para 20 nodos, a diferença aumentou significantemente, fato que os autores atribuem à rede de interconexão utilizada nos testes.

O IBM Platform HPC[22] é uma proposta de solução comercial para transformar *clusters* de computadores em nuvens de PaaS e SaaS, de modo a facilitar a vida dos pesquisadores, de modo que eles não necessitem ser especialistas em computação. Um diferencial apresentado no cenário de HPC em nuvens computacionais é a não obrigação do uso de virtualização, diminuindo assim as desvantagens das sobrecargas adicionadas pela virtualização.

O *Bright Cloud Bursting*[23] possibilita a criação de *clusters* virtuais no Amazon EC2 com poucos cliques de mouse, não necessitando que o usuário tenha conhecimento em Linux. O ambiente virtual criado já possui algumas das ferramentas usuais de um *cluster*. Exemplos dessas ferramentas são analisadores de carga de trabalho, compiladores, depuradores, biblioteca MPI, bibliotecas matemáticas, dentre outras. Possibilita dois modos de utilização: o primeiro é chamado de *cluster* sob demanda, onde toda infraestrutura de cluster é criada no EC2, enquanto o segundo diz respeito a uma extensão de um *cluster* local que o usuário possua acesso.

[22] http://www-03.ibm.com/systems/technicalcomputing/platformcomputing/products/hpc/
[23] http://www.brightcomputing.com/Linux-Cluster-Cloud-Bursting.php

O *Algorithmia*[24] é um serviço de nuvem de algoritmo como serviço (AaaS), modelo semelhante ao SaaS porém com menor granularidade de *software* e que provê um API simples multi-linguagem que permite que os desenvolvedores tenham acesso a um largo conjunto de mais de 3500 algoritmos legados. Esse ambiente de nuvem apresenta um modelo de tarifação padronizado, utilizando o formato de tarifação de 10 créditos de *royalty* e mais um crédito por segundo de execução. Para a escolha das plataformas de execução, utilizam um modelo definido em duas etapas: treinamento e predição. A fase de treinamento consiste em executar aplicações que demoram algumas horas para executar. Com base nos dados do treinamento, é feito a predição de tempo e custo. Para gerenciar as plataformas, utilizam a virtualização com *containers* e, assim como a comunicação entre processos no mesmo nó físico, buscam manter os algoritmos em execução próximos aos dados já utilizados da aplicação, a fim de reduzir o custo de comunicação entre nós físicos.

Por fim, ainda existem trabalhos que implementam a tecnologia de nuvem para HPC para ser utilizada em ambientes de nuvens pré-existentes, transformando nuvens de infraestrutura em nuvens de PaaS ou SaaS. Neste segmento de tecnologias, destacam-se as iniciativas que seguem.

O Nimbis[25] oferece um serviço de intermediação entre provedores de recursos de alto desempenho para Análise Digital de Dados, e usuários finais. Com essa mediação, empresas que não possuem equipamentos de alto desempenho podem contratar os serviços de processamento, no estilo de nuvens computacionais, através de uma interface padronizada para utilização e provimento dos recursos computacionais. A tarifação é feita através de pré-negociação entre as partes, seguindo o modelo *Pay-as-you-go*.

Malawski *et al.* (2011) apresenta um *framework* orientado a componentes que propõe a utilização de nuvens computacionais como ambiente de execução, baseado nos modelos de componentes CCA, CCM e GCM. Utilizam como base os ambientes de infraestrutura o Eucalyptus e o Amazon EC2. Na estratégia utilizada pelos autores, cada componente é encapsulado sob a forma de uma máquina virtual, de modo que ao requisitar a execução de uma aplicação de **n** componentes, são instanciadas **n** máquinas virtuais. Essa abordagem facilita a escalabilidade do *framework*, mas aumenta o custo de inicialização.

Esses ambientes de nuvem voltados para aplicações de HPC apresentam diversas abstrações para facilitar o trabalho dos usuários. Alguns oferecem transparência total das decisões

[24] https://algorithmia.com
[25] https://www.nimbisservices.com/

de execução, enquanto outros oferecem transparência apenas parcial. Finalmente, há aqueles que permitem o controle total da execução. Um problema recorrente a todos é a definição de quais recursos computacionais da nuvem irão executar as aplicações, bem como a quantidade de recursos, o que impacta diretamente nos custos da execução. Quanto maior for o nível de abstração dessas decisões, maior será a necessidade de mecanismos de escolha de recursos.

O problema da escolha de recursos de computação que irão executar determinada aplicação paralela não surgiu com as nuvens. Esse problema vem desde o início do uso de *clusters*, pois, nesses ambientes de execução, é usual o compartilhamento dos recursos entre diversos usuários que executam tarefas simultâneas.

2.3.1 Agendamento de Recursos em Nuvens HPCaaS

Em *clusters* de computadores, um problema recorrente é a escolha dos recursos que irão executar a cada momento. Esse problema se agrava pelo fato de geralmente haver o compartilhamento de um mesmo nó de processamento, por várias aplicações. A quantidade de aplicações que podem estar executando simultaneamente vai depender principalmente do tamanho das instâncias e da quantidade total de recursos. Outros fatores que influenciam nessa quantidade são a presença ou não de preempção, o uso de prioridades, e a estratégia do algoritmo de escolha de recursos, chamado *scheduler* (FEITELSON *et al.*, 1997). Caso a demanda por recursos seja maior que a quantidade total de recursos disponíveis, as tarefas são enfileiradas, aguardando pela liberação de recursos.

Em *clusters*, existe uma maior propensão a falta de recursos e enfileiramento de tarefas. Por conta de seu tamanho e por estar restrito a uma organização, o papel do algoritmo de agendamento de tarefas é muito importante, a fim de maximizar a utilização do *cluster*. Já em nuvens computacionais, como possivelmente compreende uma escala maior que a de um *cluster* local [26], o foco está na escolha de recursos baseada em critérios que vão além da otimização de compartilhamento, embora isso ainda seja desejável. Nesses casos, os custos, a localização geográfica, e eficiência energética, dentre outros, passam a ter relevância no processo de escolha, tornando essa tarefa mais árdua.

A fim de classificar as características utilizadas em *clusters* e nuvens HPCaaS, esta seção irá abordar os principais critérios considerados pelos algoritmos de escolha de tarefas conhecidos.

[26] Na língua inglesa, utiliza-se o termo *on-premise cluster* para *clusters* de gerenciamento local.

Huang *et al.* (2013) apresenta uma estratégia de identificação de quantidade de recursos computacionais para um ambiente de HPCaaS homogêneo, ou seja, onde todos os nós são iguais pela visão dos usuários. Nesse trabalho, ele classifica os agendadores de recursos em quatro categorias, que vem da combinação de duas características. A primeira diz respeito ao momento em que é feito o agendamento dos recursos, podendo ocorrer no momento da submissão, ou no momento do agendamento. Já a segunda diz respeito ao conhecimento prévio do tempo de execução que o algoritmo demandará.

Huang *et al.* (2013), além de agrupar os agendadores quanto à essa classificação, restringe o seu escopo aos algoritmos chamados de moldáveis, pois permite flexibilidade na escolha da quantidade de processadores que serão utilizados no momento da execução. A classificação de moldável vem do trabalho de Feitelson *et al.* (1997), que apresenta uma classificação dos agendadores de tarefas baseado em cinco critérios. Devido à relevância dessa classificação, para posicionar o objetivo atual do sistema de contratos contextuais, bem como apresentar as perspectivas futuras de implementação da HPC Shelf, essas características serão apresentadas em detalhes.

Os cinco critérios utilizados nessa classificação levam em consideração, respectivamente: a quantidade de processadores que será utilizada pela tarefa, a flexibilidade da tarefa, a presença de preempção e em que nível, o conhecimento da carga da tarefa, e a alocação de memória. Seguem elencadas as descrições e variações de cada critério:

- O primeiro critério especifica a quantidade de processadores que será usada.
  - Fixo: Quantidade fixa de processadores, não podendo ser alterada;
  - Variável: Quantidade variável de processadores, definido em tempo de submissão;
  - Adaptativo: Quantidade de processadores determinada durante a inicialização, com base na carga do sistema e a requisição do usuário;
  - Dinâmico: Permite alterar a quantidade de processadores durante a execução.

  Como a quantidade de recursos em um ambiente de nuvem multi organizacional é elevada e compartilhada, bem como potencialmente heterogêneos, o uso de algoritmos adaptativos de maneira indiscriminada é indesejado, pois o objetivo é cumprir os contratos, e não obter o máximo o desempenho. Para suportar os algoritmos dinâmicos, torna-se necessário a criação de mecanismos de sincronização e persistência do estado de um componente, permitindo assim uma propriedade importante dos ambientes de nuvens, a elasticidade. Embora seja algo desejável em ambientes de nuvens computacionais, a exploração dessa

classe de algoritmos ficará indicada para ser tratada em trabalhos futuros.

- O segundo critério diz respeito a flexibilidade do *job*.
  - rígidos: A decisão da quantidade de processadores é fixa, definida externamente ao agendador de recursos.
  - moldáveis: A quantidade de processadores associada ao *job* é determinada pelo agendador de recursos com base em uma série de restrições, tais como, a carga do sistema.
  - evolutivos: Quando a tarefa ocorre, por exemplo, em fases, cada uma com padrões de paralelismos diferentes. Nesse caso, quantidade de processadores pode aumentar ou diminuir com base em requisições do próprio *job*, desde que o algoritmo tenha sido implementado com tal conhecimento.
  - maleáveis: A quantidade de processadores pode variar durante a execução naturalmente, desde que o padrão de paralelismo suporte tal facilidade. Um exemplo de algoritmo maleável é o que utiliza o padrão de paralelismo *bag of tasks*, que dependendo da quantidade de processadores, faz a alocação da tarefa paralela para novos processadores, no caso de aumento, e divide igualitariamente entre os processadores restantes, no caso de redução.

  Essas características estão ligadas ao ambiente de execução, logo serão tratadas em trabalhos de outros pesquisadores do grupo de pesquisa ao qual o autor faz parte.
- O terceiro critério está relacionado a presença ou não de preempção, onde é definido se uma tarefa pode ser suspensa ou até mesmo migrada de um nó para outro. Por fugir ao escopo desta tese, não será explorados em detalhes. Todas tarefas aqui apresentadas serão não preemptivas, ou seja, uma vez que começam a executar, executam até o final. No futuro a preempção pode ser adicionada à nuvem HPC Shelf para otimizar o compartilhamento de recursos.
- Já o quarto critério relaciona-se ao nível de conhecimento do agendador de recursos. O agendador pode não ter conhecimento algum, pode conhecer a carga de trabalho, pode conhecer a classe a que o *job* pertence, e com base nessas informações tomar decisões, ou ainda ter conhecimento de tempo de execução do *job* em específico.
- O quinto e último critério referencia o tipo de hierarquia de memória da plataforma de execução, ou seja, se é de memória compartilhada ou distribuída.

A HPC Shelf é um ambiente de nuvem em que as aplicações são modulares, ou seja,

uma aplicação é composta de componentes, e cada componente é executado em uma plataforma paralela, tornando o processo de escolha mais complexo, pois para cada componente, deve-se encontrar o melhor *hardware*, levando em consideração diversos critérios.

A seção que segue apresenta o paradigma de programação chamado de orientação a componentes, onde explora-se um programa como uma série de módulos, que podem ser manipulados e executados de modo distinto, sendo o paradigma escolhido para ser utilizado na HPC Shelf, desde a sua concepção.

2.4 CBSE (Component-Based Software Engineering)

Nos dias atuais, os requisitos de velocidade e custo no processo de desenvolvimento de produtos, bem como de qualidade do produto final gerado, norteiam os interesses dos setores industriais, pois somente dessa maneira conseguem se manter competitivos no mercado. No desenvolvimento de produtos de *software*, para alcançar esses requisitos, os paradigmas e técnicas de programação geralmente utilizados não contemplam todas as necessidades dos usuários, tornando-se frequentemente necessário o uso conjunto de diversos paradigmas e técnicas para se obter o resultado desejado pelo usuário na aplicação (WANG; QIAN, 2005; CRNKOVIC *et al.*, 2011).

No processo de desenvolvimento de um sistema, existem duas visões diferentes, a visão do usuário e a visão do desenvolvedor. Embora distintas, as visões estão interligadas, de modo que a harmonia entre elas permite que uma aplicação seja desenvolvida de forma vantajosa para ambas as perspectivas.

Na visão do usuário, espera-se que um sistema satisfaça as suas necessidades, tenha um baixo custo de desenvolvimento, seja entregue no prazo, e apresente um pequeno número de defeitos. Essas necessidades são bem básicas. Porém, novas necessidades dos clientes surgem após primeira distribuição, o que motiva o surgimento da maioria dos problemas.

Já na visão do desenvolvedor, espera-se uma maior flexibilidade, definida pela facilidade para adicionar novas funcionalidades, adaptabilidade do sistema, capacidade de customização para vários usuários, boa manutenibilidade, possibilidade de reusar o código para montar aplicações rapidamente, possibilidade de atualizações com menor impacto possível e o aproveitamento de código legado. O que os desenvolvedores buscam, na realidade, é satisfazer as necessidades dos clientes da forma mais econômica possível e, com tempo de entrega reduzido, dedicando mais tempo para outras aplicações. O reaproveitamento de código permite economia

de tempo na entrega de novos produtos, com um código já validado.

Juntando-se as visões de usuários e desenvolvedores, percebe-se uma interseção de características onde o uso de reuso de código, desejado pelos desenvolvedores, possibilita a redução dos custos de desenvolvimento, prazos de entrega e quantidade de erros. Assim, esses aspectos motivam a técnica de programação em que o desenvolvimento é baseado na composição de componentes.

Wang e Qian (2005) definem a CBSE como uma tecnologia de desenvolvimento que permite que programas sejam construídos através de componentes de *software* pré-fabricados, blocos de código de computador reusáveis e auto-contidos, que devem seguir certos padrões definidos, tais como: interfaces bem estabelecidas, modelo de conexão, mecanismo de versionamento e modelo de implantação. Este paradigma objetiva o desenvolvimento baseado na separação entre as interfaces e implementação, sob a forma de componentes. Cada componente apresenta sua interface, que define seus requisitos. O grande objetivo dessa separação está no reuso dos mesmos, o principal objetivo desse paradigma (CRNKOVIC *et al.*, 2011).

Os exemplos mais largamente usados para demonstrar os benefícios do CBSE são oriundos da indústria do *hardware*, uma vez que os computadores são construídos através da montagem integrada de vários componentes eletrônicos distintos e pré-fabricados. Assim como na indústria de *hardware*, a ideia de desenvolvimento de *software* baseado em componentes possibilita a delegação de tarefas de desenvolvimento nas mãos de terceiros, permitindo foco no trabalho da montagem do *software* (WANG; QIAN, 2005).

Os componentes de *software* devem ser desenvolvidos por programadores próprios ou por empresas terceirizadas especializadas no(s) interesse(s) a serem tratados pelo componente. Espera-se que o tempo para a montagem de componentes seja menor que o tempo para desenvolvê-los de maneira própria. Além disso, como os testes de funcionamento são feitos independentemente da aplicação, os componentes não são acoplados à mesma, ou seja, podem ser compartilhados entre aplicações distintas. Os padrões são definidos através de interfaces, assim permitindo a compatibilidade de um componente com outro componente. Apesar do potencial de reuso de componentes entre aplicações, a prática demonstra a real necessidade de técnicas para tratar potenciais variações desses entre aplicações, as quais tem sido estudadas com forte ênfase em pesquisas sobre Linhas de Produto de Software (LPS), tanto no meio acadêmico quanto industrial (CLEMENTS; NORTHROP, 2001).

Enfim, os objetivos mais gerais da programação orientada a componentes são o

tratamento da complexidade, o gerenciamento das mudanças e o reuso de *software* (WANG; QIAN, 2005). A conquista de complexidade é necessária devido ao crescimento contínuo da complexidade do *software* para atender as demandas da sociedade e das economias modernas. Especialmente nos dias atuais, as aplicações devem se mostrar compatíveis com a heterogeneidade das plataformas de computação, além de lidar com a já conhecida inclusão de novas funcionalidades propiciadas pela evolução do *hardware* e sua maior interação com o ambiente. Por sua vez, o gerenciamento de mudanças é necessário devido às mudanças contínuas de requisitos da aplicação, o que é natural na engenharia de *software*, onde mudam-se frequentemente os requisitos dos usuários, o orçamento, as tecnologias com as quais o *software* deve se fazer compatível, etc. Por fim, o reuso é importante para que uma vez implementada uma funcionalidade para satisfazer as necessidades de uma aplicação, ou família de aplicações, tal funcionalidade possa ser reutilizada por outras aplicações com as quais a aplicação original possua interseção de requisitos. Além disso, favorece a otimização das aplicações, uma vez que a otimização na implementação de um componente favorece, teoricamente, as aplicações que já o utilizam. Associado às técnicas de tratamento da variabilidade de *software*, o reuso evita o desenvolvimento de uma funcionalidade várias vezes, economizando-se tempo e, por consequência, orçamento.

Lowy (2005) apresenta os seguintes objetivos do paradigma de programação orientada a componentes:

- **Separação entre interface e implementação:** A interface provê a definição de um serviço abstrato entre cliente e provedor, constituindo um agrupamento lógico de definições de métodos que atuam como contrato entre cliente e provedor de serviço, ficando cada provedor de serviço livre para prover sua própria implementação da interface.
- **Compatibilidade binária:** Um programa orientado a componentes não necessita ser recompilado em sua integralidade a cada mudança, pois não é monolítico. Uma vez que cada componente representa um bloco de construção binário, ele pode ser alterado e compilado separadamente do resto da aplicação. A capacidade de substituir e conectar uma nova versão binária do servidor de serviço implica na compatibilidade com a interface e o ambiente de execução.
- **Independência de linguagem:** Na orientação a componentes, o provedor de serviços é implementado de maneira independente em relação ao cliente, possivelmente em uma linguagem diferente, desde que seja respeitada a interface do provedor para o cliente.

- **Transparência de localização:** A transparência de localização é crucial para a programação orientada a componentes, tornando irrelevante o local onde o componente está sendo executado. Com isso, permite o desenvolvimento local do componente, de forma que o seu funcionamento seja testado de maneira similar ao modo que funcionará em um cenário distribuído.
- **Gerenciamento de concorrência:** Os componentes devem prever sua execução concorrente, evitando *deadlocks* e a interferência entre processos e/ou *threads*. O *framework* de execução de componentes pode prover um serviço de gerenciamento de concorrência. Porém, os componentes devem poder promover a sua própria solução de sincronização, permitindo assim que o componente otimize a granulosidade de tarefas para implementar o interesse a ele associado.
- **Controle de versões:** Em CBSE deve-se permitir que aplicações e componentes evoluam independentemente. É permitido que sejam distribuídas novas versões de um componente sem finalizar a aplicação, desde que a nova versão respeite a interface anterior. A tecnologia de componentes deve detectar, caso exista, a incompatibilidade entre componentes e, assim que possível, alertar a aplicação cliente ou ao próprio usuário do sistema.
- **Segurança baseada em componentes:** Um componente não tem como saber se vai ser usado como código malicioso para corromper dados ou efetuar transferências de dados sem a correta autorização ou autenticação. Além disso, um componente cliente não é capaz de saber se está se comunicando com um componente malicioso. A tecnologia de componentes deve permitir que somente componentes com permissão possam acessar suas interfaces de comunicação.

A CBSE embora tenha surgido de uma demanda de aplicações comerciais, com o passar do tempo vem sendo aplicada também na área de HPC, trazendo seus benefícios às aplicações que demandam de alto desempenho.

2.5 Computação de Alto Desempenho Baseada em Componentes (CBHPC)

O emprego de técnicas de desenvolvimento de *software* baseado em componentes (WANG; QIAN, 2005) dentro do contexto de aplicações com requisitos de HPC [27] é de especial interesse para os objetivos deste trabalho de pesquisa.

A partir do final dos anos 1990, a tecnologia de componentes passou a despertar o

[27] *C*omponent-Based High Performance Computing (CBHPC)

Tabela 4 – Plataforma de Componentes para Computação de Alto Desempenho

Nome	Modelo	Referência e/ou URL
CCA	CCAffeine	(ALLAN *et al.*, 2002)
	XCAT	(KRISHNAN; GANNON, 2004), <http://www.extreme.indiana.edu/xcat>
	DCA	(BERTRAND; BRAMLEY, 2004)
	SciRun2	(ZHANG *et al.*, 2004)
	SciJump	(PARASHAR *et al.*, 2009)
	MOCCA	(MALAWSKI *et al.*, 2005)
Fractal	Julia	<http://fractal.ow2.org/julia>
	Cecilia	<http://fractal.ow2.org/cecilia-site/current>
	Think	<http://think.ow2.org>
	AOKell	<http://fractal.ow2.org/aokell/>
GCM	ProActive	(AMEDRO *et al.*, 2010), <http://proactive.activeeon.com>
Hash	HPE	<http://hash-programming-environment.googlecode.com>

Fonte: Próprio autor.

interesse de pesquisadores de HPC, incluindo cientistas computacionais que viam nessa tecnologia uma possível resposta ao aumento da complexidade das aplicações de seu interesse (POST; VOTTA, 2005). Tal fato já despertava a preocupação da comunidade, notadamente a parcela dessa comunidade que já vinha fazendo uso mais significativo de tecnologias sofisticadas de engenharia de *software* em suas aplicações, tais como *frameworks* computacionais (STEEN, 2006), especialmente nos laboratórios nacionais do Departamento de Energia (DoE) do governo dos EUA. Nesse contexto, surgiram plataformas de componentes voltadas aos requisitos de aplicações de HPC, cujos principais representantes são o CCA[28] (*Common Component Architecture*) (ARMSTRONG *et al.*, 1999; ARMSTRONG *et al.*, 2006), o Fractal[29] (BRUNETON *et al.*, 2006; BLAIR *et al.*,) e o GCM[30] (*Grid Component Model*)(BAUDE *et al.*, 2009), iniciativas custeadas pelas principais agências de fomento a pesquisa e desenvolvimento tecnológico dos EUA[31] e comunidade europeia, devido a importância estratégica em uma maior produtividade no desenvolvimento de soluções computacionais, de problemas que constituem importantes desafios científicos e tecnológicos de interesse desses países.

O trabalho de Hall *et al.* (2008) identificou um conjunto de desafios para a aplicação do modelo de orientação a componentes em ambientes de execução em sistemas heterogêneos distribuídos, descritos a seguir:

- **Diversidade de Modelos de Programação:** A diversidade dos modelos de programação ocorre devido às especificidades da maioria das unidades de processamento, onde destacam-

[28] <http://www.cca-forum.org>
[29] <http://fractal.ow2.org>
[30] <http://gridcomp.ercim.eu>
[31] <http://www.scidac.gov/March2004/ascr_isic_3.html>

se os aceleradores computacionais, dentre os quais GPUs, FPGAs, MICs e Pezy-SC. Esses aceleradores apresentam linguagens de programação e modelos de gerenciamento de memória específicos, para que componentes compatíveis com tais aceleradores possam ser desenvolvidos e executados.

- **Código Legado e Bibliotecas Fortemente Otimizadas:** Bibliotecas de alto desempenho geralmente apresentam implementações otimizadas e fortemente acopladas ao *hardware*. Consequentemente, pode-se escolher bibliotecas com implementações otimizadas, nas quais diferentes implementações de um mesmo componente possam existir. Nesse caso, a melhor implementação deve ser utilizada, sendo a escolha efetuada automaticamente.
- **Particionamento Através de Unidades Funcionais:** Além das implementações otimizadas para modelos de programação particulares, o impacto da sobrecarga adicionada pelo componente deve ser considerado a fim de justificar a sua utilização. Como exemplo, considere um programa paralelo que realiza muitas trocas de dados entre os processos, onde a latência de comunicação da interconexão entre os nós de processamento da plataforma de computação paralela é crítica para execução eficiente. Nesse caso, o sistema escolheria um componente voltado a um ambiente de memória compartilhada, minimizando o impacto do maior custo de comunicação.
- **Gerenciamento de Movimentação de Dados e Sincronização:** Em sistemas de programação distribuída, os dados intermediários de uma aplicação podem ser passados de um componente para outro em uma orquestração de componentes. O local de armazenamento desses dados deve ser levado em consideração, evitando que ocorram esperas e falhas no acesso aos dados. O custo de transferência também deve ser levado em consideração.

Durante os anos 2000, várias plataformas de componentes, ou *frameworks* computacionais, baseados nos modelos CCA, Fractal e GCM foram construídos e validados com aplicações reais nos domínios das ciências e engenharias, cujos mais importantes estão apresentados na Tabela 4. Nas seções seguintes, esses modelos são discutidos.

2.5.1 CCA (Common Component Architecture)

Na especificação do modelo CCA, tem-se duas unidades fundamentais que constituem uma plataforma de componentes: os componentes e os *frameworks*. Os componentes encapsulam código de implementação de alguma funcionalidade. Constituem-se nas unidades de *software* que podem ser utilizadas para montagem das aplicações. Enquanto isso, *frameworks* provêem o

ambiente de implantação, ligação, interação e computação dos componentes (GOVINDARAJU *et al.*, 2007).

Cada componente CCA possui um conjunto de portas, as quais definem as interfaces que são disponibilizadas (portas do tipo *provides*) e as interfaces que são requeridas (portas do tipo *uses*), utilizadas para ligação de componentes através do *framework*. As interfaces entre essas portas são definidas usando a *Scientific Interface Definition Language* (SIDL), a qual estende a IDL do modelo de componentes de propósito geral CORBA[32] (OMG, 2006) com abstrações típicas de aplicações de computação científica, tais como números complexos e *arrays* multidimensionais. Uma especificação SIDL pode então ser mapeada para uma implementação em uma linguagem específica usando o compilador Babel (WILDE; KOHL, 2007), possibilitando a interoperabilidade entre componentes escritos em diferences linguagens usadas no meio científico.

Ao contrário de outros modelos de componentes desenvolvidos para os requisitos de aplicações comerciais, o CCA orienta que seus *frameworks* devem privilegiar conexões diretas entre componentes que residem em um espaço de endereçamento comum, evitando a sobrecarga de indireções e conversões de dados sempre que seja possível.

Frameworks CCA podem ser desenvolvidos para lidar com diferentes ambientes de computação. Para isso, os *frameworks* podem ser:

- *sequenciais*, onde os componentes compartilham um único espaço de endereçamento;
- *distribuídos*, que permitem os componentes executarem em diferentes espaços de endereçamento localizados em máquinas remotamente localizadas entre si; e
- *paralelos*, nos quais os componentes tiram proveito de plataformas de computação paralela, sejam de memória distribuída ou compartilhada.

2.5.2 Fractal e GCM (Grid Component Model)

Diferentemente do CCA, o modelo Fractal possibilita a composição de componentes do forma hierárquica, com componentes definidos pela composição de outros componentes. Permite também a introspecção de capacidades, para monitorar e controlar a execução. Ainda possibilita a reconfiguração das capacidades para modificar dinamicamente o sistema (BAUDE *et al.*, 2009). O GCM constitui uma extensão do Fractal voltado a plataformas de grades computacionais.

[32] *C*ommon Object Request Broker Architecture

A especificação Fractal suporta vários níveis de conformidade padrões, numerados de 0 a 3, onde a capacidade de introspecção e reconfiguração são crescentes. Suas implementações de referência são Julia, para Java, e Cecilia, para C. Outras implementações são AOKell (Java), Think (C) e ProActive (Java). Este último é o que tem sido usado com mais ênfase no nicho de aplicações de CBHPC.

Segundo Baude *et al.* (2009), o modelo CCA não prevê a alocação das plataformas de execução. No entanto, plataformas Fractal e GCM tem suporte nativo para isso em suas respectivas linguagens de descrição arquitetural (ADLs[33]). No GCM, em particular, existem duas possibilidades para descrição de nós da grade, com o conceito de nós virtuais e sem referências aos nós. Em adição ao arquivo com as definições de nome e endereço dos nós virtuais, existem arquivos que possuem as descrições de implantação, onde estão descritas as características da infraestrutura e a forma de implantação.

Conforme apresenta Malawski *et al.* (2007), é possível promover a interoperabilidade entre os modelos CCA e GCM, dessa forma tornando atrativa a importação de componentes já existentes nestes modelos para ambientes de nuvens computacionais (BAUDE *et al.*, 2009). Entretanto, o uso desses modelos em nuvens computacionais não é algo consolidado na prática, devido às penalidades de desempenho introduzidas pelo ambiente virtualizado e redes de intercomunicação que muitas vezes são virtuais também.

A convergência entre as tecnologias de componentes, nuvens computacionais e computação paralela, com o objetivo de alavancar aplicações de Computação de Alto Desempenho de larga escala, constitui o pano de fundo para esta tese de doutorado.

2.5.3 O Modelo Hash de Componentes

No contexto das pesquisas do grupo ao qual faz parte o autor, o modelo de componentes Hash (CARVALHO-JUNIOR; LINS, 2005; CARVALHO-JUNIOR *et al.*, 2007; CARVALHO-JUNIOR; LINS, 2008) tem sido proposto desde as pesquisas com a linguagem Haskell# (CARVALHO-JUNIOR; LINS, 2003), uma extensão paralela para a linguagem Haskell (THOMPSON, 1996) voltada a plataformas de *cluster computing* (BAKER *et al.*, 1999). O modelo Hash tem como principal característica o suporte inerente a componentes paralelos, voltados a implantação, composição e execução em plataformas de computação paralela de memória distribuída, o que permite a separação de interesses de paralelismo dos interesses de negócio dos componentes

[33] *Architecture Description Languages*.

que formam uma aplicação (CARVALHO-JUNIOR; LINS, 2005).

O modelo Hash foi concebido para o desenvolvimento orientado a componentes de programas paralelos, ou seja, a partir da composição de componentes capazes de encapsular interesses de paralelismo (CARVALHO-JUNIOR; LINS, 2005). Devido ao seu foco em computação paralela, enquadra-se dentre os modelos de componentes voltados a CBHPC, discutidos na Seção 2.5. Entretanto, merece destaque especial pois os componentes da HPC Shelf o tomam por base para representar interesses de paralelismo.

Um sistema de programação que é compatível com o modelo Hash deve implementar os seguintes conceitos:

- unidades;
- composição por sobreposição;
- espécies de componentes.

2.5.3.1 Unidades

Ao contrário de outros modelos de componentes voltados a HPC, os componentes do modelo Hash são inerentemente paralelos, bem como totalmente expressivos para expressar interesses de processamento paralelo, incluindo interesses funcionais, como algoritmos paralelos, e não-funcionais, tais como plataformas de computação paralela, estratégias de paralelismo, topologias de processos, etc. Para isso, um componente Hash é constituído por um regimento de *unidades*, associadas aos nós de processamento de uma plataforma de computação paralela de memória distribuída. Unidades comunicam-se através de interfaces de comunicação e sincronização de memória distribuída, tipicamente através de passagem de mensagens. Dessa forma, um componente do modelo Hash é capaz de encapsular as preocupações inerentes da computação paralela. Essa abordagem torna possível que desenvolvedores de software com requisitos de HPC possam preocupar-se somente com as questões do interesse do componente, abstraindo-se das preocupações de paralelismo, desde que reusem componentes pré-existentes, supostamente capazes de tirar o melhor proveito das características da plataforma de computação paralela alvo para obter melhor desempenho, desenvolvidos por especialistas terceiros.

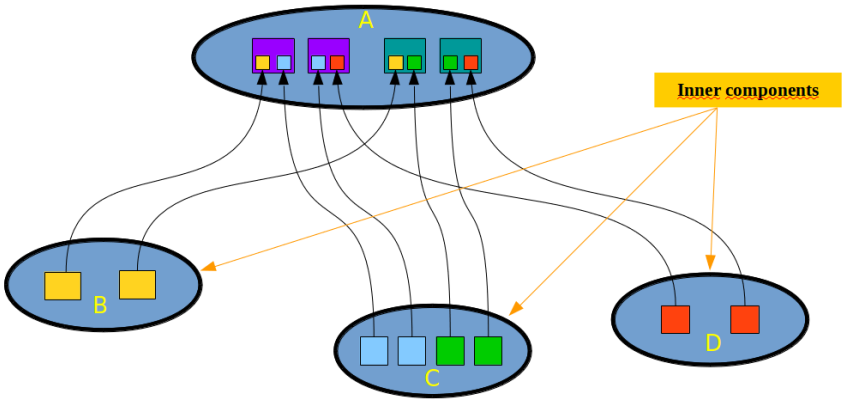
2.5.3.2 Composição por Sobreposição

Um componente do modelo Hash pode ser definido pela orquestração de outros componentes dos quais depende para realizar o seu interesse, de maneira hierárquica e recursiva,

análoga ao modelo Fractal, num processo de composição chamado de *sobreposição* (*overlapping composition*) (CARVALHO-JUNIOR; LINS, 2008). Na composição por sobreposição, as unidades de *componentes aninhados*, dos quais depende o componente em construção para realizar seu interesse, são mapeados às unidades do próprio componente em construção. Formalmente, a sobreposição pode ser definida por uma função injetora (*overlapping function*) que mapeia as unidades dos componentes aninhados às unidades do componente sendo construído. Nesse contexto, dizemos que as unidades dos componentes aninhados (elementos do domínio) mapeados a uma unidade do componente em construção são fatias dessa unidade.

Na Figura 1, é mostrado um processo de composição por sobreposição, onde os componentes são representados por elipses rotuladas por *A*, *B*, *C* e *D*, e as unidades são representadas por retângulos. O componente *A*, nesse exemplo, possui três componentes aninhados *B*, *C* e *D*, dos quais depende para realizar o seu interesse, e quatro unidades que tem como fatias unidades desses componentes aninhados.

Figura 1 – Composição por sobreposição



Fonte: Próprio Autor

2.5.3.3 Espécies de Componentes

O modelo Hash não define a natureza concreta de um componente paralelo, podendo ser visto como um metamodelo comparado a outros modelos de componentes. A definição da natureza concreta de um componente Hash é feita através dos sistemas de programação Hash. Para tanto, devem ser definidas espécies de componentes apropriadas ao ambiente de computação alvo, associadas a requisitos de um certo nicho de aplicação. Uma *espécie* de componentes agrupa componentes Hash que possuem a mesma natureza, tendo em comum apenas o fato de

serem constituídos de unidades distribuídas e compostos entre si por sobreposição. Além disso, componentes Hash de uma mesma espécie possuem o mesmo modelo de implantação em uma arquitetura como também as mesmas restrições de composição com outros componentes Hash. Uma plataforma baseada em componentes Hash pode ainda introduzir novas espécies sem alterar o seu modelo de componentes.

Espécies de componentes podem ser usadas para representar abstrações, do domínio de uma aplicação, que são usadas para especificar problemas e/ou soluções computacionais (para problemas). Nesse sentido, um sistema de programação Hash pode ser visto como uma linguagem de domínio específico (DSL[34]), onde os componentes, de naturezas diferentes distinguidas nas espécies suportadas, constituem as unidades de composição de "programas", os quais representam problemas (visão declarativa) ou soluções computacionais para problemas (visão imperativa).

Em geral, plataformas de componentes de modelos tradicionais estão associadas a uma única espécie de componente e um único tipo de conector entre eles, onde a noção de conector é distinta da noção de componente. Associados ao modelo, define-se um conjunto de abstrações para representar interesses que não podem ser capturados pela noção canônica de componentes. Novas abstrações e conectores podem ser introduzidos para satisfazer requisitos de novas plataformas e ambientes de computação. Exemplos disso são as membranas, controladores e portas coletivas do Fractal, bem como os regimentos de componentes do modelo de programação paralela SCMD[35] de *frameworks* CCA. No caso do Fractal, controladores e alguns tipos de portas coletivas passaram a ser vistos também como componentes no GCM, reforçando a ideia do modelo Hash de suportar várias espécies de componentes que podem evoluir com o tempo, de maneira homogênea.

2.5.4 O HPE (Hash Programming Environment)

A implementação de referência do modelo Hash é o HPE[36] (*Hash Programming Environment*) (JUNIOR *et al.*, 2007; CARVALHO-JUNIOR; REZENDE, 2013), desenvolvido para construção e execução de programas paralelos construídos pela composição de componentes voltados a plataformas de *cluster computing*. A fim de permitir uma prototipagem mais rápida de sua implementação, bem como maior portabilidade, seus componentes são desenvolvidos sobre a plataforma CLI (*Common Language Infrastructure*) (ECMA International, 2006), cujas

[34] *D*omain Specific Language
[35] *S*ingle Component Multiple Data
[36] <http://hash-programming-environment.googlecode.com>

principais implementações são o Mono e o .NET.

O HPE é compatível com o modelo CCA (CARVALHO-JUNIOR; CORREA, 2010). Comparado com outros *frameworks* CCA, o HPE oferece total expressividade para descrever algoritmos de computação paralela e encapsulá-los em componentes, suportando o padrão MCMD (*Multiple Component Multiple Data*). Apresenta pouca sobrecarga inerente de ligação entre os componentes (CARVALHO-JUNIOR; REZENDE, 2013).

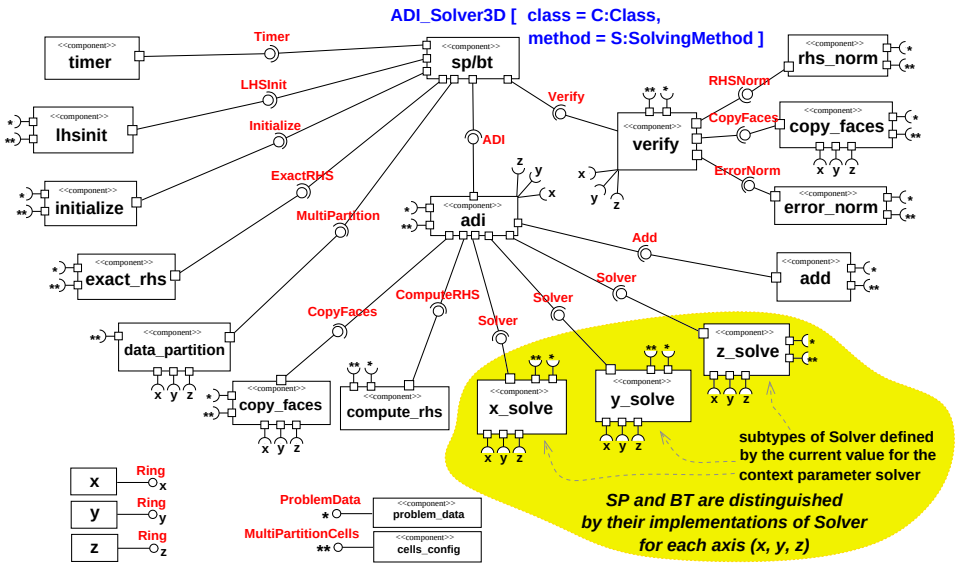
Do ponto de vista do usuário, o HPE é composto por instâncias de três elementos arquiteturais: o *Frontend*, o *Core* e um conjunto de *Backend*'s. O *Frontend* é o ambiente responsável por oferecer uma interface através da qual desenvolvedores possam construir configurações de componentes, representando aplicações finais ou outros componentes, e gerenciar o seu ciclo de vida, registrando componentes em uma biblioteca acessível através do serviço do *Core* e executando aplicações em plataformas de computação paralela (*clusters*) disponíveis, cada qual acessível através de um serviço *Backend*.

As interfaces do *Core* e do *Backend* com o *Frontend* são implementadas usando a tecnologia de serviços *web*[37], permitindo transparência de localização, enquanto o *Frontend* foi implementado como um *plugin* para a plataforma Eclipse, capaz de conectar-se ao serviço *Core* e a algum serviço *Backend* cada vez que deseja executar aplicações. O *Backend* é implementado como uma plataforma de componentes compatível com o padrão CCA sobre a plataforma de execução virtual Mono, do padrão CLI (*Common Language Infrastructure*) (ECMA International, 2006). Embora o Mono suporte várias linguagens de programação, atualmente componentes do HPE devem ser escritos na linguagem C#, a principal dentre as linguagens de plataformas de execução virtual que seguem o padrão CLI, cujos principais representantes são .NET Microsoft (2012) e o próprio Mono.

O HPE suporta as seguintes espécies de componentes: *qualificadores*, *plataformas*, *ambientes*, *computações*, *estrutura de dados*, *sincronizadores*, *topologias* e *aplicações* (CARVALHO-JUNIOR *et al.*, 2007). Elas representam unidades de composição de programas paralelos por passagem de mensagens, ou seja, programas tipicamente construídos usando MPI (DONGARRA *et al.*, 1996), a interface de programação por passagem de mensagens mais difundida e expressiva. Assim, no HPE podem ser construídos os mesmos programas que podem ser construídos em MPI, porém fatorado em componentes, sem perda significativa de desempenho (CARVALHO-JUNIOR; REZENDE, 2013).

[37] *W*eb Services.

Figura 2 – Aplicações Simuladas SP e BT do *NAS Parallel Benchmarks* no HPE



Fonte: (CARVALHO-JUNIOR; REZENDE, 2013)

Uma dos principais aspectos do projeto da plataforma HPE é a sua *estratégia de resolução de componentes*, baseada no sistema de tipos *Hash Type System* (HTS) (CARVALHO-JUNIOR *et al.*, 2016). No HPE, uma aplicação é um componente da espécie *aplicação* que orquestra seus componentes aninhados a fim de implementar o interesse da aplicação. Ao submeter uma aplicação para execução em um *cluster*, através de um serviço *Backend*, os componentes da aplicação são recursivamente resolvidos, implantados e instanciados a partir dos serviços do *Core*. Para cada componente, podem existir uma ou mais implementações. A estratégia de resolução é responsável pela seleção da implementação desenvolvida levando em consideração o maior número de características inerentes à arquitetura do *cluster* alvo, uma vez que tal componente é aquele que, teoricamente, é capaz de melhor explorar o desempenho do computador paralelo.

A Figura 2 ilustra a arquitetura de componentes de uma aplicação real implementada sobre o HPE com o propósito de avaliar o seu desempenho comparado a uma versão monolítica (CARVALHO-JUNIOR; REZENDE, 2013). Trata-se das aplicações simuladas SP e BT do NAS Parallel Benchmarks (NPB), um conjunto de 8 programas, incluindo 3 kernels e 5 aplicações simuladas, desenvolvido pelo Centro de Supercomputação Avançada da NASA, agência especial do EUA, para avaliar o desempenho de plataformas de computação paralela na execução de aplicações de dinâmica dos fluidos, com versões em várias linguagens de programação (C, Fortran, Java, C#) e usando diferentes modelos e interfaces de programação paralela (OpenMP,

HPF, Java Threads, MPI) (Bailey *et al.*, 1991). Os resultados desse trabalho constituem forte evidência de que:

- o HPE não introduz sobrecarga de desempenho inerente comparado à mesma aplicação desenvolvida sem a abstração dos componentes;
- o HPE possui expressividade para representar padrões de computação paralela que possam ser programados com MPI;
- um montador de componentes e aplicações é capaz de abstrair-se dos detalhes de paralelismo, encapsulados nos componentes mais básicos.

Além desses resultados, a abordagem de componentes permitiu identificar os interesses comuns entre as aplicações SP e BT e encapsulá-los em componentes de maneira que possam ser compartilhados entre as duas aplicações, sem redundância. Na versão original monolítica, não baseada em componentes, tais interesses estavam separadamente implementados no código fonte de cada aplicação, escondendo suas similaridades de implementação. O HPE permitiu ainda a separação de interesses não-funcionais, alguns dos quais transversais, tais como estratégias de distribuição de dados, suposições sobre classes de problemas, padrões de comunicação paralela, que não estavam, e não podiam ser, separados em procedimentos na versão original.

2.6 Convergência de Componentes e Nuvens em HPC

Este trabalho partiu da investigação dos ambientes de nuvens computacionais voltado para aplicações comerciais, seguida pela extensão dessa investigação para abranger os ambientes de nuvens computacionais que objetivam aplicações de HPC[38].

Como resultado dessa investigação, surgiram dois pontos chaves a serem considerados como restritivos para os usuários de HPC na adoção de execução em nuvens computacionais: a sobrecarga inserida pela utilização obrigatória de tecnologias de virtualização de infraestrutura e a transparência dos recursos.

Foram identificadas diversas propostas de ambientes de nuvens computacionais para aplicações de HPC. No entanto, trabalhos que avaliam tais propostas relatam que os problemas ainda persistem. Para a sobrecarga inserida pela virtualização, algumas propostas flexibilizam seu ambiente para execução nativa nas plataformas computacionais.

Ao flexibilizar a virtualização, surgem diversos problemas relacionados ao escalonamento e compatibilidade entre aplicação e plataforma. Os problemas de escalonamento

[38] High Performance Computing

referem-se a encontrar um conjunto de nós disponível e com as capacidades necessárias para a aplicação. Já a compatibilidade entre aplicação e plataforma é causada principalmente pela incompatibilidade entre aplicações e diferentes versões de sistemas operacionais e bibliotecas utilizadas pelas aplicações. Agravando o problema da compatibilidade, está a dificuldade de instalar uma nova biblioteca em uma plataforma que está em produção. Uma aplicação pode utilizar diversas bibliotecas durante a sua execução, e dificilmente uma única plataforma de execução terá todas instaladas.

Para solucionar o problema do escalonamento, uma alternativa é a virtualização a nível de sistema operacional, como é o caso do OpenVZ[39] e o Docker[40]. Nesse tipo de virtualização, existe uma série de restrições, como, por exemplo, a obrigatoriedade de utilização do mesmo *kernel*. Essa estratégia auxilia, em parte, o problema da compatibilidade. Porém, devido a restrições do sistema operacional, ainda é limitada.

Outra possibilidade que vem sendo explorada é a disponibilização de um conjunto fixo de aplicações, onde o usuário somente as utiliza, sem ter conhecimento da sua plataforma de execução. Como esse conjunto de aplicações é fixo, as plataformas de execução já foram manualmente configuradas para suportá-las. Nesse caso, torna o serviço disponibilizado restrito, não disponibilizando um ambiente genérico de nuvem computacional para HPC.

Além das questões provenientes do uso de nuvens computacionais para aplicações de HPC, a computação heterogênea, com o emprego conjunto de arquiteturas paralelas de memória distribuída e compartilhada, bem como aceleradores computacionais, introduz um grau muito maior de complexidade no processo de desenvolvimento de *software* voltado às aplicações de HPC. Como já discutido, aceleradores computacionais geralmente apresentam linguagens próprias de desenvolvimento, dentre outras peculiaridades, como uma série de restrições (HALL *et al.*, 2008), além de terem seu modelo de programação influenciado pela necessidade de sincronização eficiente com os núcleos (memória compartilhada) e nós (memória distribuída) de processamento paralelo onde estão acoplados.

Além dos desafios encontrados nesse cenário, com o estudo sobre aplicações de HPC, nuvens computacionais e o desenvolvimento de aplicações através do paradigma de orientação à componentes, foi identificado um conjunto de propriedades que um ambiente de nuvem que concilie essas três áreas deve respeitar o conjunto de propriedades identificados a seguir:

- **Interoperabilidade com Diferentes Aceleradores Computacionais:** A interoperabili-

[39] http://openvz.org/
[40] http://www.docker.com/

dade representa a propriedade de suportar diversas arquiteturas de processamento em uma mesma plataforma de computação. Os aceleradores computacionais geralmente apresentam linguagens e técnicas de programação próprias, as quais devem ter a capacidade trabalhar em conjunto a fim de tirar o melhor proveito da capacidade de processamento de uma plataforma (BAXTER *et al.*, 2007; WEBER *et al.*, 2011). Os componentes devem ser capazes de definir suposições sobre as plataformas para permitir o casamento entre *software* e o *hardware*. Tais plataformas devem ter seus recursos definidos explicitamente.

- **Segurança:** O ambiente de execução de componentes deve apresentar mecanismos de recuperação de falhas e evitar a existência de componentes maliciosos. Dessa forma, os canais de comunicação devem ser explícitos e decididos pelos usuários que compõem os componentes, ao invés dos desenvolvedores de cada componente básico.
- **Desempenho:** Refere-se às garantias de satisfação de cronogramas de execução de componentes, motivadas pelos interesses dos seus usuários, cuja responsabilidade envolve tanto os componentes de *software* quanto as plataformas de computação paralela, uma vez que o principal requisito dos usuários de HPC é o desempenho. Deve prover mecanismos de monitoramento da execução e mecanismos de manutenção do desempenho ofertado pelos provedores dos diversos tipos de componentes.
- **Comunicação entre Componentes:** A propriedade de comunicação é inerente de computações distribuídas, nas quais incluem-se as aplicações de HPC, onde deve haver a comunicação entre os nós paralelos de processamento, e essa comunicação deve ser confiável e segura. Deve ser permitido dois padrões de comunicação possíveis: a comunicação intra-plataforma e a extra-plataforma. Na comunicação intra-plataforma, os nós estão em uma mesma rede. Para ser compatível com o desempenho desejado, evita-se a adição de sobrecarga na rede de interconexão, pois o principal indicador de desempenho nesse caso é a latência da rede. Já na comunicação extra-plataforma, a comunicação ocorre entre plataformas de computação paralela distintas, engajadas em uma computação de larga escala, podendo estarem distribuídas geograficamente. Nesse caso, é admissível adicionar sobrecarga na comunicação em benefício de segurança, uma vez que o indicador de desempenho crítico passa a ser a taxa de transferência.
- **Elasticidade de Componentes Paralelos:** A elasticidade é a propriedade que diz respeito a alteração na quantidade de recursos computacionais alocada para uma aplicação específica e pode ocorrer de forma vertical ou horizontal. Os componentes devem ser de-

senvolvidos para que trabalhem de forma consciente do recurso de elasticidade e permitam explicitamente sua alteração em tempo de execução.

- **Compatibilidade com Componentes Legados:** A compatibilidade com código legado visa a incorporação de códigos escritos nos últimos 50 anos, possivelmente em linguagens como Fortran, os quais tem sido validados com o uso repetido pela comunidade. Por exemplo, esse é um requisito especial do projeto CCA, nos anos 2000, visando atender necessidades de cientistas computacionais (STEEN, 2006).
- **Tarifação de Componentes:** A propriedade de tarifação refere-se à cobrança de valores monetários ou créditos de acesso, pelo uso dos recursos de uma nuvem, abstraídos sob a figura de componentes. É uma funcionalidade necessária para possibilitar a utilização desses recursos de maneira equilibrada, garantindo a disponibilidade à vários usuários.

A fim de lidar com o maior grau de complexidade da programação paralela advinda da computação heterogênea, esta tese investigou aspectos sobre o emprego da abstração de componentes sobre plataformas de nuvens computacionais. Porém, conforme sugere Malawski *et al.* (2011), sua utilização ainda não é algo consolidado, e ainda é preservada a sobrecarga da virtualização e a transparência de localização, principais críticas por parte dos usuários de HPC. Como proposta para a solução desses problemas, foi concebida a HPC Shelf, uma plataforma para construção de aplicações de HPC baseada em nuvem computacional que visa disponibilizar um ambiente de execução capaz de satisfazer os requisitos dos usuários de HPC, bem como respeitar as propriedades identificadas nessa seção. Nessa plataforma, a escolha dos componentes é feita automaticamente através do sistema de contratos contextuais, chamado Alite, principal contribuição desta Tese. Embora tenha sido concebido no contexto da HPC Shelf, os conceitos introduzidos pelo Alite podem ser adaptados para outros arcabouços CBHPC baseados em nuvens.

3 A PLATAFORMA HPC SHELF

Como discutido na Seção 2.2, nos últimos anos, o modelo de oferta de recursos de computação sob demanda como serviços, chamado *nuvem computacional*, tem se consolidado, tendo sido empregado com sucesso em um grande número de aplicações, dentre as quais aquelas que apresentam requisitos de HPC.

A HPC Shelf é uma plataforma para provimento de serviços de HPC baseado em tecnologia e abstrações de nuvens computacionais para o desenvolvimento, implantação e execução de aplicações com requisitos severos de recursos computacionais, notadamente demandando o emprego de múltiplas plataformas de computação paralela para a solução de grandes problemas. Os serviços HPC Shelf tem a orientação a componentes paralelos do modelo Hash como principal premissa de projeto.

A HPC Shelf atende diversos perfis de usuários além de usuários finais de aplicações, ditos *usuários especialistas*. Dentre eles, incluem-se os *montadores/provedores de aplicações*, *desenvolvedores de componentes* e *mantenedores de infraestrutura*, interessados em ofertar recursos a outros usuários interessados através dos serviços da plataforma. Da união de seus esforços e equilíbrio entre seus interesses, conjectura-se que a HPC Shelf constitui em um substrato fértil para o desenvolvimento de soluções para grandes problemas em aplicações de ciências computacionais e engenharia através do compartilhamento de recursos de forma eficiente.

A orientação a componentes possibilita que uma mesma aplicação na HPC Shelf possa ser executada empregando um conjunto de plataformas de computação paralela distintas, utilizando componentes oferecidos por desenvolvedores de componentes capazes de explorar as características peculiares da arquitetura de cada plataforma, de forma transparente ao desenvolvedor de aplicações. Assim, por exemplo, componentes podem ser escritos em linguagens de programação diferentes, usando bibliotecas distintas de suporte ao paralelismo ou de subrotinas científicas, bem como diferentes tipos de aceleradores computacionais presentes na plataforma.

A principal contribuição do projeto da HPC Shelf é a aproximação dos usuários de HPC ao uso de ambientes de nuvens computacionais, tornando seu desenvolvimento relevante para o cenário atual da computação.

Para oferecer contribuições nesse contexto, torna-se necessário o tratamento dos principais problemas identificados pelos usuários de HPC. Ekanayake e Fox (2010) apresentam dois fatores que, segundo eles, dificultam que usuários de HPC migrem dos *clusters* privados

para nuvens computacionais, que são a transparência de localização geográfica dos computadores e a sobrecarga na execução adicionada pela virtualização. Esse último é um recurso tecnológico identificado como tecnologia base das nuvens computacionais, mas ambos são propriedades do conceito de computação em nuvem utilizado sob a ótica de aplicações comerciais. A identificação desses dois problemas foi reforçada por uma análise sobre o projeto UberCloud, que buscou disseminar o uso de nuvens computacionais para HPC, como já discutido na Seção 2.3.

Na HPC Shelf, busca-se principalmente eliminar ou minimizar a níveis aceitáveis o problema da sobrecarga gerada pelas tecnologias de nuvens computacionais. Para tratar essa sobrecarga, busca-se a escolha da plataforma de execução mais adequada, dentro de um conjunto heterogêneo de plataformas de computação paralela disponíveis, ditas *plataformas virtuais*. O processo de escolha considera informações sobre a plataforma de execução e os componentes, e ainda permite a flexibilização do uso da tecnologia de virtualização. Para alcançar isso, foram criadas abstrações que restringem a escolha dos recursos, dessa forma permitindo que a melhor escolha possível seja feita, inclusive que possa ser forçada a escolha uma plataforma nativa, também chamada de "*bare metal*", sem virtualização. Essa flexibilização, por permitir que a tecnologia de virtualização de *hardware* seja desativada, evita a sobrecarga de processamento inerente de uma plataforma virtualizada, e assim reduzindo a perda de desempenho dos programas. Nos casos em que não possam ser executadas de forma nativa, a virtualização é utilizada como recurso de compatibilidade entre os componentes e plataformas, e, nesses casos, sendo penalizada com a sobrecarga inerente da virtualização.

A expressão "melhor escolha possível" não está relacionada somente ao critério de desempenho. Devem também ser considerados outros critérios no processo de escolha, os quais extrapolam o critério do desempenho em termos de redução do tempo de execução e aceleração/eficiência, tais como o custo financeiro de aquisição de recursos e a eficiência energética, dentre outros, cada vez mais presentes no processo de decisão das empresas e centros de supercomputação. Assim, a HPC Shelf considera três grupos de critérios para guiar a seleção dos componentes de sistemas de computação paralela gerados pelas aplicações: os de *aplicação*, os de *plataforma*, os de *qualidade* e os de *custo*. Os critérios de aplicação e plataforma, em conjunto, permitem restringir os componentes por compatibilidade entre a implementação, a especificação do usuário e as plataformas disponíveis, visando critérios de desempenho. Por sua vez, os critérios de qualidade tem o papel de filtrar os componentes e plataformas de modo que atinjam limiares definidos pelo usuário para os critérios determinados. Assim como os critérios

de qualidade, os critérios de custo fazem a filtragem dos componentes que satisfaçam restrições de custo, seja esse financeiro ou de créditos por uso em plataformas de computação paralela de uso compartilhado.

A transparência de localização, assim como a virtualização, também pode ser flexibilizada. Caso o usuário deseje definir restrições sobre a localização, ele poderá fazê-la adicionando uma região que esteja de acordo com sua necessidade, definindo assim uma limitação geográfica para o processo de escolha de componentes.

Diferentemente de um ambiente de nuvem computacional, onde há o conceito de loja virtual onde são escolhidas as opções de execução, configurações da aplicações e infraestruturas de execução, a HPC Shelf não possui um ambiente único de interação. Cada perfil de usuário tem contato com uma interface de interação distinta e construída visando a disponibilizar os recursos necessários ao respectivo perfil de usuário. Essas interfaces de interação permitem que os usuários se comuniquem com os serviços disponibilizados pela HPC Shelf divididos em três elementos arquiteturais: o *Frontend*, o *Core*, e o *Backend*, responsáveis pela interação com os usuários intervenientes anteriormente mencionados, bem como o controle do ciclo de vida dos componentes, desde sua e seleção até sua implantação, instanciação e execução.

Na Seção 2.5.3, foram apresentados os fundamentos do modelo de componentes da HPC Shelf, o modelo de componentes paralelos Hash. Nas seções que seguem, os outros principais conceitos e abstrações que definem a plataforma HPC Shelf serão detalhados. Inicialmente, a Seção 3.1 introduz as suas espécies de componentes. Por sua vez, a Seção 3.2 apresenta as características e interesses das suas classes de usuários intervenientes. A Seção 3.3 descreve seus elementos arquiteturais: *Frontend*, *Core* e *Backend*. Finalmente, a Seção 3.4 apresenta um exemplo de arcabouço de componentes desenvolvido para a HPC Shelf, a qual permite a construção de sistemas de computação *MapReduce*(REZENDE; Carvalho Junior, 2018).

Por ser contribuição principal e específica deste trabalho, merecendo maiores detalhes, o seu sistema de contratos contextuais para seleção de componentes (Alite) será assunto do Capítulo 4.

3.1 Espécies de Componentes da HPC Shelf

Uma vez que a HPC Shelf utiliza o modelo de componentes Hash, faz-se necessário definir o conjunto de espécies de componentes usados na composição de sistemas de computação paralela. São eles:

- **Computações** representam a implementação de algoritmos de computação paralela, exportando ações computacionais através de *portas de ações*.
- **Plataformas Virtuais** representam plataformas de computação paralela de memória distribuída sobre as quais componentes de computação são implantados, instanciados e executados.
- **Fonte de Dados** representam repositórios de grandes massas de dados de interesse de computações, acessíveis por meio de *portas de ambiente*.
- **Ligações de Ambiente** representam a conexão entre uma porta de ambiente usuária de um componente com outra porta de ambiente provedora compatível de um outro componente. Uma porta provedora oferece um serviço a uma porta usuária consumidora. Esse serviço é definido por um conjunto de operações oferecidas na porta usuária e atendida através do conjunto de operações, possivelmente distintas, implementadas pela porta provedora. É responsabilidade da ligação de ambiente tratar da adaptação entre a interfaces das portas usuária e provedoras ligadas.
- **Ligações de Ações** representam a conexão entre uma ou mais portas de ações. Cada porta de ações exporta um conjunto de *nomes de ações*, as quais devem ser as mesmas entre as portas ligadas. Em uma ligação de ações, um ação se completa quando é ativada em todas as portas de ações envolvidas na ligação (*rendezvous multilateral*), quando os componentes envolvidos podem sincronizar operações. Por exemplo, realizando uma ação computacional ou sinalizando alguma condição.
- **Conectores** representam acopladores entre fontes de dados e computações. Para isso, subdividem-se em facetas, distribuídas entre as plataformas virtuais aonde os componentes acoplados encontram-se instanciados. Podem fazer o papel de orquestração, sincronizando a ativação de ações computacionais de componentes de computação, através de *ligações de ações*, de "palco" para coreografia entre componentes de computação e fontes de dados, através de *ligações de ambiente*, ou uma combinação de ambos. Uma vez que as facetas de conectores encontram-se instaladas na plataforma virtual onde os componentes acoplados estão instanciados, tais ligações são diretas, dentro do mesmo espaço de endereçamento, como preconizado pelo modelo CCA como requisito para sistemas de HPC baseados em componentes.
- **Qualificadores** são usados em valorações de parâmetros de contexto em contratos contextuais. De fato, no protótipo de referência da HPC Shelf, implementado como uma extensão

ao HPE, representam contratos contextuais de componentes das espécies do HPE (Seção 2.5.4), acrescidos de qualificadores para representar tipos numéricos necessários ao Alite.

- **Workflow** representa o componente de um sistema de computação paralela responsável pela orquestração de seus componentes de computação e conectores, chamados de *componentes de solução* junto com fontes de dados, através de ligações de ações. O código de orquestração é escrito através do subconjunto de orquestração da linguagem SAFeSWL (*SAFe Scientific Workflow Language*), a linguagen de descrição de workflows científicos suportado pelo framework de aplicações da HPC Shelf, chamado SAFe, discutido na próxima seção. De fato, SAFeSWL é a linguagem usada para descrição de sistemas de computação paralela, possuindo, além de um subconjunto de orquestração, um subconjunto arquitetural, através do qual são descritos os seus componentes e sua composição.
- **Aplicação** representa o componente de um sistema de computação paralela através do qual a aplicação comunica-se com os componentes do sistemas de computação paralela que instancia, através exclusivamente de portas de ambiente. Através dessas portas, nos papéis de usuária ou provedora, uma aplicação é capaz de capturar eventos emitidos por componentes de solução durante a execução, assim como enviar dados de entrada, coletar resultados e monitorá-los.

Por fim, é importante definir a noção de *componente de sistema*, constituído por um par formado por um componente de computação e um componente plataforma virtual sobre o qual o componente de computação encontra-se instanciado.

3.1.1 Porta de Controle do Ciclo de Vida

Todos os componentes da HPC Shelf possuem uma porta de ações destinada ao controle explícito do seu ciclo de vida durante a execução do workflow de um sistema de computação paralela. Os nomes dessas ações, bem como o efeito de sua ativação em um componente de computação ou faceta de conector, são apresentados a seguir:

- `resolve`, que causa a execução do algoritmo de resolução de contratos do Alite sobre o contrato que representa o componente, resultando na seleção de um componente concreto que atende ao contrato, com possível auxílio da aplicação;
- `deploy`, que causa o envio do código de um componente concreto para compilação e instalação no ambiente da plataforma virtual alvo;
- `instantiate`, que causa a criação de uma instância do componente, tornando-o apto a

realização de ações computacionais e de comunicação de interesse da aplicação;

- `run`, que causa o disparo da execução do procedimento principal da instância do componente, iniciando sua "lógica" de orquestração, i.e. a sincronização de ações com os componentes a ele ligados por ligações de ações, especialmente o componente workflow;
- `release`, que causa a liberação dos recursos alocados pelo componente na sua instanciação.

Para os casos de plataformas virtuais e de fontes de dados, as ações `resolve` e `release` possuem significado semelhante. No entanto, a ação `deploy` não tem qualquer efeito, embora, no caso de plataformas, dependendo da implementação do serviço *Backend*, possa resultar na execução de operações preliminares para preparar a infraestrutura para instanciação da plataforma virtual posteriormente. Por outro lado, a ação `instantiate` de uma plataforma virtual instancia a plataforma virtual. Por exemplo, pode resultar na criação das máquinas virtuais utilizadas pela plataforma. Ao final dessa operação, a plataforma virtual torna-se preparada para receber a implantação de componentes de computação e conectores. Finalmente, a instanciação de uma fonte de dados pode resultar na execução de operações para controle de acesso para a aplicação.

É importante notar que existe um protocolo para ativação das ações do ciclo de vida de componentes. Por exemplo, um componente de computação ou conector só pode ser implantado após a instanciação da(s) plataforma(s) virtual(is) sobre a qual(is) deverá ser instanciado. Caso contrário, a operação é bloqueada até que essa condição seja satisfeita. Além disso, para um mesmo componente, as operações `resolve`, `deploy`, `instantiate`, `run` e `release` só podem ser completadas nessa ordem, por razões óbvias. Entretanto, uma vez que seja liberado após a instanciação anterior, após executar zero ou mais execuções, um componente pode ser instanciado diversas vezes em um mesmo *workflow*, assim como pode ter seu contrato resolvido e implantado repetidas vezes, com a possível escolha de diferentes implementações a cada ativação de `resolve`. A expressão regular a seguir expressa o protocolo de ativação das ações do ciclo de vida de um componente: $\left(\texttt{resolve}\cdot\texttt{deploy}\cdot\left(\texttt{instantiate}\cdot\texttt{run}^{\star}\cdot\texttt{release}\right)^{\star}\right)^{\star}$

3.2 Intervenientes da HPC Shelf

Os *atores* da nuvem HPC Shelf, também chamados de intervenientes, representam as partes interessadas, no consumo e provimento de seus serviços. Para isso, são agrupados em classes que caracterizam interesses comuns bem como responsabilidades. As classes de atores

são: os *especialistas*, os *provedores de aplicações*, os *desenvolvedores de componentes*, e os *mantenedores de plataformas*.

Atores de diferentes classes possuem diferentes visões dos serviços providos pela HPC Shelf, refletido na necessidade de diferentes interfaces para acesso aos serviços, como será apresentado na Seção 3.3, na definição dos *Frontend's*.

A Tabela 5 mostra a relação entre as classes de intervientes e as abstrações que manipulam, identificando suas ações. Observe que a visão proposta na tabela é simplificada, pois há um conjunto maior de abstrações que estão envolvidas no processo de desenvolvimento e execução de uma aplicação na HPC Shelf, no entanto os conceitos serão introduzidos sob demanda nesta tese, a fim de facilitar sua compreensão.

Tabela 5 – Relação entre os Atores da Nuvem

max width=

	Desenvolvedor de Componentes	**Provedor de Aplicações**	**Mantenedor de Plataformas**	**Especialist De Domíni**
Componentes	Cria	Restringe	-	Usa
Plataformas Virtuais	Restringe	Restringe	Cria	Usa
Workflow	-	Cria	-	Usa

Fonte: Próprio Autor

O usuário especialista, ou especialista, é aquele interessado nas aplicações da nuvem, com o desejo de resolver problemas do seu domínio de interesse. O provedor de aplicações é responsável pela montagem e provimento de aplicações a partir da orquestração de componentes disponibilizados no catálogo de componentes da nuvem. Tais componentes são construídos pelos desenvolvedores de componentes, ou desenvolvedor, de maneira a usar da melhor forma possível, os recursos computacionais de plataformas de computação paralela alvo. Por fim, mantenedores de plataformas, ou simplesmente mantenedores, são responsáveis por gerenciar as plataformas de computação paralela (infraestrutura) da nuvem, através de plataformas virtuais disponibilizadas sob a abstração de componentes. Uma discussão mais minuciosa sobre as características e papéis dos intervenientes da HPC Shelf é apresentada nas próximas seções.

3.2.1 Usuário Especialista

O objetivo principal da nuvem HPC Shelf é a disponibilização de serviços de HPC para usuários finais, os quais podem ser caracterizados como especialistas em um certo domínio específico em ciências computacionais ou engenharias, de onde surgem a maioria das aplicações

com requisitos de HPC. Em geral, usuários especialistas não são especialistas em tecnologias de computação, menos ainda em arquiteturas de plataformas de computação paralela. Por consequência, não desejam, tampouco necessitam, preocupar-se com detalhes da execução para resolver problemas do seu interesse.

As aplicações de usuários especialistas seriam apresentadas sob a forma de interfaces para descrição de problemas de seu interesse e construção automatizada de soluções computacionais baseadas em componentes para sua solução, utilizando abstrações adequadas ao domínio de interesse dos especialista. Portanto, após a descrição de um problema de seu interesse, o especialista pode solicitar à HPC Shelf a construção uma solução computacional para o problema, sob a forma de um sistema de computação paralela. Dependendo da aplicação, o usuário pode então interagir com a execução da solução ou apenas esperar o resultado final. Por exemplo, fazendo o *download* dos resultados brutos, ou, alternativamente, após efetuar algum pós-processamento para sintetização de dados, sendo possível ainda utilizar uma ferramenta de visualização dos resultados da própria aplicação. Os mecanismos de entrada e saída são inteiramente definidos pela aplicação, não havendo restrições próprias da HPC Shelf para sua definição.

Enfim, a visão do especialista na HPC Shelf é de uma nuvem SaaS, abstraindo-se de características de infraestrutura e do *software* (componentes) usado para solução computacional dos problemas de seu interesse. De fato, o modelo de portais *web* dentro do contexto de nuvens SaaS, discutido por Calegari *et al.* (2019), é bastante adequado para caracterizar aplicações da HPC Shelf. Porém, não há imposição para que uma aplicação possa oferecer ao especialista uma visão PaaS, em um nível mais baixo de abstração, permitindo, por exemplo, que o especialista tome decisões no nível de infraestrutura e composição de componentes na configuração de sistemas de computação paralela, até mesmo oferecendo ao especialista a visão total dos componentes.

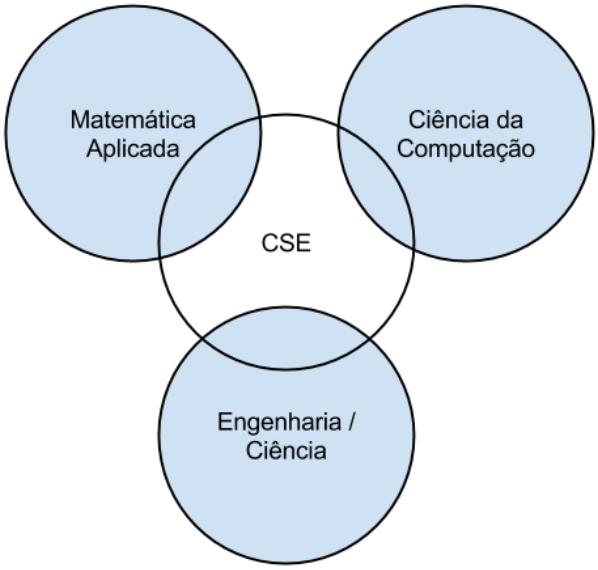
3.2.2 Provedor de Aplicações

O provedor de aplicações é um usuário especializado em métodos computacionais para resolver problemas em um determinado domínio. Ele é capaz de construir uma aplicação através da qual usuários especialistas podem descrever problemas de seu interesse, pois conhecem as abstrações do domínio para descrição de problemas em alto nível de abstração. Para isso, o provedor precisa ter a capacidade de pesquisar os componentes do catálogo de componentes da nuvem, a fim de reusar componentes preexistentes, e também de desenvolvê-los quando

necessário, atuando também, nessas circunstâncias, como um desenvolvedor de componentes.

Este perfil de usuários está associado à área denominada *Computational Science and Engineering* (CSE) [1], a qual representa a interseção entre áreas como a computação, a matemática e as engenharias, conforme é apresentado na Figura 3.

Figura 3 – Interseção de áreas de conhecimento CSE



Fonte: Próprio Autor

Os provedores de aplicação são responsáveis por definir *sistemas de computação paralela*, os quais representam soluções computacionais para os problemas declarados por especialistas. Tais sistemas são descritos por intermédio de uma linguagem de composição de workflows científicos, denominada SAFeSWL, através da qual é possível descrever a orquestração de um conjunto de componentes, bem como sua arquitetura de composição.

Através do Alite, o sistema de contratos contextuais proposto nesta Tese de Doutorado, o provedor poderá restringir os requisitos de cada componente de um sistema de computação paralela, tanto em relação a aspectos funcionais, como não-funcionais, incluindo restrições de desempenho, custo e de qualidade da solução gerada.

É de responsabilidade do provedor, para cada aplicação, a definição de todas as funcionalidades e controles que serão disponibilizados aos especialistas. Porém, caso deseje que o usuário especialista possa interferir, direta ou indiretamente, no processo de seleção de componentes, deve apresentar para tal na interface da aplicação.

[1] http://www.seas.harvard.edu/programs/graduate/computational-science-and-engineering

3.2.3 Desenvolvedor de Componentes

O desenvolvedor de componentes é um especialista em computação paralela, com possível e desejável especialização em métodos computacionais, a fim de conhecer e ser capaz de implementar da melhor maneira possível os algoritmos mais eficientes para problemas de interesse científico e de engenharia sobre arquiteturas de diferentes plataformas de computação paralela.

Portanto, o conhecimento das arquiteturas de plataformas de computação paralela e projeto de algoritmos paralelos é crucial para o desenvolvedor, uma vez que sua função é escrever implementações de componentes paralelos que atendam aos requisitos de contratos, que podem ser escritos tanto por provedores quanto por outros desenvolvedores de componentes.

Dentre os requisitos impostos por tais contratos, destacam-se, as otimizações para arquiteturas de uma determinada classe de plataformas de computação paralela, além de aspectos de qualidade e custo envolvidos na execução. O desenvolvedor deve inserir o máximo de informações possível para que permita à HPC Shelf escolhê-lo quando este seja o mais adequado.

Os componentes desenvolvidos podem ser auto-contidos ou definidos pela composição de outros componentes, que também podem ter sido desenvolvidos por composição, recursivamente. Essa característica é herdada de outros modelos de programação orientada a componentes, como CCA, e por consequência o modelo Hash.

3.2.4 Mantenedor de Plataforma

Os mantenedores de plataformas são responsáveis, em conjunto, por criar e gerenciar o conjunto *plataformas virtuais* da HPC Shelf, sobre infraestruturas de computação paralela que administram. O mantenedor deve conhecer bem as características dessa infraestrutura de modo a possibilitar que a HPC Shelf escolha de maneira otimizada os elementos de *hardware* usados em cada plataforma virtual.

A partir do conjunto de plataformas de computação paralela físicas sob responsabilidade do mantenedor, o mantenedor cadastrará no catálogo de componentes mantido pelos serviços do *Core*, *perfis de plataformas virtuais* para usufruto das aplicações, a partir dos quais plataformas virtuais que satisfazem os requisitos impostos pelo perfil podem ser instanciados sobre a sua infraestrutura. Um perfil de plataforma virtual é, de fato, um sinônimo para um contrato contextual para componentes da espécie plataforma.

A partir de um perfil de plataforma, é possível criar uma plataforma virtual, uma instância de componente da espécie plataforma, reservando os equipamentos e deixando-os prontos para receber o componente que irá executar. Logo, cada plataforma virtual é associado a um mantenedor específico, representando um procedimento de instanciação que atende ao perfil de plataforma que a implementa sobre a infraestrutura do mantenedor. No entanto, vale ressaltar que vários mantenedores podem oferecer um mesmo perfil de plataforma virtual.

Esse conceito de plataforma virtual de computação paralela é semelhante ao conceito de nós virtuais do ADL do GCM, Fractal e ProActive, porém com a abstração de uma máquina paralela e não somente um nó (BAUDE *et al.*, 2009).

3.3 Elementos Arquiteturais da HPC Shelf

A plataforma HPC Shelf é constituída pelos seguintes elementos arquiteturais:

- o *Frontend*, o qual oferece às aplicações os serviços básicos para acesso aos recursos da HPC Shelf, ou seja, a construção de sitemas de computação paralela para solução de problemas descritos pelo especialistas através de sua interface de alto nível;
- o *Core*, responsável pelos serviços de manutenção do *catálogo de componentes* e que governam o seu ciclo de vida desses componentes, dentre os quais o sistema de resolução de contratos contextuais chamado Alite;
- o *Backend*, serviço responsável pelo gerenciamento de plataformas virtuais sobre uma determinada infraestrutura de computação paralela mantida por um certo mantenedor, incluindo a instanciação de máquinas virtuais, bem como o monitoramento, controle e balanceamento da utilização de recursos computacionais.

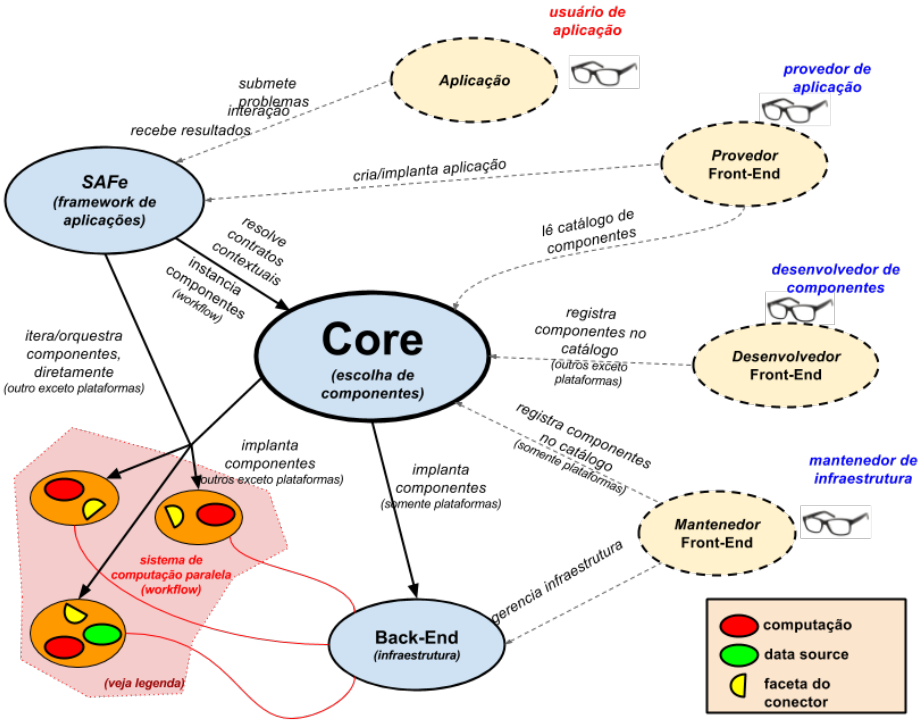
A interação entre tais elementos arquiteturais da nuvem possibilita que as aplicações possam ser criadas e executadas na HPC Shelf, dividindo-se as responsabilidades.

Nas seções que se seguem, os elementos arquiteturais da HPC Shelf são descritos em detalhes, juntamente com a definição das interações definidas através da disponibilização de serviços *web*.

3.3.1 Frontend (SAFe)

O *Frontend* da HPC Shelf é chamado SAFe (*Shelf Application Framework*), um arcabouço para construção de *aplicações* da HPC Shelf, oferecendo acesso aos serviços da

Figura 4 – Interação entre elementos arquiteturais HPC Shelf



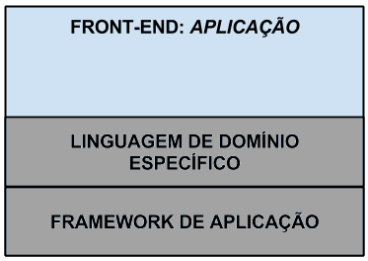
Fonte: Próprio autor

plataforma (de Carvalho Junior *et al.*, 2019).

As aplicações representam a interface dos especialistas com os serviços da HPC Shelf. Especialistas escolhem uma aplicação que desejam utilizar para descrever problemas no seu domínio de interesse, a fim de obter uma solução computacional através dos recursos computacionais oferecidos pela HPC Shelf de forma transparente por intermédio do SAFe.

Uma instância do SAFe comunica-se com sua aplicação hospedeira através da linguagem SAFeSWL (*SAFe Scientific Workflow Language*), projetada para descrição de sistemas de computação paralela baseados em componentes da HPC Shelf. Para isso, SAFeSWL é constituída de dois subconjuntos: arquitetural e de orquestração. Através do primeiro, são descritos os componentes de um sistema de computação paralela, através de contratos contextuais, e suas interligações. Por sua vez, através do último, é descrito um fluxo de execução para as ações computacionais de componentes de computação e conectores, a fim de resolver o problema de interesse do sistema de computação paralela. Portanto, é responsabilidade do SAFe gerenciar as etapas do ciclo de vida de componentes de um sistema de computação paralela desde a resolução de seu contrato contextual até a execução de suas ações computacionais, passando pela sua

Figura 5 – Diagrama da Estrutura de uma Aplicação



Fonte: Próprio Autor

implantação e instanciação sobre a plataforma virtual alvo.

A Figura 5 apresenta um diagrama da estrutura básica de uma aplicação da HPC Shelf, posicionada sobre uma DSL e o *framework* de aplicações (SAFe), vista como o *Frontend* da HPC Shelf. De fato, o especialista enxerga a aplicação através da DSL, que pode ser textual, gráfica ou meramente uma entrada de dados para computações preestabelecidas pela aplicação, acessível por meio de um navegador *web*. Por sua vez, o *framework* de aplicação diz respeito à visão do provedor de aplicações, o qual realiza suas responsabilidades atuando sobre os pontos de extensão oferecidos pelo SAFe. Portanto, o SAFe fornece ao provedor de aplicações os serviços que permitem o acesso transparente aos componentes da nuvem por parte da aplicação.

Até aqui, somente foi explorado o subconjunto arquitetural, que influencia na criação dos contratos contextuais dos componentes, identificando que componente irá se conectar e qual deve ser o tipo do próximo componente para que sejam compatíveis, principalmente no que diz respeito a formatos de entradas e saídas, além dos tipos de dados. Essa compatibilidade, uma vez que os contratos sejam corretamente construídos, é garantida pelo sistema de contratos contextuais, produto desta tese.

No entanto, além de definir os contratos, um sistema de gerenciamento de *workflow* deve garantir o controle dos momentos de inicio e fim da execução de cada componente, devendo controlar inclusive a sincronização entre os componentes que estarão executando, como por exemplo a espera pelo final da execução de um componente para começar. Esses aspectos de controle de execução são definidos no subconjunto de orquestração.

No subconjunto de orquestração são definidos construtores que permitem os controles das ações de execução que podem existir sobre os componentes. São definidas as operações *start*, *wait*, *cancel* e *invoke*, que são aplicadas a contratos contextuais já resolvidos pelo sistema de contratos contextuais. Além disso, existem as ações de resolução, que são definidas pelas operações *resolve*, *deploy*, *instantiate*, *run* e *release*, já discutidas na Seção 3.2. Considere um

sistema de computação paralela que possui um único componente de computação associado a uma plataforma virtual para a execução do mesmo, descritos por contratos contextuais. O primeiro passo de orquestração em seu *workflow* (código de orquestração) é a resolução desses contratos, a partir da ativação das suas respectivas ações *resolve*. A partir da conclusão dessas ações, o componente de computação (concreto) foi escolhido no catálogo de componentes, juntamente com sua plataforma virtual, mas até então a plataforma virtual só existe de forma abstrata. Para que a plataforma virtual seja efetivamente reservada e implantada, é necessária a ativação da sua ação *deploy*. Então, a ação *deploy* do componente de computação já pode ser ativada, causando o envio do código fonte do componente de computação para a plataforma virtual, para que seja compilado e implantado. Uma vez que que o código do componente de computação encontra-se pronto na plataforma virtual, a ação *instantiate* do componente de computação pode ser ativada para que esse seja instanciado, tornando-se pronto para que seja iniciado seu procedimento principal, como efeito da ativação de sua ação *run*. Após a execução final, o componente e a plataforma virtual podem ter seus recursos liberados pela ativação da ação *release*.

Esse exemplo de um único contrato é somente ilustrativo, pois para uma aplicação ter funcionalidade completa ela irá receber dados e deverá gerar saídas. Assim, deverão existir pelo menos mais 2 componentes para permitir a entrada dos dados e para armazenar os resultados.

Além das operações primitivas de execução, existem operações compostas que combinam as operações primitivas com comportamentos pré definidos, podendo ser composta hierarquicamente. As operações compostas são:

- sequenciamento de ações, que especifica um conjunto de operações que devem ser executadas uma após a outra, ou seja, sequencialmente;
- paralelização de ações, que especifica um conjunto de operações que serão executadas de maneira concorrente;
- escolha entre ações, que especifica a escolha entre um conjunto de operações, em função da ativação de um conjunto de ações, de modo que é executado o ramo correspondente uma das ações que se completaram primeiro;
- repetição de ações, que especifica uma execução iterativa de uma operação, sendo a condição de terminação também definida em função da ativação de ações.

3.3.2 Core

As principais responsabilidades do *Core* são:

- A manutenção de um catálogo de componentes, das diferentes espécies suportadas, oferecendo o acesso para aplicações por intermédio de contratos contextuais;
- A resolução de contratos contextuais, através do Alite, para seleção de componentes que atendam a requisitos funcionais e não-funcionais impostos pela aplicação na construção de sistemas de computação paralela, detalhadamente explicado no Capítulo 4 como a principal contribuição desta Tese;
- O gerenciamento do ciclo de vida dos componentes, observando as especificidades de suas espécies, desde sua introdução no catálogo, por parte dos seus intervenientes responsáveis (atualmente, desenvolvedores de componentes e mantenedores de plataformas virtuais), até a sua escolha, por *resolução* de contratos contextuais, *implantação*, *instanciação*, *execução* e *destruição* quando não são mais necessários em um sistema de computação paralela;

Para disponibilizar esses serviços, o *Core* disponibiliza um conjunto de *serviços web* que são acessíveis pelas aplicações, por intermédio do SAFe, e pelos serviços *Backend*:

- o *serviço de manutenção do catálogo de componentes*, oferecido a desenvolvedores de componentes e mantenedores de plataformas, incluindo operações de inclusão de componentes e gerenciamento de suas versões, para fins de atualização;
- o *serviço de resolução de contratos contextuais*, oferecido ao SAFe, a partir do qual a seleção de componentes é realizada a partir de um contrato contextual;
- o *serviços de implantação e instanciação de componentes*, também oferecidos ao SAFe, que atuam de forma diferente conforme a espécie do componente em questão.

Dentro desse contexto, vale a pena lembrar que os serviços de seleção, implantação, instanciação, execução e destruição de componentes são executados dinamicamente dentro do fluxo de execução do *workflow* do sistema de computação paralela, respectivamente pela ativação das ações `resolve`, `deploy`, `instantiate`, `run` e `release` em sua porta de ações de controle do ciclo de vida.

Na instanciação de plataformas virtuais, o *Core* comunica-se com o serviço *Backend* onde a plataforma virtual deverá ser instanciada. Por sua vez, a instanciação de componentes de computação e conectores, bem como sua implantação prévia, é realizada pelo *Core* comunicando-se com o serviço da plataforma virtual hospedeira, já previamente instanciada. Por sua vez, na ativação de ações computacional de componentes de computação e conectores, a comunicação

do SAFe é realizada diretamente com os componentes, já instanciados. Por esse motivo, existe uma ordem específica em que as operações do ciclo de vida dos componentes devem ser ativadas no código SAFeSWL.

3.3.3 Backend

O *Backend* é o elemento arquitetural responsável por realizar o mapeamento de perfis de plataformas virtuais suportados pelo mantenedor para as plataformas de computação paralela reais sob seu domínio, bem como oferecer serviços de monitoramento dessas plataformas ao *Core* e ao SAFe.

Um perfil de plataforma virtual carrega a informação de qual serviço *Backend* (mantenedor de plataforma) o instanciará. De fato, o perfil carrega toda a informação necessária para que o *Backend* seja capaz de alocar recursos do parque computacional do mantenedor para instanciar uma plataforma virtual que atenda aos requisitos impostos pelo perfil. Para isso, o *Backend* pode utilizar informação sobre o estado das plataformas de computação paralela reais do seu parque computacional e compará-las com as exigências de qualidade de serviço de serviço especificadas no perfil.

Ao criar uma plataforma virtual, o *Backend* inicia uma instância de um *Agente de Plataforma* que ficará ativo na plataforma durante todo seu ciclo de vida, respondendo às requisições do *Core*.

Quando uma plataforma virtual não for mais necessária, é responsabilidade do *Backend* decidir se destrói a mesma ou a mantém ativa, a fim de reduzir o tempo de criação de uma nova plataforma caso, em um momento futuro, seja requisitado uma instanciação de um perfil compatível, ou ainda, para otimizar o armazenamento de dados.

Caso não seja possível instanciar o perfil, o *Backend* deve informar ao *Core* que não foi possível a criação da plataforma virtual, gerando uma exceção. A partir desse momento, o *Core* irá se responsabilizar por tomar as providências cabíveis, com auxílio da aplicação, selecionando uma nova plataforma virtual para o perfil ou abortando a execução da aplicação.

Atualmente, foram desenvolvidos as seguintes implementações de serviços *Backend*, que demonstram a expressividade do modelo:

- instanciação de plataformas virtuais em um computador local, sem uso de mecanismos de virtualização, geralmente usado para testes da plataforma, mas que também poderia ser empregado para usar o computador local do usuário como uma infraestrutura de execução

de componentes (computações ou fontes de dados);

- instanciação de plataformas virtuais sobre um cluster com suporte à tecnologia de virtualização OpenStack[2], o qual tem sido empregado para uso do *cluster* do LIA (Laboratório de Pesquisa em Computação), laboratório que serve ao nosso grupo de pesquisa, como infraestrutura de computação;
- instanciação de plataformas virtuais sobre o serviço EC2 da Amazon, como descrito na Seção 5.2;
- instanciação de plataformas virtuais sobreo serviço GCP (*Google Cloud Platform*).

3.3.3.1 Agente de Plataforma

O *Agente de Plataforma* é responsável pela comunicação da plataforma virtual com o *Core* e o SAFe, disponibilizando Serviços *Web* específicos para implantação de componentes na respectiva plataforma, copiando os códigos fonte do *Core* para a plataforma e compilando os mesmos, bem como sua instanciação e ligação com outros componentes. É também responsabilidade do *Agente de Plataforma*, intermediar a comunicação do SAFe com as portas de ambiente e ações dos componentes que estão ligadas ao componente de aplicação e ao componente *workflow* do sistema de computação paralela.

Um *Agente de Plataforma* possui o tempo de vida da plataforma virtual que o hospeda, permitindo o reaproveitamento da plataforma virtual por diferentes aplicações. Atualmente, o *Agente de Plataforma* é uma adaptação do *Backend* HPE.

3.4 Exemplo: Um Arcabouço para Processamento MapReduce

Com o objetivo de oferecer exemplos de sistemas de computação paralela da HPC Shelf, apresentamos um arcabouço de componentes paralelos voltado a construção de sistemas de computação paralela que implementam o paradigma de processamento de larga escala *MapReduce* (SILVA, 2016; REZENDE, 2017; REZENDE; Carvalho Junior, 2018).

3.4.1 O framework MapReduce

Buscando reduzir a complexidade de programar aplicações paralelas de memória compartilhada e de memória distribuída, o Google Inc.[3] desenvolveu um modelo de programação

[2] <https://openstack.org>
[3] <https://www.google.com>

e execução chamado *MapReduce*, derivado das funções *map* e *reduce* presentes em linguagens funcionais, e que se tornou um dos principais modelos de programação para nuvens computacionais. Ele provê facilidade na representação de aplicações compatíveis com o modelo, de modo a explorar paralelismo de maneira implícita, delegando o controle de concorrência, tolerância a falhas e comunicação ao *framework* de execução (GUO *et al.*, 2004; LI *et al.*, 2016; DEAN; GHEMAWAT, 2008).

Uma aplicação *MapReduce* é definida principalmente pelas funções de mapeamento (*Map*) e redução (*Reduce*). Os dados são recebidos por instâncias de *Map* que transformam os dados de entrada em pares intermediários (chave/valor) que serão usados pelas instâncias de *Reduce*, que processam os dados intermediários e geram os dados de saída (GUO *et al.*, 2004). O controle da distribuição dos dados se dá pelo *Google File System* (GFS), um sistema de arquivos responsável pelo gerenciamendo dos pares intermediários. O uso desse modelo teve tamanha importância para o Google que deu início à sua adoção em ambientes de nuvens computacionais e reimplementaram o seu mecanismo de índice invertido, usado no mecanismo de buscas, usando o seu próprio arcabouço MapReduce.

Além das funções *Map* e *Reduce*, alguns refinamentos surgiram para otimizar o modelo para plataformas de execução paralelas ou auxiliar as funções *Map* e *Reduce*, como é o caso da função de particionamento (*partition function*), responsável por particionar os dados intermediários gerados pela etapa de mapeamento. Outro caso é a função de combinação (*combiner function*), responsável por um tratamento prévio dos dados intermediários, possibilitando a redução na quantidade de dados transferidos em plataformas paralelas de memória distribuída. Além desses, outras implementações buscam otimizar a alocação das tarefas, comunicação, controle de múltiplas fases de *MapReduce*, e ainda *MapReduce* iterativo, de modo a tornar o método mais eficiente (LI *et al.*, 2016).

Tanto a fase de mapeamento quanto a de redução podem ser executadas em paralelo, de modo que existam M tarefas de mapeamento e N de redução, onde M e N não são necessariamente a quantidade de nós de processamento dos ambientes de execução.

Alguns exemplos de processamentos comumente realizados com *MapReduce* são: contador de frequência de palavras, *grep* (busca de palavras-chave em arquivos) distribuído, contador de frequência de acessos a um URL, gráfico de WEB-Link reverso, índice invertido, ordenação distribuída, dentre outros. De acordo com Dean e Ghemawat (2010), no ano de 2010, já haviam sido criadas mais de dez mil aplicações distintas usando o *MapReduce*.

Além do *framework MapReduce* desenvolvido pela Google, diversos *frameworks* alternativos foram desenvolvidos, dentre os quais destaca-se o Hadoop, desenvolvido pela Apache Software Foundation[4], atualmente o *framework* de código aberto que domina esse cenário. Li *et al.* (2016) apresentam um *survey* abrangente de *frameworks MapReduce*.

A Figura 6 ilustra a execução de uma computação *MapReduce* típica, apresentada por Dean e Ghemawat (2004), cujos passos são descritos a seguir:

1. Inicialmente, tem-se a criação das diversas instâncias de agentes para executar as funções de mapeamento (*Mapper*) e redução (*Reducer*), bem como uma instância especial chamada mestre (*Master*).
2. O ambiente de execução faz a divisão da entrada de dados em pedaços (*chunks*), armazenado numa estrutura de dados no formato de pares chave/valor (*key/value*) e associado a uma tarefa de mapeamento (*Map Task*).
3. Os processos *worker* da tarefa de mapeamento aplicam a função e geram pares intermediários, que são armazenados localmente.
4. Os processos *worker* da tarefa de redução buscam os pares intermediários através de consulta com o *Master*, ordenam os pares pela chave quando diferentes chaves forem atribuídas a um mesmo *Reducer*, e aplicam a função de redução. Após a redução, os resultados são combinados para gerar os arquivos de saída.

Uma aplicação *MapReduce* pode não gerar uma saída, mas um conjunto de pares intermediários para outro *MapReduce*, o que não interfere no seu funcionamento, dado que a estrutura de dados dos campos dos pares não é rígida, permitindo que os desenvolvedores definam seu formato.

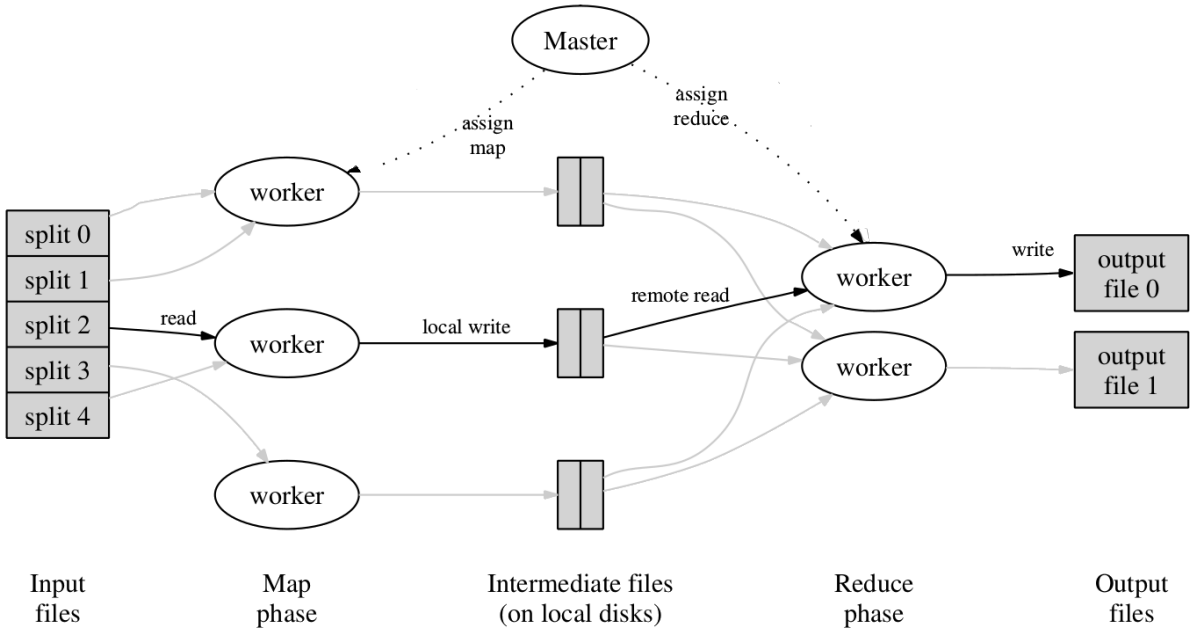
3.4.2 MapReduce na HPC Shelf

Dada sua importância no cenário de computação em nuvens, o *MapReduce* torna-se uma das aplicações para validação da HPC Shelf e por consequência, do Alite. A implementação na HPC Shelf, no entanto, consiste em criar, não só a as funções de mapeamento e redução, como também todo o ambiente de execução, permitindo que diferentes configurações do *framework* de execução sejam definidas.

Um sistema de computação paralela MapReduce é constituído por instâncias dos componentes de computação MAPPER e REDUCER, ligados através de instâncias dos com-

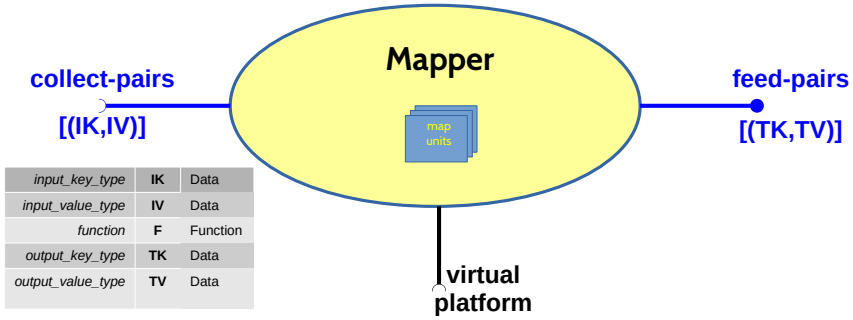
[4] <https://www.apache.org/>

Figura 6 – Visão de Execução



Fonte: (DEAN; GHEMAWAT, 2004).

Figura 7 – Componente MAPPER (Agentes de Mapeamento)



Fonte: Próprio Autor

ponentes conectores SPLITTER e SHUFFLER. Esses quatro componentes são respectivamente ilustrados nas figuras 7, 8, 9 e 10. Nessas ilustrações, deve-se observar que cada componente possui uma porta de entrada chamada **collect_pairs** e uma porta de saída chamada **feed_pairs**. O serviço dessas portas é a transmissão de listas de pares chave/valor (e.g. $[\langle \mathtt{K}, \mathtt{V} \rangle]$) ou pares chave/multi-valor (e.g. $[\langle \mathtt{K}, [\mathtt{V}] \rangle]$), onde $\mathtt{K}$ e $\mathtt{V}$ representam, respectivamente, tipos arbitrários de chaves e valores. Os componentes de computação possuem ainda uma porta de alocação, chamada **virtual platform** nas figuras 7 e 8. Finalmente, os parâmetros de contexto de aplicação são também apresentados nas figuras.

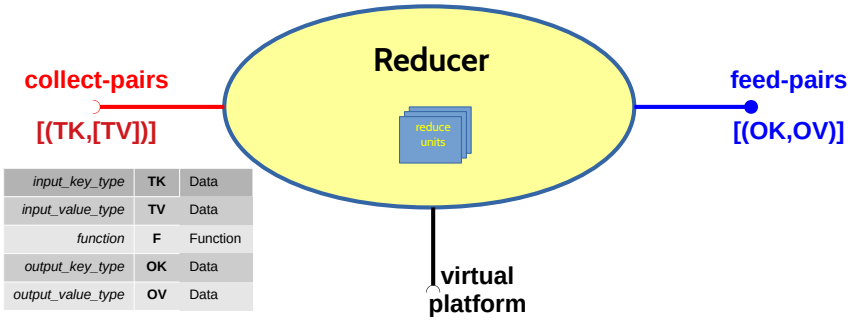
O significado de cada componente é apresentado a seguir:

1. MAPPER: agente de mapeamento, que aplica uma função a cada par chave/valor recebido através da porta **collect_pairs**, retornando, para cada um, uma lista de pares chave/valor através da porta **feed_pairs**.
2. REDUCER: agente de redução, que aplica uma função sobre par chave/multi-valor recebido através da porta **collect_pairs**, emitindo, para cada par, um único par chave/valor através da porta **feed_pairs**.
3. SPLITTER: encaminha pares chave/valor recebidos através suas portas **collect_pairs**, uma para cada faceta coletora, entre suas portas **feed_pairs**, cada uma associada a uma de suas facetas alimentadoras. Portanto, suas facetas coletora e alimentadora são múltiplas, podendo serem usadas para conectar múltiplos agentes de mapeamento e/ou redução, seja para receber saídas desses agentes ou para enviar entradas para eles, como ilustrado na Figura 11.
4. SHUFFLER: Diferencia-se do SPLITTER por agrupar os pares chave/valor recebidos de suas portas **collect_pairs** que possuem uma mesma chave em um único par chave/multi-valor, antes de encaminhá-las através de suas portas **feed_pairs**.

As figuras 7, 8, 9 e 10 omitem as portas de ações oferecidas por esses componentes, utilizadas para orquestração do sistema MapReduce realizada pelo seu componente *workflow*. Antes de apresentá-las, é preciso explicar que os pares chaves/valor são transmitidos entre agentes de computação e conectores em listas de *chunks*, onde um *chunk* corresponde a uma lista de pares chave/valor ou chave/multivalor, apenas com o propósito de controle da granularidade do processamento e redução da sobrecarga de comunicação resultante da transmissão individual de cada par.

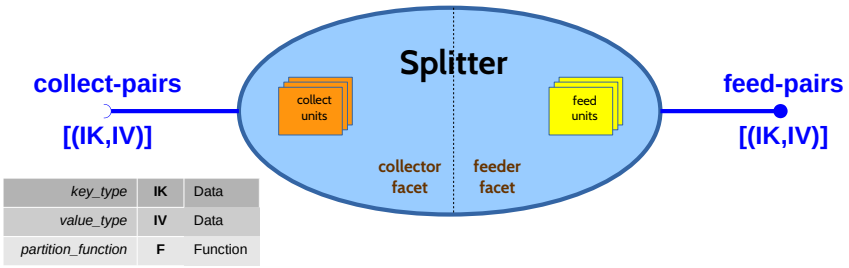
Uma porta de ações chamada **task_collector_read_chunk** encontra-se presente em

Figura 8 – Componente REDUCER (Agentes de Redução)



Fonte: Próprio Autor

Figura 9 – Componente SPLITTER



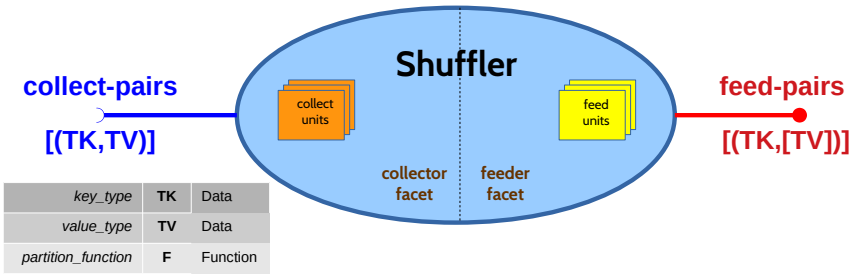
Fonte: Próprio Autor

facetas coletoras de conectores, com uma única ação contendo os seguintes nomes de ações alternativos: `READ_CHUNK`, `FINISH_CHUNK`. A primeira é ativada pelo conector sempre que um *chunk* é lido através da porta **collect_pairs**, enquanto a última é ativada após o último *chunk* ser lido, sinalizando terminação da entrada associada àquela faceta. Para orquestração da leitura de um conjunto de pares, o componente *workflow* pode realizar a ativação alternativa dos dois nomes de ação como controle de uma iteração, onde a iteração é finalizada quando a ativação se completa com um `FINISH_CHUNK`, como no fragmento de código SAFeSWL a seguir:

```
<iterate id_port="task_collector_read_chunk" until="FINISH_CHUNK" loop="READ_CHUNK">
...
</iterate>
```

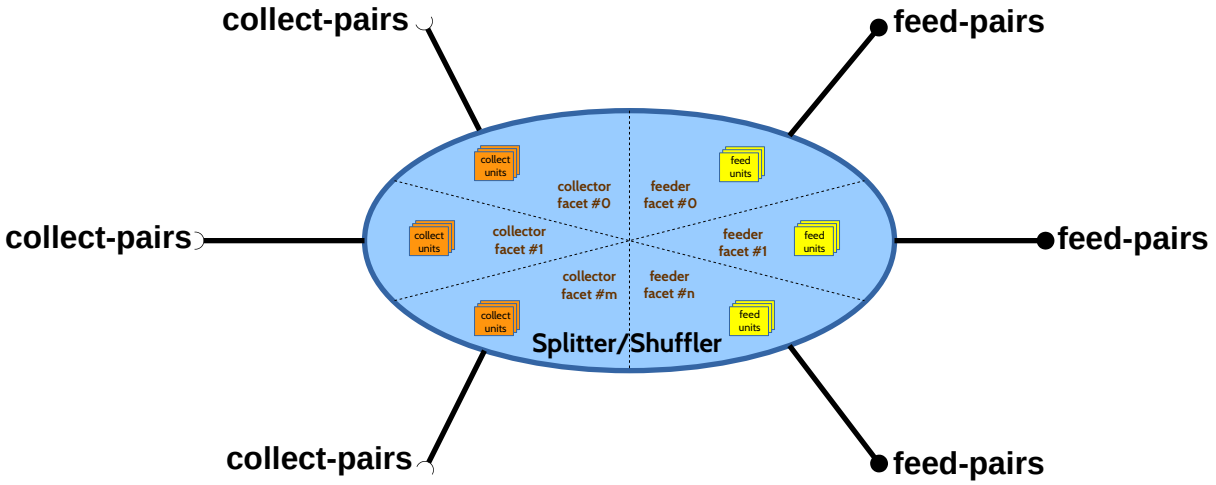
Além disso, uma porta de ações chamada **task_collector_active_status** encontra-se presente em cada faceta coletora, cujas ações são identificadas pelos seguintes nomes: `BEGIN_CHANGE_STATUS`, `ACTIVE`, `INACTIVE`, `END_CHANGE_STATUS`. O objetivo dessa porta é a habilitação ou desabilitação da leitura de pares através da faceta coletora a ela associada. A primeira e a última servem para o componente *workflow* sinalizar que pretende modificar o status da faceta, por razões de segurança de execução, enquanto as duas outras servem para

Figura 10 – Componente SHUFFLER



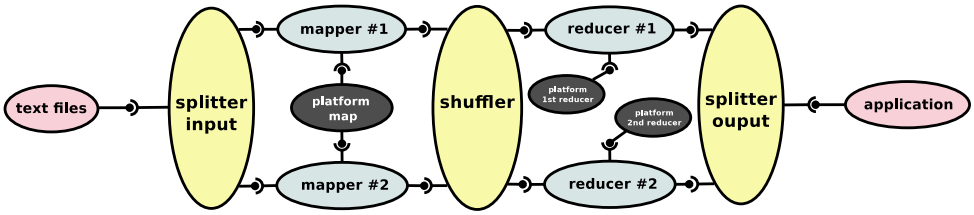
Fonte: Próprio Autor

Figura 11 – Múltiplas Facetas em SPLITTER e SHUFFLER



Fonte: Próprio Autor

Figura 12 – Sistema *MapReduce* Simples



Fonte: Próprio Autor

sinalizar que a faceta encontra-se no estado habilitado ou desabilitado, respectivamente. É importante enfatizar que a ativação de `ACTIVE` e `INACTIVE` só pode ser realizada entre chamadas de `BEGIN_CHANGE_STATUS` e `END_CHANGE_STATUS`.

Por sua vez, cada faceta alimentadora de um conector possui duas portas de ações: **task_feeder_read_chunk** e **task_feeder_chunk_ready**. A primeira possui semântica análoga à porta **task_collector_read_chunk** das facetas coletoras, enquanto a última possui uma ação com o nome `CHUNK_READY`, utilizada para sinalizar ao componente *workflow* que um *chunk* está pronto para ser lido pelo agente de computação conectado na porta **feed_pairs**.

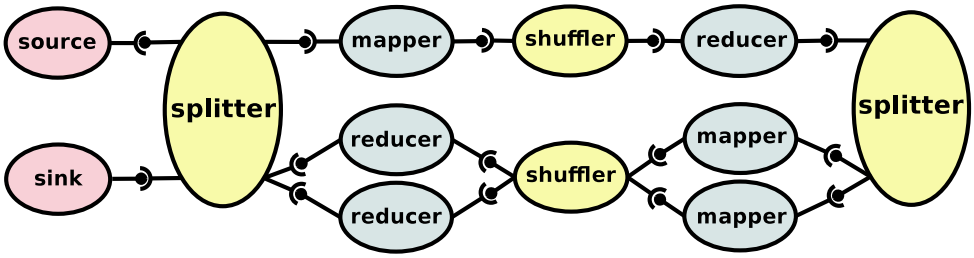
Finalmente, cada agente de computação (MAPPER ou REDUCER) possui uma única porta de ações (**task_map** ou **task_reduce**) que possui ações com os seguintes nomes: `READ_CHUNK`, `PERFORM` e `CHUNK_READY`. A primeira e a última possuem semântica análoga às ações de mesmo nome descritas para os conectores, enquanto `PERFORM` deve ser ativada após `READ_CHUNK` completar-se, a fim de que o agente de computação aplique sua função (mapeamento ou redução) sobre os pares contidos no *chunk*.

A Figura 12 apresenta a arquitetura de um sistema MapReduce simples, composto de uma única etapa de mapeamento/redução.Tanto a etapa de mapeamento quanto a etapa de redução são realizadas por um par de agentes. Todavia, enquanto os dois agentes de mapeamento estão executando sobre a mesma plataforma virtual (**platform map**), os dois agentes de redução estão executando sobre plataformas distintas (**platform 1st reducer** e **platform 2nd reducer**). Essa arquitetura pode, por exemplo, ser usadas para uma contagem de ocorrências de palavras em um conjunto de arquivos de texto, lidos a partir de um repositório de dados representado por um componente chamado **text files**, da espécie fonte de dados, sendo o resultado do processamento lido pelo componente **application**. Para conectar os agentes de saída, respositório de entrada e aplicação foram necessários dois conectores SPLITTER (**splitter input** e **splitter output**) e um conector SHUFFLER (**shuffler**).

Por sua vez, a Figura 13 apresenta um exemplo de sistema MapReduce iterativo composto de duas etapas de mapeamento e redução, onde, na segunda etapa, pares de agentes de mapeamento e de redução são empregados.

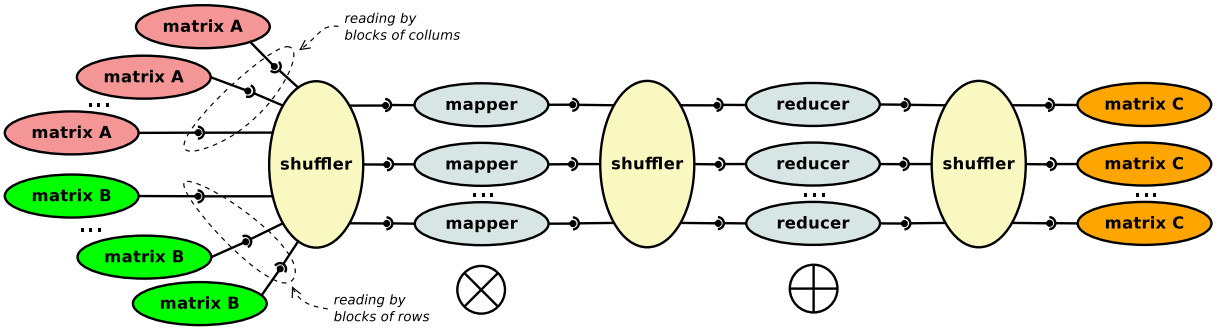
Em sistemas iterativos, condições que governam a terminação no código de orquestração podem ser sinalizadas através das funções de particionamento ou de computação (mapeamento ou redução), que podem ser customizadas de acordo com o sistema MapReduce. Para isso, cada componente PARTITIONFUNCTION, MAPFUNCTION e REDUCEFUNCTION expõe uma porta de ações cujo tipo pode ser customizado através dos contratos contextuais dos conectores SHUFFLER e SPLITTER (parâmetro ***partition_function_action_port_type***), bem como dos agentes MAPPER (***map_function_action_port_type***) e REDUCER (***reduce_function_action_port_type***), de acordo com a semântica do sistema de computação paralela.

Figura 13 – Sistema *MapReduce* Iterativo



Fonte: Próprio Autor

Figura 14 – Sistema *MapReduce* para Multiplicação de Matrizes



Fonte: Próprio Autor

3.4.3 Multiplicação de Matrizes Esparsas sobre MapReduce

A Figura 14 apresenta a arquitetura de um sistema *MapReduce* voltado a multiplicação de duas matrizes esparsas, cada uma das quais armazenadas de forma distribuída em um conjunto de fontes de dados, devido a sua grande escala.

As matrizes (**A** e **B**) são fornecidas pelas fontes de dados através de portas de serviços. Essas portas são ligadas a facetas coletoras de um conector SHUFFLER através de ligações capazes de entregar ao conector pares da forma $\langle x, [\langle y_i, v_i\rangle \mid i \in \{0,\ldots,n\}]\rangle]$, onde:

- para a matriz **A**, $v_i = \mathbf{A}_{x,y_i}$;
- para a matriz **B**, $v_i = \mathbf{B}_{y_i,x}$.

Ou seja, as ligações às fontes de dados da matriz **A** entregam colunas dessa matriz, enquanto as ligações às fontes de dados da matriz **B** entregam linhas suas, usando a chave para alinhar os blocos correspondentes. Lembre que, na multiplicação de matrizes, o número de colunas de **A** é igual a o número de colunas de **B**, o que vale também para a divisão em blocos.

Ao final da operação do primeiro SHUFFLER, temos pares da forma

$$\left\langle x, \left[[\langle y_i^a, v_i^a\rangle \mid i \in \{0,\ldots,n\}], \left[\left\langle y_j^b, v_j^b \right\rangle \mid i \in \{0,\ldots,m\} \right] \right] \right\rangle$$

, os quais definem o alinhamento de dados esperado por um regimento de agentes de mapeamento ($\otimes$), cuja função de mapeamento recebe cada um desses pares e realiza multiplicações $v_a \times v_b$, para todo par v_a e v_b, tal que $v_a \in [\langle y_i^a, v_i^a\rangle \mid i \in \{0,\ldots,n\}]$ e $v_b \in \left[\left\langle y_j^b, v_j^b \right\rangle \mid i \in \{0,\ldots,m\} \right]$. Os valores calculados são devolvidos pelo agente de mapeamento como pares $\left\langle \left\langle y_i^a, y_j^b \right\rangle, v_a \times v_b \right\rangle$, os quais são agrupados pelo segundo conector SHUFFLER, determinando elementos $\mathbf{C}_{y_i^a, y_j^b}$ (matriz de saída) que serão calculados por um regimento de agentes de redução ($\oplus$) apenas pela soma da lista de valores associados a cada chave. Ao final, um conector SPLITTER distribui os

elementos da matriz de saída em um conjunto de fontes de dados, onde a matriz de saída deve ser armazenada.

3.5 Considerações Finais

Foi apresentado neste capítulo uma visão geral da nuvem computacional HPC Shelf, onde são delineados seus atores e sua arquitetura.

A nuvem HPC Shelf utiliza o modelo de programação orientado a componentes voltado às necessidades de aplicações com requisitos de HPC. Essa nuvem parte das abstrações do modelo Hash e busca proporcionar um ambiente de execução de alto desempenho para aplicações de ciências computacionais e de engenharias.

Foram descritos os diversos perfis de intervenientes da HPC Shelf, o especialista, o provedor de aplicações, o desenvolvedor de componentes e o mantenedor de plataformas. Além dos intervenientes, foram descritos os elementos arquiteturais que definem a HPC Shelf, relecionando-os com os papéis dos intervenientes.

O próximo capítulo apresentará o Alite, o sistema de de contratos contextuais da HPC Shelf, voltado à seleção de componentes com base em requisitos funcionais e não-funcionais impostos pelas aplicações da HPC Shelf.

4 ALITE: O SISTEMA DE CONTRATOS CONTEXTUAIS

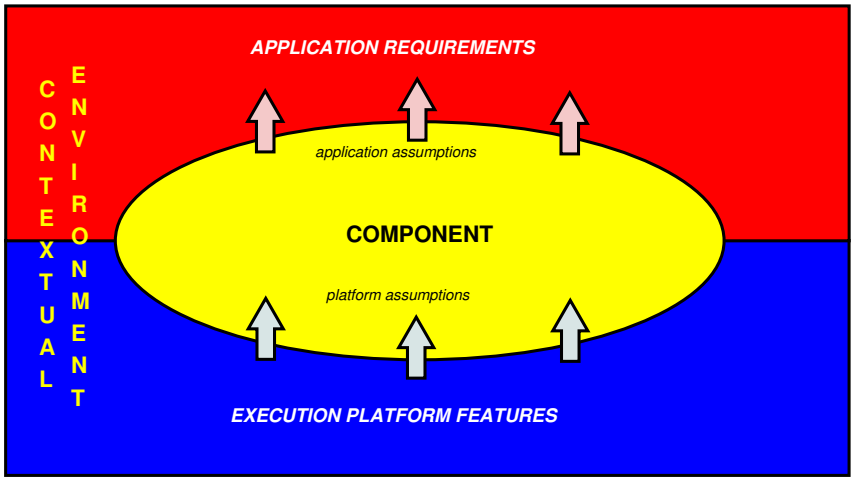
Em um ambiente de desenvolvimento de *software* orientado a componentes, os componentes que formam sistemas de computação são definidos através de interfaces precisas, que definem as suas propriedades que interessam à aplicação, bem como os meios como interagem com os demais componentes. O formato das interfaces é preestabelecido e define um contrato entre um componente dito cliente e outro componente dito provedor (WANG; QIAN, 2005). Dessa forma, a coexistência de um múltiplas implementações de um componente, com diferentes características não-funcionais adaptadas ao ambiente e/ou ao contexto de execução, é uma possibilidade verdadeira e comumente explorada na implantação e instanciação de sistemas orientados a componentes em geral.

Em ambientes heterogêneos de HPC de larga escala, ou seja, ambientes envolvendo o emprego de múltiplas plataformas de computação paralela, com variações arquiteturais em diversas dimensões possíveis, a possibilidade de dispor de componentes especialmente adaptados a explorar suas características peculiares mostra-se crucial para aproveitamento máximo do seu potencial de desempenho. A viabilidade desse tipo de ambiente tem sido alavancado devido a tecnologia de nuvens computacionais aplicadas aos serviços de HPC, possibilitando meios de abstrações sobre a complexidade de administração e integração de recursos heterogêneos de computação. Dentro desse contexto, a contribuição central desta Tese diz respeito a proposta de uma abstração para lidar com a administração e integração recursos de HPC necessários a composição de sistemas de computação paralela de larga escala orientados a componentes. Tais recursos, representados sob a abstração de componentes, podem estar disponíveis através de serviços pré-existentes de nuvens computacionais de infraestrutura, tanto IaaS quanto DaaS, públicas ou privadas. Além disso, podem estar disponíveis através das novas classes de serviços introduzidas pela plataforma HPC Shelf, notadamente a oferta tanto de componentes que representam variações de implementações de algoritmos de computação paralela quanto dos próprios sistemas de computação paralela que podem ser construídos sobre essa plataforma pela composição de componentes das diferentes espécies suportadas.

O sistema de contratos contextuais proposto nesta Tese, o qual recebe a denominação de Alite[1], oferece um mecanismo de seleção automática de componentes de acordo com um

[1] O termo Alite deriva da sigla C3S, a qual coincide com a fórmula, na notação usada pela indústria de cimento, de uma espécie de cimento chamado de Alite(CASTRO *et al.*, 2011). Essa analogia vem do fato de que o sistema de seleção de componentes baseado em contratos contextuais atua, metaforicamente, como uma cola por meio da qual componentes e aplicações da HPC Shelf são ligados.

Figura 15 – Ambiente Contextual



Fonte: Próprio Autor

determinado conjunto de propriedades do seu *contexto* de execução, representado pela aplicação e plataforma virtual hospedeira (Figura 15). Tais propriedades são definidas por meio de uma abstração para classes de componentes, chamadas de *componentes abstratos*, que representam componentes que realizam um mesmo interesse porém implementados de forma diferente a fim de atender a requisitos específicos da aplicação hospedeira e características arquiteturais peculiares da plataforma de computação paralela alvo. Portanto, *componentes abstratos* determinam um contexto de implementação, através do qual são especificados *contratos contextuais* que descrevem os requisitos de aplicações e características das plataformas virtuais que se mostram relevantes nas decisões de implementação. Tais contratos contextuais podem ser vistos como uma noção de tipos para componentes da plataforma HPC Shelf.

Embora a implementação do Alite tenha sido concebida para resolver contratos contextuais da HPC Shelf, o Alite pode ser usado em outros sistemas orientados à componentes onde seja necessária a seleção de componentes de *software*, seleção de componentes de *hardware* ou ainda a seleção de ambos simultaneamente. Para tal, o Alite necessita que o modelo de componentes utilizado respeite as características do modelo Hash de componentes, de modo que os componentes sejam descritos com seus interesses sob a forma de parâmetros e que os contratos contextuais que os valoram tenham definição hierárquica bem definida.

Portanto, o Alite é responsável por prover mecanismos de seleção de componentes que melhor se adequem às necessidades de sistemas de computação paralela da HPC Shelf, levando em consideração tanto requisitos da aplicação que dizem respeito a funcionalidade esperada do componente quanto a características da infraestrutura de computação paralela sobre

os quais devem executar. Entretanto, também por meio da abstração de contexto, o Alite permite ainda a especificação de:

- *critérios de qualidade*, os quais definem restrições de qualidade a partir de suposições sobre as plataformas alvo, de forma análoga aos acordos de nível de serviço utilizados em ambientes de nuvens computacionais;
- *critérios de custo*, que auxiliam a filtragem dos componentes compatíveis com o orçamento de provedores e especialistas; e
- *critérios de ranqueamento*, que permitem controlar a ordenação de alternativas de componentes que atendem a um mesmo contexto delimitado por um contrato contextual, definidos pela plataforma de forma a delimitar suas políticas globais de uso de recursos (componentes).

Um componente de computação possui, associado a ele, um componente da espécie plataforma (plataforma virtual), representado por um contrato contextual também conhecido como *perfil de plataforma*. O perfil de plataforma de um componente de computação especifica quais suposições guiarão a implementação do componente de computação acerca das características da plataforma virtual sobre a qual deverá ser instanciado. Portanto, um *componente sistema* é definido como um par composto por um componente de computação e uma plataforma virtual compatível com o seu perfil de plataforma. O componente sistema é uma definição central no projeto do Alite, tendo em vista a estreita relação entre o *software* e o *hardware* em sistemas HPC.

De acordo com a definição de nuvens computacionais, apresentada na Seção 2.2, a quantidade de recursos computacionais deve transparecer aos usuários a ideia de recursos ilimitados, o que sugere a existência de tamanha quantidade de recursos disponíveis que um usuário não consiga perceber as limitações. Sabemos que, para aplicações em geral, fora do contexto HPC, que satisfazem-se com *hardware* convencional, a ilusão de recursos infinitos é viável para grandes provedores. Porém, para um ambiente de serviços de HPC baseado em nuvem computacional, existe uma maior variedade de plataformas de execução, devido à corrida contínua para conquista de melhor desempenho, com a consequente introdução e consolidação de novas tecnologias de processamento em curtos espaços de tempo. Por isso, tendem a ser heterogêneas e apresentam características arquiteturais peculiares entre si. Dessa forma, a suposição de recursos ilimitados é enfraquecida, uma vez que apresentam quantidades restritas de cada tipo de plataforma de execução. Enfim, a grande variabilidade entre plataformas de

computação paralela agrava o problema da escolha dos componentes, reforçando a importância do Alite como contribuição no cenário de serviços de HPC sob a abstração de nuvem computacional.

O mecanismo de resolução de contratos contextuais requer um conjunto de abstrações para representar os componentes e o seu contexto, os quais precisam ser definidos para a compreensão do algoritmo de resolução. A apresentação das abstrações e conceitos introduzidos pelo Aliteé realizada na Seção 4.1. Na Seção 4.2, é apresentado um exemplo envolvendo componentes para resolução de sistemas lineares. Finalmente, na Seção 4.3, é apresentado o processo de resolução de contratos contextuais, incluindo o detalhamento do algoritmo de resolução.

4.1 Conceitos e Abstrações

Assim como as demais plataformas de desenvolvimento orientado a componentes, a HPC Shelf separa a especificação e a implementação dos componentes utilizando interfaces como contratos entre a especificação e a implementação. Isso é realizado através do Alite, concebido com base no *Hash Type System*(HTS), introduzido por Carvalho-Junior *et al.* (2016), distinguindo-se componentes abstratos e componentes concretos.

Um componente abstrato define uma categoria de componentes (concretos) que realizam um determinado interesse, de uma ou mais aplicações, usando diferentes implementações alternativas que atendem a diferentes contextos de implementação. Uma *assinatura contextual* é o elemento da constituição de um componente abstrato que o permite abstrair-se de um contexto de implementação particular, ou seja, dos requisitos específicos de uma aplicação hospedeira e das características particulares da arquitetura de uma classe de plataformas virtuais alvo que guiam as decisões de sua implementação, de forma que a implementação do componente atenda ao seu interesse da forma mais eficaz e eficiente possível. Um *contrato contextual* é definido ao associar-se um componente abstrato a um contexto de implementação compatível com a sua assinatura contextual, particularizando as suposições e decisões usadas de fato na implementação de um componente concreto. Contratos contextuais definem uma noção de tipo para componentes da HPC Shelf. No restante desta seção, busca-se uma formalização para o conceito de contrato contextual.

Figura 16 – Sintaxe para Componentes Abstratos e Contratos Contextuais

⟨*abstract_component*⟩ ::= ⟨*contextual_signature*⟩ ⟨*inner_components*⟩ '**{**' ⟨*units*⟩ '**}**'

⟨*contextual_signature*⟩ ::= '**[**' ⟨*constraint_list*⟩ '**]**' | ε
⟨*constraint_list*⟩ ::= ⟨*constraint*⟩ '**,**' ⟨*constraint_list*⟩ | ⟨*constraint*⟩
⟨*constraint*⟩ ::= ⟨*contextual_parameter*⟩ | ⟨*parameter_binding*⟩
⟨*parameter_binding*⟩ ::= ⟨*parameter_id*⟩ '**==**' ⟨*parameter_id*⟩
⟨*parameter_id*⟩ ::= PARAMETER_ID ('**-**' PARAMETER_ID)*
⟨*contextual_parameter*⟩ ::= PARAMETER_ID ⟨*subtyping_direction*⟩ COMPONENT_ID
⟨*subtyping_direction*⟩ ::= '**:>**' | '**<:**' | '**=**'

⟨*contextual_contract*⟩ ::= '**[**' ⟨*argument_list*⟩ '**]**'
⟨*argument_list*⟩ ::= ⟨*argument*⟩ ('**,**' ⟨*argument*⟩)*
⟨*argument*⟩ ::= ⟨*parameter_id*⟩ '**=**' COMPONENT_ID| ⟨*parameter_id*⟩ '**=**' PARAMETER_ID

⟨*inner_components*⟩ ::= '**(**' ⟨*inner_component_list*⟩ '**)**' | ε
⟨*inner_component_list*⟩ ::= ⟨*inner_component*⟩**,** ⟨*inner_component_list*⟩ | ⟨*inner_component*⟩
⟨*inner_component*⟩ ::= INNER_COMPONENT_ID **:** ⟨*contextual_contract*⟩
| INNER_COMPONENT_ID **:** ⟨*parameter_id*⟩

⟨*units*⟩ ::= ⟨*unit*⟩ ('**,**' ⟨*unit*⟩)*
⟨*unit*⟩ ::= UNIT_ID '**[**' ⟨*slice_list*⟩ '**]**'
⟨*slice_list*⟩ ::= ⟨*slice*⟩ ('**.**' ⟨*slice*⟩)*
⟨*slice*⟩ ::= INNER_COMPONENT_ID '**.**' UNIT_ID

Fonte: Próprio Autor

4.1.1 Componentes Abstratos

Na Figura 16, é apresentada uma sintaxe para descrição de componentes abstratos, definidos pelos seguintes elementos:

- uma assinatura contextual (⟨*contextual_signature*⟩), composta por um conjunto, possivelmente vazio, de *assertivas contratuais*, cada qual representando uma suposição de implementação específica para o componente;
- um conjunto, possivelmente vazio, de *componentes aninhados* (⟨*inner_components*⟩), associados a contratos contextuais que definem os seus tipos, os quais representam as dependências do componente hospedeiro em questão em relação a instâncias de outros componentes;
- um conjunto não-vazio de unidades (⟨*units*⟩) a serem implementadas pelos componentes concretos segundo as restrições de seu contrato contextual e aproveitando as dependências em relação aos componentes aninhados.

Componentes aninhados e unidades são requisitos do modelo Hash de componentes, enquanto os parâmetros de contexto são uma abstração própria do Alite para representar contexto

de implementação. Para o sistema de resolução de contratos do Alite, é suficiente concentrar-se nas assinaturas contextuais e contratos contextuais, respectivamente derivados a partir das variáveis $\langle contextual\_signature \rangle$ e $\langle contextual\_contract \rangle$.

4.1.2 Assinaturas Contextuais

Em uma assinatura contextual, há dois tipos de assertivas contratuais, *parâmetros de contexto* ($\langle contextual\_parameter \rangle$) e *ligações de parâmetros* ($\langle parameter\_binding \rangle$), descritos nas seções a seguir.

4.1.2.1 Parâmetros de Contexto

Um parâmetro de contexto ($\langle contextual\_parameter \rangle$) especifica uma propriedade do contexto de implementação dos componentes representados pelo componente abstrato. Pode ser de dois tipos:

- *ofertado* ou *covariante*, representando uma suposição que o ambiente de execução (aplicação hospedeira e plataforma alvo) faz a respeito do componente;
- *requerido* ou *contravariante*, representando uma suposição que o componente faz a respeito do ambiente de execução.

Conforme a sintaxe da Figura 16, um parâmetro de contexto de um componente abstrato é composto por:

- um nome (PARAMETER_ID), usado para referir-se ao parâmetro em contratos contextuais;
- um modificador que define se um parâmetro é ofertado ($:>$), requerido ($<:$) ou tem o seu valor determinado ($=$) para efeito de derivação de contratos;
- uma restrição contextual, definido por um *qualificador de nome*, o qual restringe o conjunto de contratos contextuais (*argumentos de contexto*), que podem ser associados a ele na definição de um contrato contextual para o componente abstrato em questão.

4.1.2.2 Componentes Qualificadores

Na HPC Shelf, qualificadores são espécies de componentes usados especificamente para os propósitos de caracterizar características não-funcionais de componentes em contratos contextuais. Como um componente Hash, qualificadores também são formados por unidades e podem ser compostos por sobreposição a partir de outros qualificadores. Além disso, podem

possuir assinaturas contextuais não-vazias, notadamente compostas de parâmetros de contextos restringidos por contratos contextuais de outros qualificadores.

Assume-se que todo qualificador de assinatura vazia possui um único componente concreto associado, correspondendo, em termos de teoria de tipos, a um tipo unitário. Por sua vez, no caso de qualificadores com assinatura não vazia, componentes concretos podem existir ou não para certos contratos construídos a partir desses qualificadores, representando uma *dependências funcionais entre qualificadores*. Um exemplo da aplicação de contratos de qualificadores para definir tais dependências funcionais, demonstrando sua utilidade, será mostrado na Seção 4.2.

4.1.2.3 Qualificadores de Nome

A assinatura contextual de qualquer componente abstrato deve possuir pelo menos um parâmetro de contexto covariante, identificado por ***name***, cuja restrição contextual deve ser um *qualificador de nome*, usado apenas para nomear o componente abstrato. Qualificadores de nome constituem uma tipo especial de qualificadores de assinatura vazia derivados do qualificador NAME, que aparecem na sintaxe de assinaturas contextuais através da unidade léxica COMPONENT_ID. Logo, é possível afirmar que NAME é a restrição contextual básica (*default*) do parâmetro ***name***, presente na assinatura de qualquer componente abstrato. Ao referir-se a um qualificador de nome em uma restrição contextual, seja em uma assinatura ou em um contrato contextual, faz-se referência a um componente abstrato supostamente registrado no catálogo de componentes com aquele nome, o qual possui sua própria assinatura contextual, possivelmente não vazia.

A relação de derivação entre componentes abstratos *estruturalmente compatíveis* é introduzida nominalmente no catálogo de componentes, associando-se dois qualificadores de nome (e.g. C e C') de forma a definir que um é subtipo do outro ($C' <: C$). A relação de combatibilidade estrutural entre componentes abstratos está definida na Seção 4.1.6.

Em assinaturas contextuais de componentes abstratos, o uso do modificador '=' com o parâmetro ***name*** especifica que não podem mais ser derivados componentes abstratos a partir do componente abstrato em questão, o qual é chamado de componente abstrato final. Portanto, a partir de um componente abstrato final, é possível apenas a especificação de um contrato contextual.

4.1.2.4 Ligações de Parâmetros

Em uma assinatura contextual de componente abstrato, ligações de parâmetros associam os valores de dois parâmetros de contexto, de modo que devem possuir os mesmos valores em contratos contextuais definidos a partir da assinatura. Em uma ligação de parâmetro é possível fazer referência tanto a parâmetros da própria assinatura como a parâmetros de assinaturas de componentes abstratos referenciados em suas restrições contextuais, de forma recursiva. Esse é o motivo do uso, em ligações de parâmetros, da notação qualificada (⟨*parameter_id*⟩) para fazer referência a parâmetros de contexto.

4.1.3 Contratos Contextuais

Um contrato contextual (⟨*contextual_contract*⟩) é a associação de argumentos de contexto a parâmetros de contexto de uma assinatura contextual de um componente abstrato (⟨*argument_list*⟩) que não tenham sido fechados com o modificador '='. Para isso, um argumento de contexto (⟨*argument*⟩) associa um parâmetro de contexto da assinatura, possivelmente de restrições contextuais aninhadas usando a notação qualificada, a uma restrição contextual definida por um qualificador de nome, usando o modificador '='[2]. Entretanto, em contratos contextuais de componentes aninhados, a restrição contextual pode também referir-se a um identificador de parâmetro de contexto do componente hospedeiro (PARAMETER_ID). Em um contrato contextual bem formado, toda restrição contextual de argumento é compatível com a restrição contextual do parâmetro ao qual o argumento está associado. A compatibilidade pode ser uma relação de subtipo ou de supertipo, dependendo se o parâmetro é covariante ou contravariante, respectivamente. A relação de compatibilidade entre contratos contextuais será definida de maneira mais detalhada na Seção 4.1.7.

4.1.4 Componentes Aninhados

O Alite, por requisitos do modelo Hash, tem seus componentes definidos por sobreposição, de forma que um componente pode ser definido a partir da composição de unidades de

[2] De fato, apenas do ponto de vista sintático, seria possível definir um contrato contextual como um componente abstrato final para o qual todos os parâmetros estão fechados. Entretanto, contratos contextuais e assinaturas de componentes abstratos finais possuem semântica diferente para parâmetros que não são explicitamente mencionados no contrato ou assinatura. Neste caso, enquanto são chamados de parâmetros *livres* no contrato, são apenas formas abreviadas em assinaturas, evitando fazer referência a parâmetros herdados da assinatura do componente abstrato a partir do qual foi derivado.

outros componentes pré-existentes, os quais são denominados componentes aninhados. Um componente aninhado (⟨*inner_component*⟩) é definido por um nome (INNER_COMPONENT_ID), através do qual é referenciado na composição de unidades, e um contrato contextual (⟨*contextual_contract*⟩), que define o seu tipo. Alternativamente ao contrato contextual, o tipo do componente aninhado pode ser associado a um dos parâmetros de contexto do componente abstrato hospedeiro. Dessa forma, o contrato contextual que irá definir o tipo desse componente aninhado só será conhecido em um contrato contextual do componente hospedeiro, onde um argumento é associado ao parâmetro.

4.1.5 Unidades

Finalmente, as unidades de um componente abstrato definem as partes distribuídas do componente paralelo. Para o caso de um componente de computação, por exemplo, representam os agentes de processamento distribuído responsáveis por executar o algoritmo paralelo, ou processos. São as unidades que de fato irão definir o comportamento do componente durante a sua execução. É obrigatório que cada componente tenha ao menos uma unidade. Uma unidade (⟨*unit*⟩) possui um nome UNIT_ID, usado para referir-se a ela, e um conjunto de fatias, referentes a unidades de componentes aninhados que podem ser compostas para definir a realização de seu interesse, através da composição por sobreposição (Seção 2.5.3). Assim, sintaticamente, uma *fatia* (⟨*slice*⟩) é definida como um par de um identificador de componente aninhado e um identificador de uma de suas unidades que constitui uma fatia da unidade do componente hospedeiro.

4.1.6 Compatibilidade Estrutural entre Componentes Abstratos

Sejam C e C' qualificadores de nomes associados no catálogo de componentes pela relação de subtipos: $C' <: C$. Para isso, o componente abstrato cujo parâmetro de contexto ***name*** em sua assinatura tem o qualificador de nome C' como restrição contextual deve ser estruturalmente compatível com aquele componente abstrato de nome C.

Um componente abstrato de nome C' é estruturalmente compatível com um componente abstrato C, para fins de derivação, se as seguintes condições são satisfeitas:

- Se U^C e $U^{C'}$ são os conjuntos dos identificadores das unidades de C e C', respectivamente, então $U^C = U^{C'}$;
- Se I^C e Se $I^{C'}$ são os conjunto dos identificadores dos componentes aninhados de C e C',

respectivamente, então $I^C \subseteq I^{C'}$;

- Se X e X' são contratos contextuais respectivamente associados aos componentes aninhados identificados por um certo i em C e C', então X' é subtipo de X;
- Se s é uma fatia de uma unidade u de C' que não existe na unidade u de C, então s é uma unidade de um componente abstrato i introduzido por C' ($i \in I^{C'}$ e $i \notin I^C$);
- se a assinatura contextual de C' possui um parâmetro de contexto de nome $\boldsymbol{p}$ que também existe em C ou qualquer outro componente abstrato na cadeia de componentes abstratos dos quais C é derivado, desde que não fechado a um valor definitivo através do modificador "=", sua restrição contextual deve ser:
  - um subtipo da restrição contextual de todas as ocorrências de $\boldsymbol{p}$ nessa cadeia, caso $\boldsymbol{p}$ seja contravariante (diz-se "estreitamento" da restrição contextual, no caso de ser um subtipo próprio);
  - um supertipo, caso $\boldsymbol{p}$ seja covariante (nesse caso, diz-se "alargamento");

Portanto, essa definição permite que, em uma derivação C' a partir de C podem ser incluídos novos componentes aninhados e novos parâmetros de contexto, desde que os tipos dos componentes aninhados e parâmetros já existentes em C sejam compatíveis, respectivamente. Por sua vez, o conjunto de unidades deve ser mantido, embora seja ainda possível incluir em uma unidade fatias de novos componentes aninhados incluídos em C'.

A definição também garante que não é possível modificar a restrição contextual de um parâmetro de contexto cujo valor já foi fechado em C ou em algum dos componentes abstratos em sua cadeia de derivação. Portanto, obviamente, caso a referência ao parâmetro cuja restrição contextual deseja-se modificar seja feita com o modificador "=", sua restrição contextual não poderá mais ser estreitada em componentes abstratos derivados de C'.

Finalmente, a relação de tipos entre contratos contextuais, que definem restrições contextuais e os tipos de componentes aninhados, é definido na próxima seção.

4.1.7 Relação de Tipos entre Contratos Contextuais

Para que o algoritmo de resolução do Alite seja capaz de determinar se um componente, implementado de forma a atender um certo contrato contextual, pode ser selecionado para ser ligado a um componente aninhado associado a um outro contrato contextual, é determinada uma relação de compatibilidade entre contratos contextuais, por meio de uma relação de subtipos. Dessa forma, um componente concreto que implementa um contrato contextual A poderá ser

utilizado quando é requerido um componente aninhado cujo tipo é definido por um contrato contextual B, se A é um subtipo de B segundo a relação de subtipos, ou seja, $A <: B$.

Sejam dois contratos contextuais T e T', os quais possuem C e C' como componentes abstratos, respectivamente, tais que C' é derivado transitivamente de C.

A relação de subtipos garante que é seguro utilizar um componente concreto que implementa o contrato contextual T' quando um componente que implemente um contrato contextual T é requisitado, ou seja, $T' <: T$ se cada qualificador de nome B' aplicado como argumento de contexto a um parâmetro x em T' é um subtipo do qualificador de nome B aplicado como argumento ao mesmo parâmetro x em T ($B' <: B$), caso o parâmetro seja requerido, ou um supertipo ($B <: B'$), caso o parâmetro seja ofertado.

4.1.8 Parâmetros de Contexto de Qualidade e de Custo

Na plataforma HPC Shelf, pressupõe-se que a implementação de componentes de computação, que implementam algoritmos de computação paralela, está intimamente relacionada à plataforma de computação paralela alvo aonde esses componentes irão ser executados. Por esse motivo, a seleção de componentes se dá em torno de componentes sistema, definidos pelos pares formados por componentes de computação e a plataforma virtual alvo sobre a qual executará. Ao associar um componente de computação à sua plataforma de execução, novas possibilidades de suposições surgem no contexto de implementação para guiar o processo de seleção de componentes, motivadas a seguir.

Considere, por exemplo, um componente capaz de explorar o desempenho de um número arbitrário de nós de processamento. Sendo o especialista um usuário proprietário de uma conta com grande capacidade de recursos para execução, se considerada uma política arbitrária de uso de recursos da nuvem, ele pode solicitar a execução de um componente de maneira a influenciar o algoritmo de resolução para escolher uma plataforma virtual com a quantidade máxima de nós de processamento possível, o que, de acordo com o senso comum, seria uma garantia de que a execução da tarefa seria realizada de maneira mais rápida, em comparação com o cenário onde uma quantidade menor de nós de processamento fosse empregada. Desprezando-se suposições acerca dos limites de escalabilidade inerentes aos algoritmos paralelos, essa ideia estaria correta. Porém, sistemas de computação paralela, representados na HPC Shelf pelos componentes sistema, possuem limitações teóricas de escalabilidade do paralelismo, que fazem com que os algoritmos sofram perdas de desempenho devido a sobrecargas inerentes ao uso

de mais recursos de processamento na execução paralela (GRAMA, 2003). No caso de uma plataforma construída com o recurso de virtualização, também pode-se inserir os custos de inicialização das máquinas virtuais (SHI *et al.*, 2011). Além disso, o especialista pode desejar não utilizar todos seus recursos nesta execução.

Para lidar com a análise da escalabilidade de sistemas de computação paralela, existem diversas técnicas de modelagem analítica para esses sistemas. Dentre essas técnicas, uma das mais conhecidas é a análise da função de *isoeficiência*, a qual permite calcular o número de nós de processamentos que devem ser usados para um certo tamanho de problema a fim de garantir um certo valor pré-estabelecido para a métrica de *eficiência* (GRAMA, 2003). A função de isoeficiência é determinada com o conhecimento de uma outra função que define a sobrecarga da execução paralela, necessária para calcular o tempo de execução da implementação de um algoritmo paralelo para uma determinado tamanho de entrada e número de nós de processamento.

A fim de otimizar o uso de recursos de computação na HPC Shelf, o Alite distingue parâmetros de contexto de componentes abstratos em quatro tipos:

- **aplicação**, referente a suposições da aplicação hospedeira em relação ao componente;
- **plataforma**, referente a suposições da aplicação, ou do próprio componente, em relação às características arquiteturais da plataforma virtual aonde o componente deverá ser instanciado e executar;
- **qualidade**, referente a suposições da aplicação acerca de aspectos do desempenho do componente, que podem incluir, por exemplo, métricas de aferição do seu desempenho computacional (tempo de execução, aceleração, eficiência, etc) e eficiência energética;
- **custo**, referente a suposições da aplicação acerca do custo, em termos monetários, da instanciação e execução de cada componente.

Parâmetros de aplicação e de plataforma já foram tratados nesta seção, de maneira uniforme. Já os parâmetros de qualidade e custo são uma extensão proposta pelo Alite a fim de atender a requisitos de políticas de uso de recursos computacionais na HPC Shelf, tanto globais quanto de aplicações em particular.

Conhecidas as propriedades da aplicação e da plataforma virtual que representam o ambiente de execução do componente, tais como o tamanho de problema e o número de unidades de processamento disponíveis, expressos em argumentos para parâmetros de contexto de aplicação e de plataforma, funções obtidas pela modelagem analítica podem então ser utilizadas para cálculo de argumentos para parâmetros de qualidade do sistema de computação paralela,

tais como eficiência energética, tempo de execução, aceleração, etc, a partir dos quais podem ser inferidos os argumentos para parâmetros de custo, que estabelecem os custos de utilização dos componentes, especialmente plataformas virtuais.

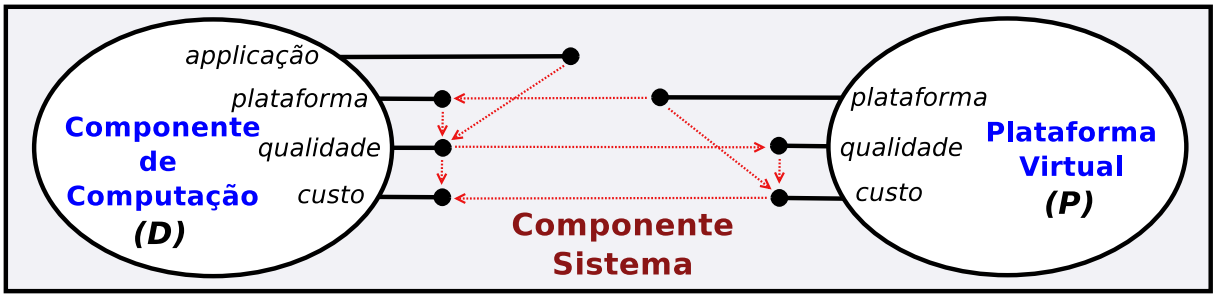
Na situação em que desenvolvedores de componentes de computação não forneçam valores para os parâmetros de qualidade padrão da plataforma, tampouco uma função de como devem ser calculados, seu custo de execução é considerado desconhecido, uma vez que é impossível determinar que recursos da plataforma virtual com a qual comporá um componente sistema serão necessários para a execução. Na melhor das hipóteses, poderá fornecer uma estimativa conservadora sobre o custo, a qual, por consequência, torna menos provável a escolha desse componente pela estratégia de resolução de contratos perante a existência de outros componentes que forneçam parâmetros que determinem suas métricas de qualidade, embora o custo cobrado possa ser menor do que a expectativa estabelecida em contrato na execução real. Dessa forma, supõe-se que desenvolvedores de componentes na HPC Shelf sentiriam-se estimulados a determinar modelos analíticos para os parâmetros de qualidade dos seus componentes, buscando um ajuste fino dessas métricas para diminuição dos custos de seus componentes e evitando penalidades por má reputação na oferta de serviços com o objetivo de otimização de lucros.

Pode-se afirmar que os contratos contextuais definem parte da noção de Acordo de Nível de Serviço (SLA) da nuvem HPC Shelf. A outra parte refere-se à definição das violações e das penalidades aplicadas pela HPC Shelf em caso de não cumprimento do acordado, a qual é definida de maneira global para todo o ambiente da nuvem. Porém, para que os acordos sejam respeitados, algumas metas de desempenho e disponibilidade devem ser definidas, de modo que cada meta descreve um Objetivo de Nível de Serviço (SLO[3]) (STURM *et al.*, 2000). Parâmetros de qualidade e custo, uma vez que modelam métricas de qualidade e de custo, representam a noção de SLOs.

Como a HPC Shelf é uma nuvem de componentes que representam recursos de HPC para aplicações, são exemplos de SLOs: o desempenho medido em termos de tempo de execução; a quantidade de computação a ser realizada pelo componente, medido por número de operações de ponto-flutuante e aritmética inteira; e a movimentação entre hierarquias de memória. Outros parâmetros de qualidade podem também ser considerados relevantes, como parâmetros de disponibilidade, comumente utilizados em nuvens computacionais voltadas para

[3] *S*ervice Level Objective.

Figura 17 – Fluxo do Cálculo de Parâmetros de Componentes



Fonte: Próprio Autor

aplicações comerciais.

Os parâmetros de custo, informados tanto por componentes de computação quanto por componentes da espécie plataforma, representam métricas de custo de execução do componente e uso da plataforma, incluindo custos monetário e energético.

Seja $\langle D, P \rangle$ um par que representa um componente sistema, onde D representa um componente de computação e P representa uma plataforma virtual que atende aos requisitos do contrato contextual do componente plataforma do qual D depende para executar. A Figura 17 sintetiza o fluxo de valoração dos parâmetros de SLO (qualidade e custo de serviço) a partir dos parâmetros de aplicação e plataforma. De fato, é preciso definir os argumentos e parâmetros de contexto do componente sistema $\langle D, P \rangle$ a partir dos argumentos dos parâmetros de contexto de D e P, vistos isoladamente.

D possui argumentos pré-definidos para os parâmetros de aplicação e argumentos para os parâmetros de plataforma que definem as suposições sobre a arquitetura da plataforma para a qual foi desenvolvida a fim de explorar melhor o desempenho potencial. Por sua vez, P possui argumentos para os parâmetros de plataforma que definem as suas características arquiteturais relevantes. Uma vez que a estratégia de resolução de contratos, na fase de seleção, casou D e P em um componente sistema, os argumentos dos parâmetros de plataforma de P são compatíveis com os argumentos dos parâmetros de plataforma de D. Nesse caso, para o componente sistema como um todo, prevalece os argumentos dos parâmetros de plataforma de P, que são mais específicos e, portanto, substituições seguras para os argumentos dos parâmetros de plataforma de D.

A partir dos argumentos dos parâmetros de aplicação e plataforma do componente sistema, respectivamente provenientes de D e P, os parâmetros de qualidade, definidos por D, podem se tornar conhecidos, sejam pré-fixados ou calculados por meio de funções derivadas

de modelagem analítica sobre os valores dos argumento dos parâmetros de plataforma (ex: número de nós de processamento, núcleos de processamento por nó, presença e característica do acelerador, etc) e da aplicação (ex: tamanho do problema, caracterísitca da entrada, etc).

Por sua vez, os argumentos dos parâmetros de custo da plataforma *P* e do componente *D* podem ser calculados a partir dos argumentos dos parâmetros de qualidade do componente sistema, derivados de *D*, usando funções que levam em conta aspectos econômicos. A partir desses custos, a aplicação pode calcular, com auxílio de serviço de contabilização da nuvem, qual o valor monetário devido pelo especialista e o que deverá ser repassado para o provedor da aplicação (montador), desenvolvedor do componente *D* e mantenedor da plataforma *P*.

Os parâmetros de plataformas e SLOs, ou seja, os parâmetros de qualidade e de custo, são pré-definidos pela nuvem HPC Shelf. Porém, uma vez que o arcabouço é genérico, esse conjunto de parâmetros pode evoluir com o tempo, em novas versões da HPC Shelf, mantendo compatibilidade vertical e horizontal. O gerenciamento de nível de serviço se dá pela comparação entre as metas de qualidade, definidas nos contratos contextuais, e os argumentos calculados para os respectivos parâmetros pelos componentes.

Parâmetros de contexto de aplicação, plataforma, qualidade e custo não são os únicos introduzidos por Alite. Na Seção 4.3.2, serão ainda introduzidos os *parâmetros de ranqueamento*, cuja finalidade é especificar políticas de alocações de recursos alternativas suportadas pela plataforma (e.g. HPC Shelf), que podem ser escolhidas dinamicamente pelas aplicações ao executar sistemas de computação paralela. Ainda nessa seção, especificamente na Seção 4.3.2.2, a noção de subtipos entre parâmetros de qualidade e de custo é explicada de maneira mais precisa, em termos de valores de parâmetros de ranqueamento.

4.1.9 Componentes Quantificadores

Uma importante extensão do Alite em relação ao HTS é a introdução de valorações numéricas para parâmetros e argumentos de contexto, ou seja, contratos contextuais podem representar valorações numéricas. Essa extensão é motivada pela necessidade de suportar certos tipos de parâmetros de contexto, principalmente que dizem respeito à propriedades de plataformas virtuais, tais como o número de nós de processamento distribuído e de núcleos de processamento por nó, e na definição de parâmetros de qualidade e custo, os quais podem ser numericamente calculados a partir dos valores de outros parâmetros, como será demonstrado na Seção 4.3. Dessa forma, haverão contratos contextuais para representar valorações inteiras e

reais, bem como relações de subtipos entre eles. Para que isso seja possível, foram embutidos ao Alite componentes abstratos de uma nova espécie, os *quantificadores*, para representar valorações numéricas, juntando-se aos componentes *qualificadores* com o propósito principal de descrever propriedades não-funcionais que regem a implementação, implantação e instanciação de componentes.

Os domínios de quantificadores introduzidos pelo Alite são:

- INT $\downarrow$: um domínio de inteiros com relação de subtipos direta, ou seja, tal que $n <: m \Leftrightarrow n \leq m$, onde n e m são qualificadores inteiros;
- INT $\uparrow$: um domínio de inteiros com relação de subtipos inversa, ou seja, tal que $n <: m \Leftrightarrow m \leq n$, onde n e m são qualificadores inteiros;
- REAL $\downarrow$: um domínio de reais com relação de subtipos direta, ou seja, tal que $n <: m \Leftrightarrow n \leq m$, onde n e m são qualificadores reais;
- REAL $\uparrow$: um domínio de reais com relação de subtipos inversa, ou seja, tal que $n <: m \Leftrightarrow m \leq n$, onde n e m são qualificadores reais.

É importante ressaltar que INT $\downarrow \subset$ REAL $\downarrow$ e INT $\uparrow \subset$ REAL $\uparrow$.

Na definição de um parâmetro de contexto, o formato de uma restrição contextual associado a um quantificador é $\langle n \rangle : \langle d \rangle$, onde $\langle d \rangle$ é o domínio (INT $\downarrow$, INT $\uparrow$, REAL $\downarrow$ ou REAL $\uparrow$) e $\langle n \rangle$ é o valor do domínio (literal inteiro ou real) que define a restrição.

Sejam N e N' dois quantificadores numéricos de INT $\downarrow$ ou REAL $\downarrow$, N' somente poderá ser utilizado quando um quantificador numérico N for requisitado, se $N' \leq N$, uma vez que $N' <: N$ (relação direta). Analogamente, caso fossem quantificadores numéricos de INT $\uparrow$ e REAL $\uparrow$, $N' \geq N$, uma vez que $N <: N'$ (relação inversa). Portanto, a *restrição contextual mínima* em domínios de quantificadores diretos (INT $\downarrow$ ou REAL $\downarrow$) é um valor abstrato referenciado como $+\infty$, definido de modo que $v <: +\infty$ para qualquer v pertencente ao domínio. Analogamente, $-\infty$ é a restrição contextual mínima para domínios de quantificadores inversos, uma vez que, nesse domínio, $v <: -\infty$ para qualquer v. A restrição contextual mínima é o valor *default* para domínios de quantificadores, sendo assumidos quando um valor específico não for informado na restrição contextual de um parâmetro de contexto. Ou seja, é possível definir a restrição contextual de um parâmetro do contexto apenas referindo-se ao domínio ($\langle d \rangle$), ignorando o valor da restrição ($\langle n \rangle$).

4.1.9.1 Cenários de Exemplos de Quantificadores

Considere um parâmetro de contexto requerido por um componente de computação (contravariante) que representa o *número mínimo de nós de processamento* esperados na plataforma virtual que o componente deve executar. Um componente que espera ser alocado em uma plataforma com no mínimo N nós de processamento poderia ser selecionada em um contexto onde é exigida uma computação que espera ser alocado em uma plataforma virtual com no mínimo M nós de processamento, desde que $N \geq M$. Para esse parâmetro, o limite seria $+\infty$, no domínio INT $\downarrow$. Por outro lado, caso o parâmetro representasse o *número máximo de nós de processamento* exigidos na plataforma virtual por parte do componente de computação a fim de garantir alguma outra propriedade de desempenho (e.g. eficiência), o limite seria 1 no domínio INT $\uparrow$.

De maneira análoga, uma plataforma cujos nós de processamento possuem N núcleos poderia ser selecionada em um contexto onde é exigida uma plataforma com um número M, menor ou igual que N, nós de processamento ($M \leq N$), pois se fossem exigidos mais de N núcleos, a plataforma selecionada não teria como supri-los. Para esse parâmetro de contexto, o limite seria 1, no domínio dos inteiros INT $\uparrow$.

Por outro lado, considere um parâmetro de contexto ofertado que representa a eficiência garantida da implementação de um algoritmo paralelo, ou seja, a fração do tempo da execução total do componente no qual suas unidades estão realizando trabalho útil, ignorando-se as sobrecargas relativas à execução paralela que não existem na sua versão sequencial. Portanto, a eficiência é medida no intervalo real $[0,1]$. Um componente de computação com eficiência mínima garantida de $0,7$ (70%), quando executando sobre uma plataforma virtual cujo contrato contextual seja compatível com o contrato contextual da plataforma virtual indicada em seu componente aninhado da espécie plataforma, pode ser utilizado em um contexto onde é exigido um componente de computação com eficiência mínima de $0,6$. Neste caso, tendo em vista que o parâmetro de contexto é ofertado (portanto, covariante), tem-se que $0,7 <: 0,6$. Portanto, a restrição contextual do parâmetro de contexto seria 0 (zero) e o domínio de quantificadores seria REAL $\uparrow$.

Com base neste último exemplo, podemos levantar uma questão relevante em relação aos parâmetros de qualidade e custo, notadamente aqueles limitados por valorações numéricas, no que diz respeito à definição da relação de subtipos entre os valores. De maneira informal, e notando que parâmetros de qualidade e de custo são parâmetros ofertados por um componente,

entende-se que um componente cujo contrato contextual prevê um valor "melhor" (mais desejado) para um parâmetro de qualidade possa ser usado no contexto onde um componente com um valor "pior" (menos desejado) é requisitado. Logo, a relação de subtipos vai na direção do "pior" (supertipo) para o "melhor" (subtipo), ou seja, o subtipo é considerado o "melhor" valor. Se o "melhor" valor é o maior, como no caso da eficiência, utiliza-se o domínio de relação direta ($\uparrow$), enquanto se o "melhor" valor é o menor, como no caso de parâmetros de custo monetário, utiliza-se a relação inversa ($\downarrow$). No caso de parâmetros que medem recursos da plataforma, o "melhor", como regra geral, pode corresponder tanto à maior quanto à menor quantidade de recursos, dependendo da semântica do parâmetro, como no caso do número de processadores ou núcleos de processamento (mínino *versus* máximo).

Considere agora um parâmetro de contexto ofertado (covariante) que define o tempo de *startup* da interconexão entre os nós de processamento da plataforma virtual alvo, em nanosegundos, também no domínio INT $\downarrow$, com valor limitante $+\infty$. Nesse caso, um componente perfil de plataforma aninhado a um componente de computação cuja implementação assume que o tempo de *startup* da interconexão entre os nós de processamento da plataforma virtual alvo é X poderia ser usada no contexto onde uma plataforma com tempo de *startup* Y, maior que X, é exigida. A generalização sobre o argumento de contexto que representa o tempo de *startup*, a fim de obter um subtipo do componente de computação, decrementaria o valor do argumento, buscando uma plataforma que tenha menor tempo de *startup* ("melhor" valor, para o tempo de *startup* de uma interconexão) e portanto pudesse tornar viável o uso no contexto.

Finalmente, considere um parâmetro de contexto ofertado que representa a *compute capability* das GPUs oferecidas por uma plataforma virtual em seus nós de processamento. Essa característica revela a versão de CUDA, bem como que funcionalidades desta versão, são suportados pela GPU. Uma plataforma virtual com GPUs de *compute capability* 5 poderia ser utilizada em um contexto onde é requisitada uma plataforma virtual com GPUs de *compute capability* 4, considerando que se trata de uma parâmetro covariante. Intuitivamente, componentes que são selecionados para fazer uso das particulares da versão 4 podem ter tais particulares presentes na versão 5, devido a retrocompatibilidade. Nesse caso, o valor limite do parâmetro de contexto, dentro do domínio INT $\uparrow$, seria 1, ou seja, o menor valor suportado para *compute capability* de uma GPU.

4.2 Exemplo: Resolvedores de Sistemas Lineares

Com a finalidade de exemplificar cenários de uso de contratos contextuais para expressar propriedades do contexto de implementação de componentes, utilizaremos componentes hipotéticos que implementam um contrato contextual de um componente abstrato chamado LINEARSYSTEMSOLVER.

Os componentes concretos de LINEARSYSTEMSOLVER implementam resolvedores de sistemas lineares $A_{m\times n} \cdot \mathbf{x}_n = \mathbf{v}_m$, onde A é a matriz de coeficientes, $\mathbf{x}$ é um vetor de incógnitas, $\mathbf{v}$ é o vetor de constantes e os índices m e n representam as dimensões das matrizes e vetores envolvidos, os quais diferenciam-se uns dos outros de acordo com suposições a respeito do contexto onde serão utilizados, expressas pelo conjunto de parâmetros de contexto discutidos a seguir.

A Figura 18(a) apresenta a assinatura contextual de LINEARSYSTEMSOLVER segundo a sintaxe sugerida na Figura 16, onde pode-se notar que apenas o parâmetro ***name*** é ofertado (:>), enquanto os demais são requeridos (<:). Os parâmetros de contexto de aplicação são:

- ***name***: interesse oferecido pelo componente, i.e. resolução de sistemas lineares;
- ***max_dim_size_m***: quantidade máxima de linhas da matriz de coeficientes (A);
- ***max_dim_size_n***: quantidade máxima de colunas da matriz de coeficientes;
- ***element_type***: tipo de representação numérica;
- ***format***: estrutura de dados que implementa a matriz A.

A assinatura contextual do tipo do parâmetro ***format***, MATRIXFORMAT, encontra-se na Figura 18(b), com um único parâmetro ***value_type***, o qual, na assinatura de LINEARSYSTEMSOLVER, é ligado ao valor do parâmetro ***element_type***. Dessa forma, um contrato de LINEARSYSTEMSOLVER só é válido para valores iguais de ***element_type*** e ***format-value_type***.

Por sua vez, LINEARSYSTEMSOLVER possui um único parâmetros de contexto de plataforma, chamado ***virtual_platform***, do tipo CLUSTER, cuja assinatura contextual é apresentada na Figura 18(c). Note que, na assinatura de LINEARSYSTEMSOLVER, a restrição contextual do parâmetro **node** é estreitada de NODE para seu subtipo direto MULTIPROCESSORNODE, descrevendo a assinatura de um tipo de nó de processamento hipotético composto por dois ou mais processadores de 1 ou mais núcleos e uma unidade aceleradora do tipo GPU. A assinatura de MULTIPROCESSORNODE está apresentada na Figura 18(d). A seguir, breves descrições de cada parâmetro de contexto de plataforma acessíveis a partir da assinatura de

Figura 18 – Assinatura Contextual do componente LINEARSYSTEMSOLVER

(a) $\begin{bmatrix} \textbf{\textit{name}} :> \text{LINEARSYSTEMSOLVER}, \\ \textbf{\textit{max\_dim\_size\_m}} <: \text{INT} \downarrow, \\ \textbf{\textit{max\_dim\_size\_n}} <: \text{INT} \downarrow, \\ \textbf{\textit{element\_type}} <: \text{NUMERIC}, \\ \textbf{\textit{format}} <: \text{MATRIXFORMAT}, \\ \textbf{\textit{format-value\_type}} == \textbf{\textit{element\_type}}, \\ \textbf{\textit{virtual\_platform}} <: \text{CLUSTER}, \\ \textbf{\textit{virtual\_platform-node\_type}} <: \text{MULTIPROCESSORNODE} \end{bmatrix}$

(b) $\begin{bmatrix} \textbf{\textit{name}} :> \text{MATRIXFORMAT}, \\ \textbf{\textit{value\_type}} <: \text{NUMERIC} \end{bmatrix}$ (c) $\begin{bmatrix} \textbf{\textit{name}} :> \text{CLUSTER}, \\ \textbf{\textit{node}} :> \text{NODE}, \\ \textbf{\textit{nodes\_count}} :> 1{:}\text{INT} \uparrow, \\ \textbf{\textit{network}} :> \text{NETWORK}, \\ \textbf{\textit{node-network}} == \textbf{\textit{network}} \end{bmatrix}$

(d) $\begin{bmatrix} \textbf{\textit{name}} :> \text{MULTIPROCESSORNODE}, \\ \textbf{\textit{processor\_model}} :> \text{PROCESSORMODEL}, \\ \textbf{\textit{cores\_per\_processor\_count}} :> 1{:}\text{INT} \uparrow, \\ \textbf{\textit{processors\_count}} :> 1{:}\text{INT} \uparrow, \\ \textbf{\textit{memory\_size}} :> 0{:}\text{INT} \uparrow, \\ \textbf{\textit{memory\_type}} :> \text{MEMORYTYPE}, \\ \textbf{\textit{network}} :> \text{NETWORK}, \\ \textbf{\textit{accelerator}} :> \text{GPUACCELERATOR} \end{bmatrix}$

(e) $\begin{bmatrix} \textbf{\textit{name}} :> \text{NETWORK}, \\ \textbf{\textit{startup\_time}} :> \text{INT} \downarrow, \textit{(ns)} \\ \textbf{\textit{node\_latency}} :> \text{INT} \downarrow, \textit{(ns)} \\ \textbf{\textit{bandwidth}} :> 0{:}\text{INT} \uparrow \textit{(Mb/s)} \end{bmatrix}$ (f) $\begin{bmatrix} \textbf{\textit{name}} :> \text{GPUACCELERATOR}, \\ \textbf{\textit{compute\_capability}} :> 10 : \text{INT} \uparrow, \textit{(v is v/10)} \\ \textbf{\textit{architecture}} :> \text{GPUARCHITECTURE} \end{bmatrix}$

Fonte: Próprio Autor

LINEARSYSTEMSOLVER na descrição de contratos:

- ***virtual_platform-nodes_count***: quantidade de nós de processamento;
- ***virtual_platform-node***: tipo de nó de processamento;
- ***virtual_platform-node-processor_model***: modelo de processador dos nós de processamento;
- ***virtual_platform-node-processors_count***: quantidade de processadores de um nó de processamento;
- ***virtual_platform-node-cores_per_processor_count***: quantidade de núcleos de cada processador;
- ***virtual_platform-node-memory_size***: quantidade de memória de cada nó de processamento;

- ***virtual_platform-node-memory_type***: tipo da memória empregada em cada nó de processamento (e.g. SDRAM, DDR1, DDR2, DDR3, etc);
- ***virtual_platform-node-network*** e ***virtual_platform.network***: tipo de rede de interconexão entre os nós de processamento, que devem ter o mesmo valor devido a restrição presente na assinatura CLUSTER;
- ***virtual_platform-network-startup_time***: tempo requerido para manipulação de uma mensagem nos nós de origem e destino da comunicação, determinado tanto pelo tipo do nó de processamento do cluster quanto por características da rede de interconexão;
- ***virtual_platform-network-node_latency***: tempo que uma mensagem leva ao sair do nó de origem para chegar ao nó de destino na rede;
- ***virtual_platform-network-bandwidth***: largura de banda da interconexão entre os nós de processamento;
- ***virtual_platform-node-accelerator***: tipo de GPU suportada;.
- ***virtual_platform-node-accelerator-architecture***: arquitetura da GPU;
- ***virtual_platform-node-accelerator-compute_capability***: *compute capability da GPU*.

Esse conjunto de parâmetros de contexto proposto para LINEARSYSTEMSOLVER é propositalmente abrangente a respeito de suposições que possam influenciar a implementação do algoritmo de resolução de sistemas lineares, de forma a demonstrar a expressividade dos contratos contextuais, permitindo que o código de resolução leve em consideração diferentes requisitos da aplicação sobre o componente, notadamente suposições a respeito do tamanho da matriz de coeficientes, a representação numérica utilizada e a estrutura de dados usa virtual da para armazenar a matriz na memória. Além dessas suposições, o componente faz suposições a respeito da plataforma sobre o qual irá executar, incluindo os tipos de processadores e aceleradores empregados em cada nó de processamento.

Desenvolvedores podem oferecer componentes concretos para contratos contextuais de LINEARSYSTEMSOLVER, para diferentes combinações de argumentos de contexto. Por exemplo, um componente concreto poderia implementar o contrato contextual apresentado na Figura 19, supondo que a aplicação deseja resolver sistemas lineares cuja matriz de coeficientes é composta por valores de ponto flutuante de precisão dupla e está representada como um vetor bidimensional, em um cluster cujos nós de processamento são equipados com processadores Intel Xeon da família E7 e GPUs de arquitetura Pascal com suporte a *compute capability* 6.1. Dessa forma, a implementação poderia, com segurança, fazer uso de características especiais

Figura 19 – Contrato contextual de um componente concreto

$$\left[\begin{array}{l} \textbf{\textit{name}} = \text{LINEARSYSTEMSOLVER}, \\ \textbf{\textit{element\_type}} = \text{DOUBLE}, \\ \textbf{\textit{format}} = \text{ARRAY2D}, \\ \textbf{\textit{virtual\_platform-node-processor\_model}} = \text{INTEL\_XEON\_E7}, \\ \textbf{\textit{virtual\_platform-node-accelerator-architecture}} = \text{PASCAL}, \\ \textbf{\textit{virtual\_platform-node-accelerator-compute\_capability}} = 61, \textit{(6.1)} \\ \textbf{\textit{virtual\_platform-node\_type-memory\_size}} <: \textit{mem} \begin{pmatrix} \textbf{\textit{max\_dim\_size\_m}}, \\ \textbf{\textit{max\_dim\_size\_n}}, \\ \textbf{\textit{element\_type}}, \\ \textbf{\textit{format}} \end{pmatrix} \end{array}\right]$$

Fonte: Próprio Autor

de modelos de GPUs que seguem essas especificações, tais como TITAN X, Quadro P6000, Tesla P40, dentre outras[4], a fim de acelerar a computação. Além disso, poderia fazer uso de características específicas comuns a processadores da família Xeon E7 da fabricante Intel.

É importante salientar, em relação ao contrato na Figura 19, que alguns parâmetros de contexto não possuem argumentos associados explicitamente. Isso se trata de um açúcar sintático que facilita a escrita de contratos extensos, com muitos parâmetros de contexto, evitando-se a obrigação de expressar argumentos *default*. Nesses casos, o argumento *default* para o parâmetro é a própria restrição contextual, para os propósitos do algoritmo de resolução de contratos que será apresentado na Seção 4.3. Por exemplo, para o caso dos parâmetros ***max_dim_size_m*** e ***max_dim_size_n***, a ausência de argumentos denota que a implementação do componente não deve fazer suposições acerca das dimensões (tamanho) da matriz de coeficientes. O valor *default*, referente a restrição contextual mínima do domínio INT $\downarrow$ é $+\infty$, de modo que qualquer dimensão $m \times n$ é válida para contexto. Além disso, a implementação não faz qualquer suposição sobre a rede de interconexão empregada no cluster.

Ainda no contrato da Figura 19, o argumento que é aplicado para o parâmetro ***virtual_platform-node_type-memory_size***, que denota a quantidade de memória necessária na plataforma virtual alvo, é calculado através de uma função chamada *mem*, a partir dos valores reais das dimensões da matriz de coeficientes (***max_dim_size_m*** e ***max_dim_size_n***), tipo de dado numérico de seus elementos (***element_type***) e estrutura de dados que a representa (***format***). Cada vez que o algoritmo de resolução afrouxa os requisitos referentes às dimensões das matrizes de coeficientes do componente LINEARSYSTEMSOLVER, diminuindo os valores

[4] <https://en.wikipedia.org/wiki/CUDA>

Figura 20 – Assinatura Contextual do componente SPARSELINEARSYSTEMSOLVER

(a) SPARSELINEARSYSTEMSOLVER* $\begin{bmatrix} \textbf{\textit{sparse\_pattern}} <: \text{SPARSEMATRIXPATTERN}, \\ \textbf{\textit{format}} <: \text{SPARSEMATRIXFORMAT}, \\ \textbf{\textit{format-sparse\_pattern}} == \textbf{\textit{sparse\_pattern}} \end{bmatrix}$

(b) SPARSEMATRIXFORMAT* $\left[\textbf{\textit{sparse\_pattern}} <: \text{SPARSEMATRIXPATTERN}\right]$

Fonte: Próprio Autor

dos argumentos ***max_dim_size_m*** e ***max_dim_size_n***, buscando achar uma implementação compatível, o requisito de memória se torna cada vez menor, de modo que mais plataformas se tornam habilitadas para serem escolhidas.

Poderíamos propor ainda a existência de um componente abstrato mais específico para componentes concretos que implementam a resolução de sistemas lineares esparsos, nomeado SPARSELINEARSYSTEMSOLVER, cuja assinatura é apresentada na Figura 20, tal que o qualificador SPARSELINEARSYSTEMSOLVER é nominalmente declarado como subtipo do qualificador LINEARSYSTEMSOLVER no catálogo de componentes. Dessa forma, o componente abstrato cujo parâmetro ***name*** é associado a SPARSELINEARSYSTEMSOLVER herda todos os parâmetros de contexto do componente abstrato cujo parâmetro ***name*** é LINEARSYSTEMSOLVER, podendo fixar os argumentos para esses parâmetros, estreitar o alcance de suas restrições contextuais ou acrescentar novos parâmetros. No caso particular, o limite do parâmetro ***format*** é estreitado para SPARSEMATRIXFORMAT e é acrescentado um parâmetro de contexto chamado ***sparse_pattern***, o qual especifica o padrão de elementos não-nulos da matriz de coeficientes.

Na Figura 20, note que uma sintaxe alternativa é usada para expressar o nome do componente abstrato, inspirada no HTS (CARVALHO-JUNIOR *et al.*, 2016), onde:

- $\langle name \rangle * [\ldots]$ pode ser usado ao invés de $[\textbf{\textit{name}} <: \langle name \rangle \ldots]$;
- $\langle name \rangle [\ldots]$ pode ser usado ao invés de $[\textbf{\textit{name}} = \langle name \rangle \ldots]$.

Seja MULTIBLOCKDIAGONALPATTERN um subtipo do qualificador SPARSEMATRIXPATTERN que adiciona dois parâmetros de contexto à assinatura contextual de SPARSEMATRIXPATTERN, denominados ***block_size*** e ***diagonal_size***, voltado matrizes de coeficientes esparsas do tipo bloco-diagonal com tamanho de bloco determinado pelo valor do parâmetro ***block_size*** e número de diagonais de blocos determinados pelo valor do parâmetro ***diagonal_size***. A partir do qualificador MULTIBLOCKDIAGONALPATTERN, vários padrões mais específicos comuns de matrizes esparsas podem ser definidos, valorando-se ***block_size*** e ***diagonal_size***,

Tabela 6 – Subtipos para MULTIBLOCKDIAGONALPATTERN

Padrão	***block_size***	***diagonal_size***
DIAGONAL	1	1
TRIDIAGONAL	1	3
BLOCKTRIDIAGONAL	*aberto*	3
PENTADIAGONAL	1	5

Fonte: Próprio Autor

Figura 21 – Contrato contextual de SPARSELINEARSYSTEMSOLVER

$$\left[\begin{array}{l} \textbf{\textit{name}} = \text{SPARSELINEARSYSTEMSOLVER}, \\ \textbf{\textit{format}} = \text{CRSFORMAT}, \\ \textbf{\textit{sparse\_pattern}} = \text{BLOCKTRIDIAGONAL}, \\ \textbf{\textit{sparse\_pattern-block\_size}} = 2, \\ \textbf{\textit{virtual\_platform-node-accelerator-architecture}} = \text{VOLTA} \end{array}\right]$$

Fonte: Próprio Autor

como os apresentados na Tabela 6.

No catálogo de componentes, seriam registrados componentes concretos para contratos contextuais do qualificador MULTIBLOCKDIAGONALPATTERN apenas para as valorações correspondentes à Tabela 6, de modo a restringir as escolhas de argumentos em contratos MULTIBLOCKDIAGONALPATTERN apenas para um subconjunto daquelas consideradas válidas. Portanto, MULTIBLOCKDIAGONALPATTERN é um exemplo de aplicação de dependências funcionais entre qualificadores, mencionadas na Seção 4.1.2.2. O qualificador MATRIXFORMAT é um outro exemplo de qualificador com assinatura não vazia cujos componentes concretos estariam registrados para delimitar os tipos de elementos permitidos para os formatos específicos de matrizes derivados de MATRIXFORMAT. Por sua vez, no caso de SPARSEMATRIXFORMAT, o parâmetro ***sparse_pattern*** pode adicionalmente ser considerado, permitindo delimitar os padrões de matrizes aos quais cada formato orientado.

A Figura 21 apresenta um exemplo de contrato contextual de SPARSELINEARSYSTEMSOLVER, que representa o tipo de componentes que implementam a resolução de sistemas lineares esparsos onde a matriz de coeficientes é esparsa com elementos não-nulos acumulados nas três diagonais centrais de blocos de tamanho 2 e é armazenada segundo a estrutura de dados conhecida com CRS (*Compressed Row Storage*). Além disso, o componente assume o acesso, a partir dos nós de processamento da plataforma virtual hospedeira, de aceleradores GPU de arquitetura Volta.

Figura 22 – Contrato contextual do componente a ser resolvido pelo Alite

$$\begin{bmatrix} \textbf{\textit{name}} = \text{SPARSELINEARSYSTEMSOLVER}, \\ \textbf{\textit{max\_dim\_size\_m}} = 10^6, \\ \textbf{\textit{max\_dim\_size\_n}} = 10^6, \\ \textbf{\textit{matrix\_pattern}} = \text{TRIDIAGONAL}, \\ \textbf{\textit{virtual\_platform-node-accelerator-architecture}} = \text{PASCAL} \end{bmatrix}$$

Fonte: Próprio Autor

Em um sistema de computação paralela, suponha que, através do contrato contextual apresentado na Figura 22, um componente de resolução de sistemas lineares seja requisitado para resolver um sistema linear esparso com ordem de grandeza $10^6 \times 10^6$ cujos elementos não-nulos estão concentradas nas três diagonais centrais, utilizando uma GPU que suporte a arquitetura Pascal. Para satisfazer esse sistema, o algoritmo de resolução do Alite buscará um componente concreto que implemente um contrato contextual compatível com o contrato submetido pela aplicação. Para esse exemplo, poderia ser selecionado o componente concreto que implementa o contrato da Figura 21. Porém, note que, em alguns parâmetros, os argumentos associados são diferentes. Nesses casos, é usada a relação de subtipos para calcular a compatibilidade. Por exemplo, a restrição quanto ao padrão de elementos não-nulos da matriz esparsa pode ser afrouxada, generalizando-se o valor do parâmetro ***sparse_pattern***, que é requerido, de TRIDIAGONAL para BLOCKTRIDIAGONAL[***block_size***=1,***diagonal_size***=1], uma vez que matrizes tridiagonais são um caso especial de matrizes bloco-tridiagonais onde tamanho do bloco é unitário. Logo, um componente que resolve sistemas lineares esparsos cuja matriz de coeficientes segue o padrão bloco-tridiagonal para qualquer tamanho de bloco também irá resolver um sistema com matriz esparsa tridiagonal, onde o tamanho do bloco é 1. Um outro caso é o valor dos parâmetros ***max_dim_size_m*** e ***max_dim_size_n***. Caso não seja encontrado algum componente específico para matrizes de dimensões no máximo 10^6, será buscado um componente que opere sobre matrizes de tamanho maior, até que seja atingido o valor limite $+\infty$, caso não exista algum componente que opere sobre uma dimensão específica. Finalmente, VOLTA é um subtipo de PASCAL, pois a arquitetura Volta de GPUs NVIDIA é uma extensão sobre a arquitetura Pascal.

Para um contrato requisitado ao Alite, a interpretação de um parâmetro não suprido, ao qual não foi associado um argumento, tais como os parâmetros referentes ao formato das matrizes (de fato, a aplicação não é indiferente quanto ao formato usado para representar as matrizes), é diferente da interpretação dada em um contrato implementado por um componente

concreto. Nesse caso, diz-se que o parâmetro encontra-se livre, ou seja, o algoritmo de resolução não irá fazer suposições a cerca do valor desse parâmetro, sendo a escolha indiferente a eles. Por esse motivo, os valores para o parâmetro ***format*** do componente concreto da Figura 21 são compatíveis com a ausência de uma valoração para esse parâmetro no contrato contextual requisitado ao Alite da Figura 22.

Considere agora o seguinte contrato contextual:

SPARSELINEARSYSTEMSOLVER [***element_type*** = DOUBLE,***format*** = ARRAY2D]

Nesse contrato, é requisitado um componente que realize a resolução de um sistema linear esparso de números de ponto flutuante de precisão dupla representados como arrays bidimensionais. Uma vez que ***virtual_platform-node-processor_model*** é um parâmetro livre, não recebendo argumento explicitamente, diz-se que o contrato não faz suposição sobre o tipo de processador da plataforma virtual alvo. De fato, não faz suposição qualquer parâmetro referente à plataforma virtual (***virtual_platform-****). Segundo o algoritmo de resolução de contratos, o componente concreto que implementa o contrato contextual da Figura 19, o qual requere que a plataforma virtual possua nós com processador Xeon da família E7, é compatível com esse contrato, sendo um dos candidatos no processo de escolha.

Uma vez que a resolução de contratos de um componente de computação é feita em conjunto com a resolução do contrato da plataforma virtual sobre o qual vai executar, há uma tendência maior a ser escolhido um componente melhor adequado para explorar as características peculiares da plataforma.

4.2.1 Inserindo Suposições sobre Qualidade e Custo

A assinatura do componente abstrato LINEARSYSTEMSOLVER suporta diversas suposições a respeito de características arquiteturais da plataforma virtual alvo relevantes para construção de modelos de predição de desempenho para componentes concretos implementados a partir desse componente abstrato. Em particular, para esses propósitos, são conhecidos o tipo da interconexão entre os nós de processamento, o modelo dos processadores utilizados nos nós de processamento (e suas quantidades), o tipo da memória empregada nos nós e os modelos dos aceleradores. Desse modo, é possível saber, com certa precisão, o desempenho da troca de mensagem entre os nós, os níveis da hierarquia de memória (*cache* e memória principal) e o

Figura 23 – Extensão para a Assinatura Contextual de LINEARSYSTEMSOLVER

$$\left[\begin{array}{l} \vdots \\ \textbf{\textit{execution\_time}} :> \text{INT} \downarrow, \\ \textbf{\textit{efficiency}} :> 0 : \text{INT} \uparrow, \\ \textbf{\textit{estimated\_energy}} :> \text{INT} \downarrow, \\ \textbf{\textit{reputation\_score}} :> 0{:}\text{INT} \uparrow, \\ \textbf{\textit{price}} :> 0{:}\text{INT} \downarrow \end{array}\right]$$

Fonte: Próprio Autor

desempenho de acesso em cada um desses níveis, bem como a velocidade de processamento em operações de aritmética de ponto flutuante (precisão simples e dupla) e inteira.

Os parâmetros de qualidade e custo a seguir, incluídos na assinatura contextual do componente de resolução de sistemas lineares (Figura 18) conforme o fragmento de assinatura contextual da Figura 23, poderiam também ser incluídos:

- ***execution_time***: tempo de execução estimado, o qual pode ser calculado através de uma função derivada de uma modelagem analítica, aplicada sobre os valores atuais dos parâmetros de *aplicação*, tais como os que especificam a dimensão das matrizes de entrada, e de *plataforma*, tais como os que especificam número de nós, processadores e núcleos de processamento, bem como o desempenho da interface de comunicação e do sistema de memória;
- ***efficiency***: fração estimada do tempo de execução no qual o componente executa computação útil, ou seja, desprezando-se sobrecargas do paralelismo, tais como ociosidade de processos, comunicação e computação adicional;
- ***estimated_energy***: estimativa da quantidade de energia consumida na execução do componente sobre a plataforma virtual alvo;
- ***reputation_score***: escore de reputação atribuído pela plataforma (e.g. escala de 1 a 10) em função da sua confiabilidade no atendimento das estimativas oferecidas pelo componente quanto aos seus parâmetros de qualidade e de custo;
- ***price***: custo monetário estimado do uso do componente, o qual pode ser calculado de várias formas, dependendo do interesse do provedor do componente e possivelmente levando em consideração o seu escore de reputação, tais como:
  - em função do tempo desde a instanciação (ativação da ação instantiate) até a destruição do componente (execução da ação release);

Figura 24 – Contrato Contextual de LINEARSYSTEMSOLVER

$$\begin{bmatrix} \textbf{\textit{name}} = \text{LINEARSYSTEMSOLVER}, \\ \textbf{\textit{max\_dim\_size\_m}} = 10^3, \\ \textbf{\textit{max\_dim\_size\_n}} = 10^3, \\ \textbf{\textit{element\_type}} = \text{FLOAT} \\ \textbf{\textit{format}} = \text{ARRAY2D}, \\ \textbf{\textit{execution\_time}} = 150\text{s}, \\ \textbf{\textit{reputation}} = 9, \\ \textbf{\textit{efficiency}} = 0.7, \\ \textbf{\textit{cost}} = 10 \end{bmatrix}$$

Fonte: Próprio Autor

- em função do tempo executando computações úteis (eficiência);
- em função do número de operações de leitura/escrita de memória e de ponto flutuante executadas pelo componente.

Um componente concreto que implemente o contrato contextual da Figura 19[5] pode fornecer funções derivadas de modelos analíticos que permitam calcular os valores desses parâmetros de qualidade e de custo. Durante a execução do procedimento de resolução de contratos, aplicado a um componente de sistema, essas funções podem determinar argumentos para tais parâmetros, os quais devem ser comparados, em relação à compatibilidade, com os valores solicitados por uma aplicação através do contrato que está sendo resolvido. Por exemplo, uma aplicação poderia demandar por um componente de resolução de sistemas lineares que atenda ao contrato da Figura 24.

É então responsabilidade do sistema de resolução de contratos, em sua fase de *seleção*, encontrar uma ou mais plataformas virtuais cujas características arquiteturais levem ao cálculo de argumentos compatíveis para esses parâmetros de qualidade e custo.

Fazendo a combinação de todos os componentes de computação que implementam LINEARSYSTEMSOLVER para matrizes de coeficientes de dimensões $10^3 \times 10^3$ com valores ponto flutuante de precisão simples armazenados em um vetor bidimensional com plataformas virtuais que atendam aos requisitos impostos pelos argumentos de qualidade e custo informados no contrato, é formada uma lista de pares que representam *componentes sistema candidatos*. A partir dessa lista, na fase de *classificação*, é escolhido o componente melhor classificado,

[5] Resolução de sistemas lineares com a matriz de coeficientes formada por números de ponto flutuante de precisão dupla armazenados em um vetor bidimensionais, sobre um cluster com nós equipados com processadores Xeon da família E7 e aceleradores Pascal programados com CUDA 6.1

segundo critérios da nuvem.

Uma descrição formal do algoritmo de resolução de contratos para componentes sistema é apresenta na próxima seção. Outros estudos de caso seguem no Capítulo 5.

4.3 Resolução de Contratos Contextuais

A resolução de um contrato contextual de um componente sistema (componente de computação + plataforma virtual) em um sistema de computação paralela criado por uma aplicação da HPC Shelf inicia-se através de uma requisição ao serviço *Core*. Como mencionado no final da seção anterior, o processo é definido em duas etapas:

1. **seleção**, na qual é gerada uma lista de componentes sistema compatíveis com o contrato contextual em questão (critério de *segurança*, visando escolha de componentes compatíveis, ou seja, que satisfaçam as imposições expressas no contrato);
2. **classificação**, onde é feito o ranqueamento dos componentes sistema compatíveis com o contrato com base nos valores associados aos parâmetros de qualidade e custo (critério de *qualidade*, visando escolha do melhor componente compatível segundo as políticas de alocação de recursos da plataforma).

A seguir, nas seções 4.3.1 e 4.3.2, são detalhadas as abstrações e os algoritmos envolvidos nas etapas de seleção e classificação, respectivamente.

4.3.1 Seleção de Componentes

No Capítulo 3, foram introduzidos os principais elementos que constituem a arquitetura de sistemas de computação paralela da HPC Shelf. Esses são basicamente constituídos a partir da composição de componentes de computação e fontes de dados. Por intermédio desses últimos, podem ser acessadas extensas massas de dados tratadas pelos primeiros. Computações e fontes de dados são ligados entre si através de conectores e ligações (de ambiente) indiretas. Por sua vez, conectores são conectados a componentes de computação e fontes de dados através de ligações diretas.

Fontes de dados são componentes estritamente acoplados a uma determinada infraestrutura de computação onde os dados encontram-se fisicamente armazenados, sendo sua responsabilidade oferecer acesso a esses dados através de um serviço exposto por intermédio de uma *ligação* de ambiente (indireto). Por esse motivo, a plataforma virtual sobre a qual um

componente da espécie fontes de dados encontra-se instanciado é, por definição, transparente para a aplicação.

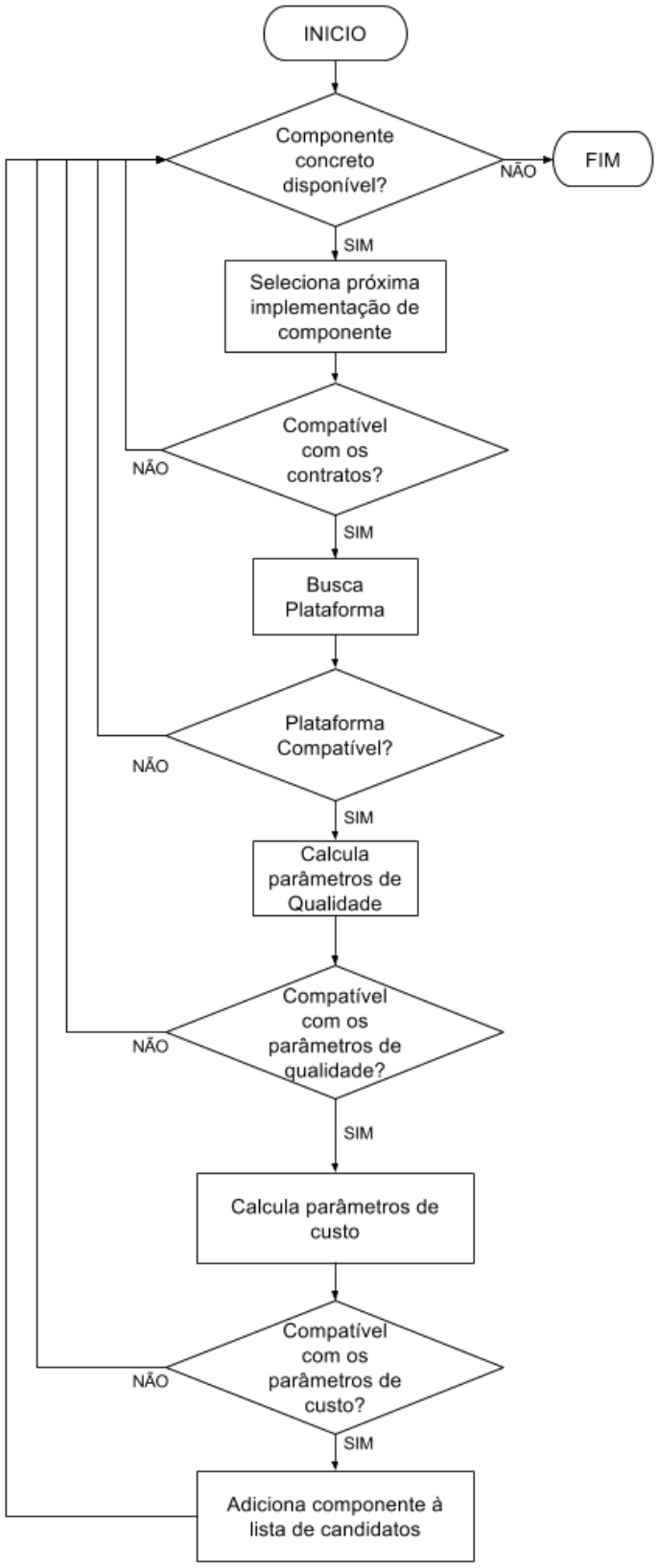
Por outro lado, componentes de computação podem ter implementações para diferentes plataformas virtuais, de diferentes mantenedores e apresentando variações em seus contratos contextuais que definem suas propriedades arquiteturais particulares. O contrato contextual da plataforma virtual onde de fato um componente de computação será instanciado é definido na especificação do sistema de computação paralela e resolvido em tempo de execução. A fim de lidar com essa relação íntima entre um componente de computação e a plataforma virtual a ela associada, onde os valores atuais de parâmetros de qualidade e custo de um pode ser usados para calcular valores de parâmetros de qualidade e custo do outro, foi introduzida a abstração de componente sistema. Assim, o processo de seleção de componentes é aplicado a cada par formado por um componente de computação e uma plataforma virtual, ao invés de individualmente sobre cada um deles.

Com relação aos conectores e ligações, são selecionados de forma a que suas facetas possam ser instanciadas sobre as plataformas virtuais onde encontram-se alocados os componentes de computação e fontes de dados a eles ligados, de modo que a seleção de componentes sistema e fontes de dados precede a seleção dos conectores de um sistema de computação paralela. Por esse motivo, descrevemos nesta seção o processo de seleção para componentes sistema, sendo a seleção de fontes de dados, conectores e ligações instâncias mais simples do processo.

O processo de seleção de componentes de Alite é automático, implementado pelo *Core*. Nesse processo, o catálogo de componentes é percorrido de modo a encontrar componentes sistema compatíveis com os contratos contextuais de componentes de computação e de suas plataforma virtuais associadas. A Figura 25 apresenta as etapas segundo as quais o processo de seleção de componentes é realizado, a saber:

1. Sejam $\mathscr{A}$ e $\mathscr{P}$ contratos contextuais do componente de computação e da plataforma virtual associada como plataforma alvo do componente de computação;
2. Para o contrato $\mathscr{A}$, o *Core* aplica uma extensão do *algoritmo de resolução de contratos contextuais* de Carvalho-Junior *et al.* (2016), descrito na Seção 4.3.1.1, para encontrar todos os componentes de computação que satisfazem $\mathscr{A}$. Chamaremos esses componentes de $\mathscr{A}_1, \mathscr{A}_2, \ldots$ e $\mathscr{A}_m$, onde m é o número de contratos contextuais que satisfazem $\mathscr{A}$, restrito aos parâmetros de aplicação e de plataforma.

Figura 25 – Fluxo da Geração da Lista de Componentes



Fonte: Próprio Autor

3. Por definição das espécies de componentes da HPC Shelf, cada componente de computação $\mathscr{A}_i^{\ i\in\{1,\ldots,m\}}$ possui uma dependência (componente aninhado) para um componente da espécie plataforma, que especifica através de um contrato contextual, a plataforma de computação paralela capaz de executá-lo. Chamaremos esses contratos de plataformas de $\mathscr{P}(\mathscr{A}_i)$.
4. Para o contrato $\mathscr{P}$, o algoritmo de resolução de contratos contextuais escolhe um conjunto de componentes da espécie plataforma $\mathscr{P}_j$ que os satisfaz, tal que $j \in \{1,\ldots,n\}$, onde n é a quantidade de plataformas que satisfazem $\mathscr{P}$;
5. Faz-se um produto cartesiano com os m contratos de $\mathscr{A}$ e os n contratos de $\mathscr{P}$. Em posse da lista de pares gerada no produto cartesiano, verifica-se em cada par se $\mathscr{P}_j <: P(\mathscr{A}_i)$. Todos pares que a relação de subtipo for verdadeira serão adicionados à lista de sistemas computacionais candidatos, enquanto os demais serão descartados.
6. Tem-se agora um conjunto de pares $\langle \mathscr{A}_i, \mathscr{P}_j \rangle$, os quais representam componentes sistema que atendem aos argumentos dos parâmetros de contexto de aplicação $\mathscr{A}$, os argumentos dos parâmetros de plataforma de $P(\mathscr{A}_i)$ e os argumentos dos parâmetros da plataforma do contrato $\mathscr{P}$, sendo agora necessário filtrar aqueles que satisfazem os argumentos dos parâmetros de qualidade e de custo. Uma característica importante desses últimos argumentos é que podem ser calculados a partir da análise dos demais parâmetros.
7. Assim, em um par $\langle \mathscr{A}_i, \mathscr{P}_j \rangle$, os argumentos dos parâmetros de plataforma de $\mathscr{P}_j$ são aplicados aos parâmetros de plataforma de $\mathscr{A}_i$, de tal forma que torna-se possível calcular os argumentos dos parâmetros de qualidade de cada componente $\mathscr{A}_i$ no par. Então, todos aqueles pares nos quais pelo menos um dos argumentos dos parâmetros de qualidade de cada componente $\mathscr{A}_i$ não satisfaz o respectivo argumento do contrato $\mathscr{A}$, segundo a relação de subtipos do sistema, são descartados.
8. No mesmo par $\langle \mathscr{A}_i, P_j \rangle$, os argumentos dos parâmetros de qualidade calculados para $\mathscr{A}_i$ determinam os argumentos dos parâmetros de qualidade de P_j, de tal forma que é possível calcular os argumentos dos parâmetros de custo de cada par $\langle \mathscr{A}_i, P_j \rangle$, que por sua vez determinam os argumentos dos parâmetros de custo de $\mathscr{A}_i$. Mais uma vez, todos aqueles pares nos quais pelo menos um dos argumentos dos parâmetros de custo de cada componente $\mathscr{A}_i$ não satisfaz o respectivo argumento do contrato $\mathscr{A}$, segundo a relação de subtipos do sistema, são descartados.
9. Os pares $\langle \mathscr{A}_i, \mathscr{P}_j \rangle$ remanescentes representam componentes sistema compatíveis com os

contratos $\mathscr{A}$ e $\mathscr{P}$, os quais serão submetidos para a fase de classificação.

4.3.1.1 Algoritmo de Resolução de Contratos Contextuais

O algoritmo de resolução de Carvalho-Junior *et al.* (2016), necessário para a primeira etapa do algoritmo de seleção de componentes, busca componentes cujos contratos contextuais registrados no *Core* casem com contratos contextuais fornecidos pelo SAFe, em relação apenas aos parâmetros de aplicação e de plataforma. De fato, o algoritmo busca todos os subtipos do contrato contextual para os quais há componentes registrados no *Core.* Caso nenhum seja encontrado, uma exceção é lançada, de modo que a aplicação não pode continuar até que um componente um contrato compatível seja registrado por algum desenvolvedor. A versão do algoritmo apresentada a seguir é um extensão que leva em consideração a covariância e a contravariância de parâmetros de contexto.

Usando a sintaxe da Figura 16, seja

$$\mathrm{C} \equiv \left[\boldsymbol{par}_1 \sqsupset_1 \mathrm{B}_1, \boldsymbol{par}_2 \sqsupset_2 \mathrm{B}_2, \cdots, \boldsymbol{par}_p \sqsupset_p \mathrm{B}_p\right]$$

a assinatura do componente abstrato C, onde $\{\boldsymbol{par}_1, \boldsymbol{par}_2, \ldots, \boldsymbol{par}_p\}$ são os parâmetros de contexto de C, $\{\mathrm{B}_1, \mathrm{B}_2, \ldots, \mathrm{B}_p\}$ são suas restrições contextuais, e $\{\sqsupset_1, \sqsupset_2, \ldots, \sqsupset_p\}$ abstraem o tipo de parâmetro, podendo ser "$<:$" (requerido) ou "$:>$" (ofertado).

Finalmente, seja $\mathscr{T}$ um contrato contextual para o componente abstrato C com os seguintes argumentos de contexto:

$$\left[\boldsymbol{par}_{i_1} = \mathrm{A}_1, \boldsymbol{par}_{i_2} = \mathrm{A}_2, \cdots, \boldsymbol{par}_{i_q} = \mathrm{A}_q\right]$$

, onde $\{\boldsymbol{par}_{i_1}, \boldsymbol{par}_{i_2}, \cdots, \boldsymbol{par}_{i_q}\} \subseteq \{\boldsymbol{par}_1, \boldsymbol{par}_2, \cdots, \boldsymbol{par}_p\}$ e $\mathrm{A}_j \sqsupset_j \mathrm{B}_{i_j}$, para $j \in \{1, 2, \ldots, q\}$, o contrato contextual de um argumento de contexto arbitrário ι que contém q_i parâmetros de contexto. O algoritmo de resolução que buscará por todos componentes concretos que implementam contratos contextuais de C compatíveis com o contrato $\mathscr{T}$ (subtipos de $\mathscr{T}$), pode ser formalmente descrito pelos seguintes passos:

1. Sejam $\mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_q$ os componentes abstratos de $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_q$, respectivamente;
2. Sejam $\mathrm{C}_1^{\top}, \mathrm{C}_2^{\top}, \ldots, \mathrm{C}_q^{\top}$ os componentes abstratos de $\mathrm{B}_{i_1}, \mathrm{B}_{i_2}, \ldots, \mathrm{B}_{i_q}$, respectivamente;
3. Para cada $j \in \{1, 2, \ldots, q\}$, calcule a sequência $\{\mathrm{C}_j^0, \mathrm{C}_j^1, \ldots, \mathrm{C}_j^{k_j}\}$, para os casos abaixo:
    a) se $\boldsymbol{par}_{i_j}$ é um parâmetro ofertado (covariante), a sequência é calculada pela travessia em largura da árvore de subtipos de C_i restrita àqueles que aparecem no catálogo de componentes como valorações para $\boldsymbol{par}_{i_j}$ em contratos de C;

b) se $\boldsymbol{par}_{i_j}$ é um parâmetro requerido (contravariante), a sequência inclui todos os super-tipos de C_i que aparecem no catálogo de componentes como valorações para $\boldsymbol{par}_{i_j}$ em contratos de C que não sejam super-tipos de $\mathrm{C}_j^{\top}$, de modo que:
   - $\mathrm{C}_j^0 \equiv \mathrm{C}_j$;
   - C_j^i é super-tipo de C_j^{i-1}, para $i \in \{1,2,\ldots,k_j\}$; e
   - $\mathrm{C}_j^{k_j} \equiv \mathrm{C}_j^{\top}$;

4. Seja $\mathscr{R}$ uma *árvore de resolução*, construída da seguinte forma:
   a) A raiz é rotulada com C_0^0;
   b) Um nó rotulado C_j^k possui filhos com rótulos C_{j+1}^0, C_{j+1}^1, ..., $\mathrm{C}_{j+1}^{k_{j+1}}$.
5. Realize uma travessia em profundidade, em pré-ordem, de $\mathscr{R}$. A cada visita a um nó folha, a sequência $\{\mathrm{C}_1^{j_1}, \mathrm{C}_2^{j_2}, \ldots, \mathrm{C}_q^{j_q}\}$ inclui os rótulos dos nós no caminho do nó seguinte à raiz até a folha. Por construção, constituem argumentos de contexto para $\boldsymbol{par}_{i_1}, \boldsymbol{par}_{i_2}, \ldots, \boldsymbol{par}_{i_q}$, respectivamente, os quais são parâmetros não-livres de $\mathscr{T}$. Então, faça as seguintes operações:
   a) Construa um contrato contextual $\mathscr{T}'$ a partir de $\mathscr{T}$ da seguinte forma:
      i. defina os contratos contextuais $\mathrm{A}'_1, \mathrm{A}'_2, \ldots, \mathrm{A}'_q$ pela substituição de $\mathrm{C}_1, \mathrm{C}_2, \ldots, \mathrm{C}_q$ por $\mathrm{C}_1^{j_1}, \mathrm{C}_2^{j_2}, \ldots, \mathrm{C}_q^{j_q}$ em $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_q$, respectivamente;
      ii. substitua $\mathrm{A}_1, \mathrm{A}_2, \ldots, \mathrm{A}_q$ por $\mathrm{A}'_1, \mathrm{A}'_2, \ldots, \mathrm{A}'_q$ em $\mathscr{T}$, de tal forma que as ocorrências dos parâmetros de contexto par_{i_1}, par_{i_2}, ..., par_{i_q} da assinatura de C possuem $\mathrm{A}'_1, \mathrm{A}'_2, \ldots, \mathrm{A}'_q$ como argumentos em $\mathscr{T}'$;
   b) Busque na biblioteca de componentes um componente concreto cujo contrato contextual case com $\mathscr{T}'$ nos seus argumentos de contexto.
   c) Se um componente é encontrado, adicione na lista de candidatos, continue para buscando os componentes até que alcance o limite, e por fim retorne os componentes encontrados para a escolha das plataformas;
6. Se todos os nós foram visitados e nenhum componente concreto foi encontrado, uma exceção é lançada informando que não existe na biblioteca um componente compatível com o tipo de instanciação de ι.

4.3.2 Classificação dos Resultados

A *etapa de classificação* consiste no ranqueamento dos componentes sistemas encontrados na *etapa de seleção* com auxílio do Alite. Essa classificação é necessária para que

as escolhas de componentes que satisfaçam a uma política de alocação de recursos imposta pela HPC Shelf. Dentre essas políticas, pode-se buscar, por exemplo, obter maior eficiência energética, menor custo financeiro para as aplicações, menor tempo de execução, dentre outros critérios. A escolha de cada critério para um sistema computacional pode ser um processo complexo, envolvendo parâmetros de qualidade de serviço e parâmetros de custo do componente. Tais políticas, que afetam o ranqueamento, são pré-definidas pela HPC Shelf. No entanto, podem evoluir em atualizações de versão do *Core*.

Neste capítulo, apresentamos um *framework* de classificação, ao invés de um algoritmo ou estratégia de classificação propriamente dito, uma vez que a definição de uma política de alocação de recursos global é propriedade específica de uma implementação da HPC Shelf. No Capítulo 5, uma estratégia particular de classificação será apresentada, denotando uma política de alocação, como prova de conceito e estudo de caso. Portanto, é importante ressaltar que a apresentação de contribuições relativas à definição de políticas de alocação de recursos sobre plataformas de serviços HPC em nuvem, usando a HPC Shelf como base, não está dentre os objetivos desta Tese.

4.3.2.1 Parâmetros de Ranqueamento

O *framework* de classificação proposto por Alite introduz um novo tipo de parâmetro de contexto, doravante chamados de *parâmetros de ranqueamento*. Parâmetros de ranqueamento são pré-definidos pela plataforma (e.g. HPC Shelf) e calculados automaticamente segundo uma função de ranqueamento aplicada sobre os valores de parâmetros de qualidade e custo, sem intervenção do desenvolvedor de componentes ou provedor de aplicações. Cada parâmetro de ranqueamento define uma política de alocação de recursos (componentes) suportada pela plataforma. De fato, para cada parâmetro de ranqueamento de cada componente selecionado, a função de ranqueamento associa um valor numérico que define uma ordem total entre os componentes selecionados (classificação) segundo a política denotada pelo parâmetro.

É responsabilidade do provedor de aplicações especificar como será realizada a escolha, dinamicamente, de qual parâmetro de ranqueamento será aplicado a cada componente de um sistema de computação paralela, de acordo com os requisitos da aplicação em si. Portanto, as alternativas de parâmetros de ranqueamento adotadas pela plataforma (e.g. HPC Shelf) é uma informação pública para provedores de aplicações, muito embora não precisem fazer referência a valorações para esses parâmetros em contratos contextuais, uma vez que são automaticamente

calculados da forma anteriormente descrita.

4.3.2.2 Parâmetros Ascendentes e Descendentes (Qualidade e Custo)

Uma vez definido o *framework* de classificação do Alite, através da noção de parâmetro de contexto de ranqueamento, podemos explicar melhor os princípios por trás da definição da relação de subtipos entre valores de parâmetros de qualidade e de custo.

Parâmetros de qualidade e custo podem ser de dois tipos, *ascendentes* ou *descendentes*. No caso de um parâmetro de contexto ascendente, quanto maior o valor a ele associado (argumento), maior tende a ser o valor de qualquer parâmetro de ranqueamento que dele depende. Por sua vez, para parâmetros descendentes, um valor maior tende a tornar menor o valor de ranqueamento. Posto através de exemplos, a eficiência mínima garantida pela implementação de um algoritmo paralelo por parte de um componente de computação é um parâmetro de qualidade ascendente, pois quanto maior o valor da eficiência mínima garantida, melhor será a classificação de um componente de computação segundo o ranqueamento. Por outro lado, um parâmetro de que meça o custo monetário de uso de um componente é descendente, de modo que quanto mais custoso um componente, pior tende a ser a sua classificação.

O caráter ascendente ou descendente de um parâmetro de contexto é portanto consequência de sua semântica. Sua importância é ajudar na definição da relação de subtipos entre valores de parâmetros de qualidade e custo, de modo que um parâmetro ascendente deve ter valor numérico INT $\uparrow$ ou REAL $\uparrow$, enquanto um parâmetro descendente deve ter um valor numérico INT $\downarrow$ ou REAL $\downarrow$. Dessa forma, um valor maior para o parâmetro eficiência, do tipo REAL $\uparrow$, usado no exemplo anterior, é um subtipo de um valor menor. Desse modo, sendo um parâmetro ofertado, em um contexto no qual é solicitado um componente com eficiência 70%, um componente com eficiência 80% pode ser selecionado. Analogamente, para o parâmetro de custo monetário, também ofertado, do tipo REAL $\downarrow$, um componente com menor custo pode ser escolhido em um contexto onde um componente de maior custo é requisitado.

Vale a pena enfatizar que essa relação ascendente e descendente não pode ser definida para parâmetros de aplicação e plataforma, que por esse motivo não são usados para calcular o valor de parâmetros de ranqueamento. Por exemplo, a relação entre o número de processadores aplicado a um componente de computação pode indiretamente influenciar positivamente ou negativamente no seu ranqueamento, em função de suas propriedades de escalabilidade.

4.4 Considerações Finais

O modelo de descoberta de componentes da HPC Shelf, representada pela estratégia de resolução de contratos contextuais, permitiu a escolha de componentes que representam sistemas de computação paralela, chamados de componentes sistema, para suprir as necessidades das aplicações na busca por soluções computacionais para problemas descritos por usuários especialistas. A escolha de componentes sistema diferencia-se das estratégias de escolha comumente adotadas para recursos computacionais em nuvens devido ao fato de que componentes sistema são representados pelo componente de *software* que descreve uma computação, ou estrutura de dados, paralela e pela plataforma de computação paralela, representada como um componente da espécie *plataforma* a qual denominamos plataforma virtual. A escolha se dá pelo casamento de parâmetros de contexto que descrevem propriedades do ambiente de execução do componente, representado pela aplicação que o requisita e pela plataforma (virtual) que ofertará o serviço de execução ao componente, bem como por parâmetros de contexto que descrevem parâmetros de qualidade e de custo, mensuráveis. O mecanismo proposto possibilita flexibilidade para que a HPC Shelf possa definir diferentes critérios de escolha de componentes, controlando políticas de alocação de recursos da nuvem, porém oferecendo a possibilidade de conciliar seus interesses com os interesses das aplicações.

Neste capítulo, foram apresentados os detalhes arquiteturais do *Core*, o qual é responsável pelo armazenamento e alocação dos componentes. O processo de seleção de componentes concretos e plataformas é feito através do Alite, o qual tem em seu resultado, uma lista de sistemas computacionais ranqueados segundo os critérios da HPC Shelf. Espera-se que, com as abordagens escolhidas, sejam satisfeitos os requisitos dos usuários de HPC, aproximando estes usuários aos ambientes de nuvens computacionais.

5 ESTUDOS DE CASO E AVALIAÇÃO DO ALITE

Um protótipo de implementação do Alite foi desenvolvido com propósito de avaliar o sistema de contratos contextuais com respeito a aspectos de expressividade, desempenho e eficácia, evidenciando a sua aplicabilidade em cenários reais. Os estudos de caso e experimentos conduzidos para os fins dessa avaliação são o objeto desta seção.

Com relação à expressividade, é apresentado, na Seção 5.1 o projeto de um arcabouço de plataformas virtuais para a HPC Shelf, onde as plataformas virtuais constituem plataformas de *cluster computing* com algumas características consideradas comuns no projeto moderno de *clusters*. A partir desse arcabouço, por mecanismos de derivação próprios do Alite, é apresentada um arcabouço de plataformas virtuais sobre o serviço EC2 da Amazon na Seção 5.2, onde os nós de processamento das plataformas virtuais são instâncias de máquinas virtuais de modelos suportados pelo serviço. Os princípios que norteiam o projeto desse arcabouço podem ser utilizados para construção de arcabouço baseados em virtualmente qualquer outros servico IaaS, tais como Google Cloud Platform e Microsoft Azure, os quais, juntos com o Amazon EC2, constituem os maiores provedores IaaS da atualidade. Finalmente, foi também desenvolvido um arcabouço de componentes de computação para multiplicação de matrizes com base na interface de nível 3 do BLAS (*Basic Linear Algebra Subprograms*) (DONGARRA, 2002), apresentado na Seção 5.3, demonstrando como funcionalidades de bibliotecas científicas de uso disseminado podem ser encapsuladas e reutilizadas por meio de componentes na HPC Shelf.

Com relação ao desempenho, os resultados de um experimento para aferir o tempo de resolução de contratos contextuais são apresentados na Seção 5.4. Para esse experimento, foram utilizados contratos provenientes do estudo de caso com componentes de multiplicação de matrizes baseada em BLAS apresentados na Seção 5.3, bem como cargas de plataformas virtuais hipotéticas que buscam imitar cenários reais. Embora hipotéticos, os perfis de plataformas de computação paralela utilizados são baseados em *logs* de execução de programas paralelos sobre o *cluster* principal do CENAPAD-UFC, o centro de computação de alto desempenho da Universidade Federal do Ceará, utilizando o arcabouço de plataformas virtuais descrito na Seção 5.1.

Finalmente, com relação à eficácia, um experimento foi conduzido para aferir a qualidade de um mecanismo de classificação da lista de sistemas candidatos baseado em técnicas MCDM (*Multi-Criteria Decision Making*[1]), cujos resultados são apresentados na Seção 5.4. Para

[1] Tomada de Decisão Baseada em Múltiplos Critérios

isso, os resultados obtidos com diferentes técnicas MCMD são comparadas não apenas entre si, mas também com classificações realizadas por especialistas humanos. Além da avaliação do mecanismo de classificação em si, esse também pode ser visto como um estudo de caso de expressividade, cujo objetivo é demonstrar uma instanciação do arcabouço de classificação. Como descrito na Seção 4.3.2, o Alite não está preso a um determinado mecanismo de classificação, sendo o MCMD apenas uma alternativa, relativamente simples, dentre várias possibilidades.

5.1 Um Arcabouço para Plataformas Virtuais

Nesta seção, é apresentada uma base para arcabouços de plataformas virtuais voltado à computação em clusters (*cluster computing*), construído com base em um estudo sistemático sobre o ranque Top500, discutido na Seção 2.1.1, o qual reflete o estado-da-arte em relação às tecnologias de plataformas de computação paralela.

5.1.1 Fundamentos, Premissas e Nomenclatura

Uma das características que distinguem a HPC Shelf de outras abordagens de HPC em nuvens é sua orientação a componentes paralelos, onde coexistem tanto componentes ditos de *software*, representando algoritmos, estruturas de dados, padrões de comunicação, estratégias de particionamento de dados e mapeamento de tarefas a processos, etc, quanto os componentes ditos de *hardware*, representando plataformas de computação paralela. Ou seja, elementos de *software* e *hardware* são tratados sob uma abstração homogênea, dentro de arquiteturas orientadas a componentes que representam sistemas de computação paralela. Para isso, existe a espécie de componentes chamada de *plataformas*, cujos componentes representam as *plataformas virtuais* da HPC Shelf. Porém, é importante ressaltar que o termo plataforma virtual, utilizado neste trabalho, não está necessariamente associado à virtualização de infraestrutura provido por gerenciadores de máquinas virtuais (VMM), como o VMware[2] e XEN[3], muito embora seja uma tendência na própria HPC Shelf. De fato, uma plataforma virtual é uma abstração para uma plataforma de computação paralela de memória distribuída.

Para facilitar o uso da HPC Shelf pelos mantenedores, algumas denominações próprias para conceitos já existentes no sistema de contratos contextuais foram introduzidos no Alite, a fim de discutir melhor plataformas de computação paralela homogêneas representadas

[2] http://www.vmware.com/br/
[3] http://www.xenproject.org/

como plataformas virtuais. São eles:

- Componentes abstratos da espécie plataforma são também chamados de *categorias de plataformas virtuais*, as quais definem plataformas virtuais de arquitetura semelhante, ou seja, cujas características que influenciam no projeto de algoritmos e estruturas de dados de computação paralela podem ser representadas segundo o mesmo conjunto de parâmetros de contexto.
- Contratos contextuais, os quais atribuem argumentos aos parâmetros de contexto de categorias de plataformas, são chamados de *perfis de plataformas virtuais*, onde todos os parâmetros de contexto são supridos com argumentos que representam as propriedades particulares de uma certa plataforma virtual que pode ser instanciada sobre um ou mais mantenedores que suportam um perfil compatível.
- Componentes (concretos) da espécie plataforma são as *plataformas virtuais* em si, derivadas a partir de perfis, as quais implementam o contrato. Uma plataforma virtual deve suprir o parâmetro de contexto referente à identificação do mantenedor da infraestrutura de computação paralela sobre a qual a plataforma virtual será instanciada. Podemos dizer que o componente concreto da espécie plataforma, ou plataforma virtual, é a implementação de um procedimento para criação de uma instância de plataforma virtual sobre a infraestrutura do mantenedor informado no seu contrato, possivelmente incluindo um conjunto de imagens de máquinas virtuais em plataformas IaaS, tais como Amazon EC2, Google Cloud Platform e Microsoft Azure, a partir das quais seus nós de processamento são instanciados.
- Instâncias de componentes (concretos) da espécie plataforma são chamados de *instâncias de plataformas virtuais*. Usando analogia, são obtidos pela execução do procedimento de instanciação do componente concreto sobre a infraestrutura de computação paralela do mantenedor.

Existem duas abordagens que poderiam ser adotadas para instanciação de plataformas virtuais por parte dos mantenedores, a partir de plataformas virtuais especificadas livremente por desenvolvedores de componentes e provedores de aplicações, com base nas categorias de componentes registradas no catálogo de componentes:

1. Na primeira abordagem, os mantenedores recebem perfis arbitrários de plataformas virtuais e são capazes de avaliar se podem instanciá-la sobre a sua infraestrutura. Caso seja possível, devem ser capazes de inferir os custos de instanciação e uso daquele perfil sobre a sua infraestrutura. Portanto, o mantenedor seria capaz de instanciar uma plataforma virtual a

partir de um perfil de plataforma arbitrário.

2. Na segunda abordagem, os mantenedores são responsáveis por definir as plataformas virtuais que são capazes de instanciar sobre a sua infraestrutura física a partir de perfis de plataformas pré-determinados que registram no catálogo de componentes através do serviço *Core*, sendo capazes de calcular previamente os custos dessas plataformas virtuais a partir de modelos de desempenho próprios. Para isso, categorias de plataformas virtuais devem incluir a identidade do mantenedor como parâmetro de contexto, pois o procedimento de instanciação de uma plataforma virtual depende da infraestrutura do mantenedor, possivelmente através de uma API proprietária.

O Alite adota a segunda abordagem. Um dos motivos para isso é sua maior simplicidade, pois permite que plataformas virtuais sejam tratadas uniformemente pelo *Core* em relação às demais espécies de componentes. Porém, há outros motivos mais fortes, uma vez que reflete melhor a realidade do uso de sistemas de computação paralela, bem como permite uma maior otimização do uso dos recursos de infraestrutura da nuvem. O mantenedor é responsável por definir quais perfis de plataforma virtual podem ser instanciados eficientemente sobre a sua infraestrutura, sendo capaz controlar o balanceamento e a ocupação dos seus recursos, bem como a valoração dos parâmetros de qualidade. Além disso, os componentes podem ser implementados segundo classes de perfis dos quais já se conhece a existência de mantenedores capazes de executá-los, ao invés de fazer suposições arbitrárias sobre as plataformas virtuais para as quais são otimizadas, sem saber se realmente existem na nuvem. Finalmente, essa segunda abordagem é diretamente compatível com plataformas IaaS existentes, onde o usuário instancia máquinas virtuais a partir de um conjunto de imagens pré-existentes, possivelmente customizadas e otimizadas para atender a interesses específicos de classes de usuários recorrentes.

Um outro problema inerente à primeira abordagem é levar a problemas de difícil tratamento computacional, como o mapeamento de um perfil de plataforma virtual arbitrário sobre uma determinada infraestrutura de computação paralela do mantenedor. Isso exigiria a definição de modelos também para as plataformas reais, lidando com toda a sua heterogeneidade. Na segunda abordagem, a maior abstração permite ainda que as plataformas de computação paralela reais estejam encapsuladas pelo serviço *Backend* do mantenedor, sendo total responsabilidade do mantenedor a política de mapeamento dos perfis sobre as plataformas reais cada vez que um perfil precisar ser instanciado. Assim, um mantenedor pode dispor de vários computadores paralelos reais sobre os quais plataformas virtuais podem ser instanciadas, sendo capaz de evoluir

sua infraestrutura sem afetar o conjunto de recursos disponibilizados na nuvem.

No catálogo de componentes do *Core*, desenvolvedores de componentes e provedores de aplicações podem escolher dentre as categorias de plataformas existentes e suprir seus parâmetros de contexto, especificando as suposições que restringem a escolha da plataforma de computação paralela alvo. Os perfis gerados a partir dessas categorias, por parte de desenvolvedores e provedores, são casados com os perfis das plataformas virtuais registradas pelos mantenedores, que representam a infraestrutura da HPC Shelf.

O Alite permite a existência de componentes abstratos definidos a partir de categorias de plataformas, suprindo todos os parâmetros de contexto necessários para especificar um perfil com exceção dos parâmetros que identificam o mantenedor. Dessa forma, mantenedores podem definir perfis de plataformas comuns entre eles, os quais variam apenas pela identificação do mantenedor. Espera-se que exista um certo conjunto de perfis padrões entre mantenedores na HPC Shelf, os quais variam de acordo com a evolução tecnológica das plataformas de computação paralela. A característica de ser capaz de lidar com a evolução dos sistemas de computação paralela do Alite é fundamental dentro do contexto de HPC.

A infraestrutura de computação paralela sob o domínio dos mantenedores é transparente ao *Core*, que apenas lida com as plataformas virtuais instanciadas sobre essa infraestrutura. Parte-se do pressuposto de que é responsabilidade do mantenedor garantir que as instâncias de plataformas virtuais atendam aos requisitos do perfil.

5.1.2 Construção de um Arcabouço de Plataformas Virtuais: Visão Geral

Com a finalidade de demonstrar a construção de uma categoria de plataformas virtuais realísticas para a HPC Shelf, foi realizado um estudo exploratório sobre os computadores paralelos classificados no ranque Top500 (TOP500.ORG, 2019). Dentre essas plataformas, predominam *clusters*, homogêneos ou heterogêneos, e MPPs (*Massive Parallel Processors*). Os *clusters* homogêneos são simples de caracterizar, uma vez que as características são comuns para todos os nós de processamento. Já os *clusters* heterogêneos apresentam perfis diferentes de nós de processamento, tornando mais difícil definir um modelo de plataforma de computação paralela que leve em consideração a heterogeneidade. As aplicações típicas de HPC, no entanto, são desenvolvidas para atuar em um conjunto homogêneo de nós de processamento, uma vez que essa é uma suposição comum no projeto de algoritmos paralelos (GRAMA, 2003). Além disso, nos computadores do Top500, mesmo os *clusters* que apresentam heterogeneidade em seus recursos,

não costumam misturar perfis diferentes de recursos na execução de uma mesma aplicação, e sim atender a perfis diferentes de usuários, entregando maior desempenho dependendo da aplicação. Com base nisso, assume-se que as plataformas virtuais da HPC Shelf possuem unidades de processamento distribuídos homogêneas, sendo a heterogeneidade abstraída sob diferentes perfis.

Outro fato marcante observado é presença de aceleradores computacionais e coprocessadores, os quais constituem atualmente uma tendência. Em junho de 2012, a quantidade de plataformas com a presença de aceleradores ou coprocessadores totalizava 52. Já em novembro do mesmo ano, esse número subiu para 62. Atualmente, na lista de Junho de 2019, são 138 plataformas.

Quanto aos processadores, todos os sistemas empregam processadores com 6 ou mais núcleos, dentre os quais $98,8\%$ empregam processadores com 8 ou mais núcleos. Em alguns casos, uma mesma plataforma possui diferentes tipos de processadores, de modo que essa diferença ocorre em nós diferentes.

Com relação às interfaces de comunicação, 395 máquinas, ou seja, 79% do total, suportam uma dentre duas tecnologias de interconexão, *Gigabit Ethernet* e *Infiniband*, apresentando características diferenciadas de topologia.

Essas informações identificadas tem o objetivo de ajudar no processo de modelagem das categorias de plataformas da HPC Shelf, bem como determinar os parâmetros de contexto de plataforma utilizados em contratos contextuais. Já na identificação das características mais relevantes, foram consideradas as disponibilizadas pelos proprietários das plataformas. Essas características aparecem em pelo menos uma das plataformas, ignorando características disponibilizadas por fabricantes de componentes e que nenhuma das plataformas citou como relevante na descrição de seus recursos. Para facilitar o entendimento das características, foi feita uma classificação das mesmas em três grupos:

1. Características dos nós de processamento, supondo plataformas homogêneas;
    a) relacionadas ao processador;
    b) relacionadas a presença de aceleradores computacionais;
    c) relacionadas ao sistema de memória.
2. Características da rede de interconexão entre os nós de processamento;
3. Características do ambiente de execução.

No grupo 1, são representadas as características através dos seus subgrupos. As características identificadas no subgrupo (1a) são:

- quantidade de processadores;
- modelo do processador;
- quantidade de núcleos por processador;
- quantidade de aceleradores;
- nome do fabricante do processador;
- frequência (*clock*);
- capacidade em cada nível de *cache* de memória: L1, L2 e L3;
- latência de cada nível de *cache* de memória: L1, L2 e L3.

Já no grupo (1b), foram identificadas as características dos aceleradores computacionais presentes nas plataformas, que variam de acordo com o tipo do acelerador. Entretanto, na lista mais recente, de Junho/2019, quase todas as plataformas utilizam GPUs (*Graphical Processing Units*) como dispositivos de aceleração, praticamente desaparecendo da lista plataformas baseadas em MICs (*Many Integrated Cores*), cuja participação vinha crescendo até que a Intel resolveu descontinuar o desenvolvimento dessa tecnologia. Dentre os modelos de GPUs, todas as plataformas que as utilizam fazem uso de GPUs da fabricante NVIDIA, muito embora existam outros fabricantes relevantes no mercado, como a AMD. Mesmo entre GPUs da NVIDIA, há diferentes modelos, de diferentes arquiteturas, cujas características distintas torna relativamente difícil, comparado com CPUs, caracterizá-las segundo os mesmos parâmetros. Portanto, os parâmetros que caracterizam aceleradores computacionais variam de acordo com o modelo do acelerador. Os parâmetros comuns aos aceleradores computacionais são listados abaixo:

- tipo de acelerador;
- modelo do acelerador;
- família do acelerador;
- nome do fabricante;
- quantidade de memória global.

No subgrupo (1c), referente ao sistema de memória, foram identificadas as seguintes características:

- quantidade de memória RAM do nó de processamento;
- *throughput* da memória RAM;
- capacidade armazenamento local.

Já no grupo (2), que apresenta as características de rede, foram identificadas:

- tipo de rede;

- topologia da rede;
- largura de bisseção;
- latência;
- topologias configuráveis pelo usuário;
- largura de banda máxima.

Por fim, no grupo (3), são apresentadas as seguintes características gerais do *cluster*:

- a quantidade de nós de processamento;
- capacidade de processamento de ponto flutuante (em FLOPs);
- razão de processamento pela potência para identificar a eficiência energética.

Com o Alite, uma grande variedade de plataformas deve estar disponível para o algoritmo de resolução da HPC Shelf. Logo, torna-se necessário o desenvolvimento de uma representação que facilite a descrição de plataformas de computação paralela. Essa representação, além de permitir a escolha correta, deve permitir o reuso da descrição de características comuns entre componentes plataforma. Pode-se citar como exemplo de reuso, a descrição de todas características de um processador usada em uma certa quantidade de plataformas paralelas. No Alite, os parâmetros são definidos através de componentes abstratos. Logo, o reuso de componentes de descrição já é inerente ao modelo.

O próprio desenvolvimento do *hardware*, na indústria, é baseado em componentes, de modo que, para montar um computador, diversos componentes são ligados formando um sistema computacional (WANG; QIAN, 2005). Esse desenvolvimento baseado em componentes levou os fabricantes a trabalharem a descrição através de linhas de produtos, dentre outras classificações presentes nos componentes. Fazendo-se uso dessa classificação de linhas de produtos, desenvolveu-se um esquema para construção de taxonomias para a classificação de plataformas virtuais na HPC Shelf, facilitando tanto o processo de definição, quanto o processo de generalização na escolha de plataformas, que será exemplificado na próxima seção para processadores (CPUs).

5.1.3 Exemplo: Um Modelo de Contratos Contextuais para Processadores

Considere a classificação de um processador. A principal característica observada na classificação de processadores é a sua arquitetura, que define o conjunto de instruções presentes. Além do conceito de arquitetura, a IBM em 1964 criou o conceito de família de processadores, onde apresentam um conjunto de diferenças entre os modelos de produtos. Este conceito é

Figura 26 – Hierarquia de qualificadores para Fabricantes de Processadores

[.MANUFACTURER [.INTEL] [.AMD] [.IBM] [.⋮]]

Fonte: Próprio Autor

Figura 27 – Hierarquia de Qualificadores para Famílias de Processadores

[.FAMILY [.INTELFAMILY [.XEONFAMILY] [.⋮]] [.AMDFAMILY [.OPTERONFAMILY] [.⋮]]
[.IBMFAMILY [.POWERFAMILY] [.⋮]] [.⋮]]

Fonte: Próprio Autor

aplicado desde o IBM/360 (STALLINGS, 2003).

Segundo Stallings (2003), as diferenças básicas dos modelos de uma família são:

- Conjunto de instruções semelhante ou idêntico;
- Sistema operacional semelhante ou idêntico;
- Velocidade aumentada;
- Número cada vez maior de portas de E/S;
- Tamanho de memórias crescente;
- Maior custo.

Essas características valem para os modelos de uma mesma família, onde tais modelos diferenciam-se por uma designação geralmente numérica que especifica a ordem crescente de desempenho dos modelos. Atualmente, uma série de novas designações são utilizadas, como o conceito de geração de processadores de uma mesma família. O próprio conceito de família é utilizado recursivamente para definir séries de processadores compatíveis. Neste trabalho, todos os agrupamentos de séries de fabricação de processadores serão tratados como famílias.

No processo de catalogação dos contratos contextuais de plataformas, é definida a hierarquia dos componentes, que pode ser observada em uma estrutura de árvore. A partir da definição hierárquica dos componentes de *hardware*, permite-se que o processo de generalização de componentes possa ocorrer inclusive na escolha de plataformas de computação, sendo a relação de subtipos definida normalmente como covariante para os parâmetros de contexto, uma vez que as propriedades de uma categoria de plataforma virtual são representados por parâmetros de contexto ofertados.

As figuras 26, 27, 28 e 29 mostram partes das hierarquias da classificação de fabricantes, famílias, séries e modelos de processadores, respectivamente. Essas hierarquias

Figura 28 – Hierarquia de Qualificadores para Séries de Processadores

[.PROCESSORSERIES [.INTELSERIES [.XEONSERIES [.*5500*] [.*E3*] [.*E5*] [.*E7*] [.⋮]] [.⋮]]
[.AMDSERIES [.OPTERONSERIES [.*3000*] [.*4000*] [.*5000*] [.⋮]] [.⋮]] [.⋮]]

Fonte: Próprio Autor

Figura 29 – Hierarquia de Qualificadores para Modelos de Processadores

[.PROCESSORMODEL [.INTELMODEL [.XEONMODEL [.XEONMODEL-E3 [.⋮]] [.XEONMODEL-E5 [.*2600*] [.*2650*] [.*2670*] [.*2680*]
[.*2690*]] [.XEONMODEL-E7 [.⋮]] [.⋮]] [.⋮]] [.⋮]]

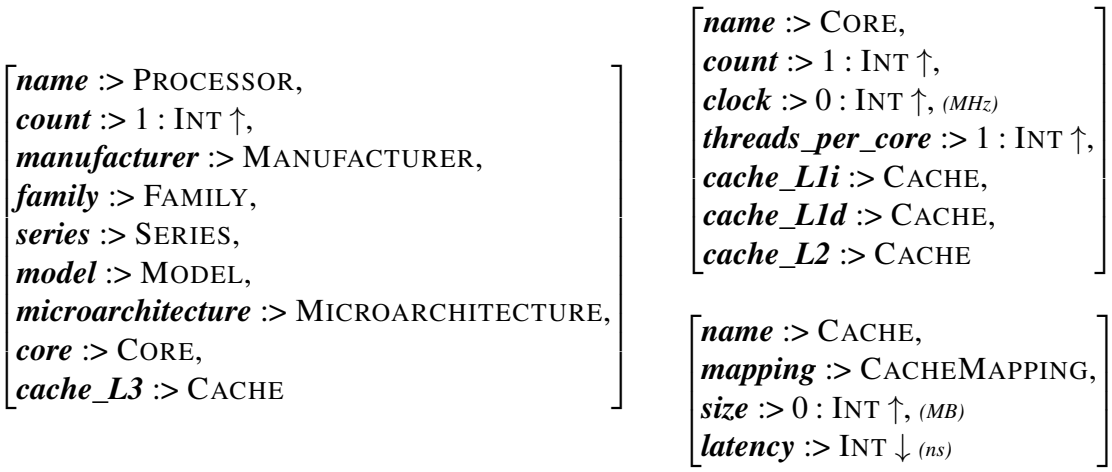
Fonte: Próprio Autor

são formadas através do registro de componentes qualificadores que definem as relações de compatibilidades entre componentes. Considere essas hierarquias nas definições dos próximos componentes que qualificam um processador.

Um processador genérico é representado pelo componente abstrato da espécie qualificador de nome PROCESSOR, cuja assinatura contextual encontra-se do lado esquerdo da Figura 30. O parâmetro de contexto ***count*** define o número de processadores em um sistema multiprocessado, enquanto os parâmetros ***manufacturer***, ***family***, ***series***, ***model*** e ***version*** identificam a origem e denominação do processador em particular. A microarquitetura do processador é determinada pelo parâmetro ***microarchitecture***. Por sua vez, o parâmetro ***core*** permite fazer suposições sobre propriedades dos núcleos de processamento do processador. Através do tipo (restrição contextual) que o restringe, chamado CORE, é possível fazer suposições acerca do número de núcleos de processamento (***count***), frequência (***clock***) e propriedades das caches de nível 1 e 2 (***cache_L1i***, ***cache_L1d*** e ***cache_L2***). Finalmente, o tipo CACHE determina o tipo de suposição que podem ser feitas sobre os níveis de cache L1, L2 e L3 do processador e seus núcleos, ou seja, o tipo de mapeamento (***mapping***), tamanho (***size***) e latência (***latency***).

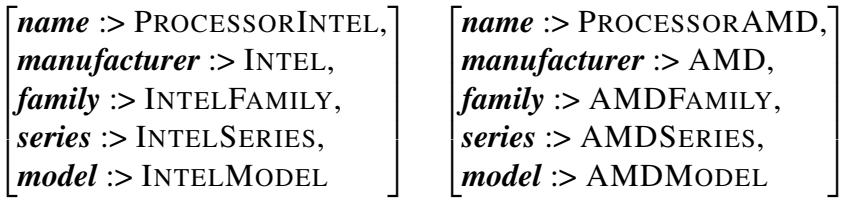
Para especializar PROCESSOR para representar um processador específico, especializamos o limite dos parâmetros de contexto e introduzimos parâmetros específicos relativos a um fabricante, família, série ou modelo específico do processador. Por exemplo, considere, na Figura 31, as assinaturas contextuais de componentes abstratos que representam processadores Intel e AMD, respectivamente, derivados de PROCESSOR. Em ambas as assinaturas, os parâmetros de contexto ***manufacturer***, ***family***, ***series*** e ***model*** tiveram suas restrições contextuais estreitadas para suportarem os valores específicos de cada fabricante. Os demais parâmetros, herdados de PROCESSOR continuam com as mesmas restrições contextuais, sendo redundante reescrevê-los.

Figura 30 – Assinatura Contextual de PROCESSOR, CORE and CACHE



Fonte: Próprio Autor

Figura 31 – Assinatura Contextual de PROCESSORINTEL e PROCESSORAMD



Fonte: Próprio Autor

Portanto, vale enfatizar, ao especializar um componente abstrato em outro, não é necessário referir-se aos parâmetros de contexto que mantém-se com os mesmos valores, mas apenas os que foram modificados. Esse tipo de açúcar sintático é importante para lidar com contratos envolvendo um grande número de parâmetros de contexto, com os tratados nesta seção. Por exemplo, se uma plataforma virtual tem um processador com contrato contextual da assinatura PROCESSORINTEL, os argumentos de contexto já são, por *default*, determinados pelas restrições contextuais dos parâmetros, sendo necessário apenas escrever argumentos que sejam subtipos próprios da restrição contextual. Além disso, assume-se que se o qualificador PROCESSORINTEL é nominalmente declarado subtipo de PROCESSOR. Assim, ao criar um componente abstrato usando o qualificador PROCESSORINTEL como valor de *name*, esse componente abstrato é considerado derivado do componente abstrato cujo parâmetro ***name*** têm valor PROCESSOR. Porém, para isso, é preciso que as seguintes restrições sobre o catálogo de componentes sejam válidas:

- Não podem existir dois ou mais componentes abstratos com a mesma restrição contextual

Figura 32 – Assinatura Contextual de PROCESSORINTELXEON

$$\text{PROCESSORINTELXEON}^{*}\ \begin{bmatrix} \textbf{\textit{manufacturer}} :> \text{INTEL}, \\ \textbf{\textit{family}} :> \text{XEONFAMILY}, \\ \textbf{\textit{series}} :> \text{XEONSERIES}, \\ \textbf{\textit{model}} :> \text{XEONMODEL} \end{bmatrix}$$

Fonte: Próprio Autor

Figura 33 – Assinaturas Contextuais para Processadores das Séries E5 e 5500, da Família Xeon da Fabricante Intel.

$$\begin{bmatrix} \textbf{\textit{name}} :> \text{PROCESSORINTELXEON-E5}, \\ \textbf{\textit{manufacturer}} :> \text{INTEL}, \\ \textbf{\textit{family}} :> \text{XEONFAMILY}, \\ \textbf{\textit{series}} :> \text{XEONSERIES-E5}, \\ \textbf{\textit{model}} :> \text{XEONMODEL-E5}, \\ \textbf{\textit{core-count}} :> 4 : \text{INT} \uparrow, \\ \textbf{\textit{core-threadscount}} = 2, \\ \textbf{\textit{core-clock}} :> 1.8 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L1i-size}} :> 128 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L1d-size}} :> 128 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L2-size}} :> 1 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L3-size}} :> 10 : \text{INT} \uparrow \textit{(MB)}, \\ \textbf{\textit{cache\_L1i-latency}} :> 2 : \text{INT} \downarrow, \\ \textbf{\textit{cache\_L1d-latency}} :> 2 : \text{INT} \downarrow, \\ \textbf{\textit{cache\_L2-latency}} :> 4 : \text{INT} \downarrow, \\ \textbf{\textit{cache\_L3-latency}} :> 17 : \text{INT} \downarrow \textit{(ns)} \end{bmatrix} \quad \begin{bmatrix} \textbf{\textit{name}} :> \text{PROCESSORINTELXEON-5500}, \\ \textbf{\textit{manufacturer}} :> \text{INTEL}, \\ \textbf{\textit{family}} :> \text{XEONFAMILY}, \\ \textbf{\textit{series}} :> \text{XEONSERIES-5500}, \\ \textbf{\textit{model}} :> \text{XEONMODEL-5500}, \\ \textbf{\textit{core-count}} :> 4 : \text{INT} \uparrow, \\ \textbf{\textit{core-clock}} :> 1 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L1i-size}} :> 128 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L1d-size}} :> 128 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L2-size}} :> 1 : \text{INT} \uparrow, \\ \textbf{\textit{cache\_L3-size}} :> 4 : \text{INT} \uparrow \textit{(MB)}, \\ \textbf{\textit{cache\_L1i-latency}} :> 1 : \text{INT} \downarrow, \\ \textbf{\textit{cache\_L1d-latency}} :> 1 : \text{INT} \downarrow, \\ \textbf{\textit{cache\_L2-latency}} :> 3 : \text{INT} \downarrow, \\ \textbf{\textit{cache\_L3-latency}} :> 30 : \text{INT} \downarrow \textit{(ns)} \end{bmatrix}$$

Fonte: Próprio Autor

sobre o parâmetro de contexto ***name***;

- Se é declarado um componente abstrato com valor de ***name*** restringido pelo qualificador N e existe um qualificador M tal que N <: M, então deve existir obrigatoriamente um componente abstrato cujo ***name*** é limitado pelo qualificador M.

Para representar processadores de uma mesma família (Xeon) de um mesmo fabricante (Intel), define-se a assinatura na Figura 32, usando a versão "açucarada" da sintaxe de apresentação das assinaturas. Nesse caso, PROCESSORINTELXEON é uma especialização de PROCESSORINTEL, de modo que os parâmetros ***family***, ***series*** e ***model*** tem suas restrições contextuais estreitadas para os valores referentes à família.

Por sua vez, para representar processadores de uma mesma série, tais como E5 e 5500 da família Intel Xeon, temos as assinaturas contextuais apresentadas na Figura 33. Para PROCESSORINTELXEON-E5 e PROCESSORINTELXEON-5500, os limites dos parâmetros ***series***

Figura 34 – Contratos Contextuais para Processadores dos Modelos 2620v4 e E5540 das Séries E5 e 5500 da Família Xeon da Fabricante Intel, Respectivamente.

name = PROCESSORINTELXEON-2620V4,	***name*** = PROCESSORINTELXEON-E5540,
manufacturer = INTEL,	***manufacturer*** = INTEL,
family = XEONFAMILY,	***family*** = XEONFAMILY,
series = XEONSERIES-E5,	***series*** = XEONSERIES-5500,
model = XEONMODEL-E5-2620V4,	***model*** = XEONMODEL-5500-E5540,
microarchitecture = BROADWELL,	***microarchitecture*** = NEHALEM,
core-count = 8,	***core-count*** = 4,
core-threads_per_core = 2,	***core-threads_per_core*** = 2,
core-clock = 3000,	***core-clock*** = 2500,
cache_L1-size = 640,	***cache_L1-size*** = 640,
cache_L2-size = 2560,	***cache_L2-size*** = 2560,
cache_L3-size = 2560,	***cache_L3-size*** = 2560,
cache_L1-latency = 1,	***cache_L1-latency*** = 1,
cache_L2-latency = 4,	***cache_L2-latency*** = 3,
cache_L3-latency = 14	***cache_L3-latency*** = 18

Fonte: Próprio Autor

Figura 35 – Hierarquia de componentes definida nos exemplos de Processadores
[.PROCESSOR [.PROCESSORINTEL [.PROCESSORINTELXEON [.PROCESSORINTELXEON-E5 [.PROCESSORINTELXEON-E5-2620V4] [.PROCESSORINTELXEON-5540]]]] [.⋮]]

Fonte: Próprio Autor

e ***model*** foram alterados para refletir valores suportados pela série, já que pertencem ao mesmo fabricante e família das assinaturas a partir das quais foram respectivamente derivados. Além disso, são também estreitadas as restrições contextuais que descrevem propriedades dos núcleos de processamento e das memórias cache. No caso, as novas restrições contextuais refletem valores garantidos para os respectivos parâmetros de contexto em relação a processadores da mesma série. Para todos os parâmetros em questão, tratam-se de valores mínimos. Por exemplo, assume-se que processadores Intel Xeon da série E5 garantem pelo menos 4 núcleos de processamento. Caso os processadores de uma mesma série possuam uma mesma valoração para um certo parâmetro de contexto, é possível fixar esse valor. Por exemplo, para a série Intel Xeon E5, é assumido que todos os núcleos de processamento possuem 2 *threads*, fixando o parâmetro ***core-threadscount***.

No nível do modelo, todos os parâmetros de contexto fixam os valores (argumentos) que representam os valores esperados pelo modelo de processador específico. Ao fixar o parâmetro ***name***, chega-se ao nível de contrato contextual. Por exemplo, para registrar, no catálogo

Figura 36 – Assinatura Contextual de CLUSTER

$$\text{CLUSTER}\ \left[\begin{array}{l} \textbf{\textit{maintainer}} :> \text{MAINTAINERID}, \\ \textbf{\textit{virtual}} :> \text{BOOLEAN}, \\ \textbf{\textit{dedicated}} :> \text{BOOLEAN} \\ \textbf{\textit{node}} :> \text{NODE}, \\ \textbf{\textit{locale}} :> \text{ANYWHERE}, \\ \textbf{\textit{interconnect}} :> \text{INTERCONNECT}, \\ \textbf{\textit{storage}} :> \text{STORAGE}, \\ \textbf{\textit{performance}} :> \text{PERFORMANCE}, \\ \textbf{\textit{power}} :> \text{POWER}, \\ \textbf{\textit{cost}} :> \text{INT} \downarrow \textit{(monetary units)} \end{array}\right]$$

Fonte: Próprio Autor

de componentes, os modelos de processadores Xeon 2620v4 e Xeon E5540, respectivamente das séries E5 e 5500 da Intel, são precisos componentes concretos para contratos contextuais de PROCESSORINTELXEON-E5-2620V4 e PROCESSORINTELXEON-E5540, apresentados na Figura 34. Portanto, uma plataforma virtual específica terá, no seu contrato, uma definição completa do modelo de processador que utiliza. Para isso, basta referir-se ao nome do contrato contextual (e.g. PROCESSORINTELXEON-E5-2640V4) que representa o modelo específico de processador.

Novas funcionalidades, em termos de componentes de *hardware* mais modernos, podem ser incorporadas em qualquer nível da hierarquia, devendo ser aplicável aos novos componentes que herdam suas características. Essa estratégia de representação facilita a definição de novos modelos de processadores, além de permitir um maior grau de generalização por parte dos desenvolvedores de componentes, que podem utilizar qualquer característica de famílias como limite. Todos os demais componentes classificados hierarquicamente deverão ser definidos na HPC Shelf da mesma maneira, para facilitar o reuso de suas descrições.

5.1.4 Uma Categoria de Plataformas Virtuais para Cluster Computing

A Figura 36 representa a assinatura contextual de uma categoria de plataformas virtuais, chamada CLUSTER, que representa *clusters* constituídos de um conjunto homogêneo de nós de processamento equipados com 1 ou mais processadores de múltiplos núcleos. Cada nó pode estar equipado com 1 ou mais aceleradores computacionais, de mesmo tipo para todos os nós, bem como um armazenamento local de tamanho uniforme. O *cluster* possui ainda um

Figura 37 – Assinatura Contextual de NODE

$$\text{NODE}\left[\begin{array}{l}\textbf{\textit{count}} :> 1 : \text{INT} \uparrow, \\ \textbf{\textit{processor}} :> \text{PROCESSOR}, \\ \textbf{\textit{accelerator}} :> \text{ACCELERATOR}, \\ \textbf{\textit{memory}} :> \text{MEMORY}, \\ \textbf{\textit{storage}} :> \text{STORAGE}, \\ \textbf{\textit{operating\_system}} :> \text{OPERATINGSYSTEM}\end{array}\right]$$

Fonte: Próprio Autor

armazenamento global, que pode ser acessado por cada nó de processamento a partir da rede de interconexão. Vale ressaltar que a categoria CLUSTER representa uma arquitetura particular para uma plataforma de *cluster computing*, que consideramos representativo. Em uma implementação real da HPC Shelf, várias categorias poderiam ser definidas, representando projetos alternativos de um *cluster*.

A assinatura contextual da categoria de plataformas virtuais CLUSTER possui um conjunto parâmetros de contexto que descrevem as principais suposições que desenvolvedores de componentes devem fazer a respeito da arquitetura de um *cluster* que ela representa. Os parâmetros são descritos nos próximos parágrafos.

O parâmetro ***maintainer*** serve para especificar o mantenedor, e portanto o serviço Backend, responsável pela instanciação da plataforma virtual. O parâmetro ***virtual*** serve para especificar se o cluster é instanciado usando serviço de virtualização ou constituem máquinas físicas, completas ou partições. Para isso, o qualificador BOOLEAN, com subtipos TRUE e FALSE, é usado como restrição contextual. Por sua vez, o parâmetro ***dedicated***, também restringido por BOOLEAN, especifica se o hardware sobre o qual o cluster encontra-se instanciado é dedicado a executar uma única aplicação ou é de locação múltipla[4], ou seja, compartilhado entre várias aplicações.

Cada nó é do tipo restringido pelo tipo NODE, através do parâmetro ***node***. A assinatura contextual do componente abstrato NODE está na Figura 37. O parâmetro ***count*** restringe o número de nós de processamento distribuído do *cluster*. Por sua vez, parâmetros ***processor***, ***accelerator***, ***memory***, ***storage*** e ***operating_system*** permitem restringir propriedades que dizem respeito aos processadores, aceleradores, memória principal, armazenamento persistente (local) e sistema operacional do nó de processamento, respectivamente através dos qualificadores PROCESSOR, ACCELERATOR, MEMORY, STORAGE e OPERATINGSYSTEM, cujas assinatu-

[4] *multi-tenancy*

Figura 38 – Assinatura Contextual de ACCELERATOR

$$\text{ACCELERATOR*} \begin{bmatrix} \textbf{\textit{count}} :> 1 : \text{INT} \uparrow, \\ \textbf{\textit{manufacturer}} :> \text{ACCELERATORMANUFACTURER}, \\ \textbf{\textit{type}} :> \text{ACCELERATORTYPE}, \\ \textbf{\textit{architecture}} :> \text{ACCELERATORARCHITECTURE}, \\ \textbf{\textit{model}} :> \text{ACCELERATORMODEL}, \\ \textbf{\textit{memory\_size}} :> 0 : \text{INT} \uparrow, \textit{(MB)} \end{bmatrix}$$

Fonte: Próprio Autor

Figura 39 – Assinatura Contextual de MEMORY

$$\text{MEMORY*} \begin{bmatrix} \textbf{\textit{size}} :> 0 : \text{INT} \uparrow, \textit{(MB)} \\ \textbf{\textit{latency}} :> \text{INT} \downarrow, \textit{(ns)} \\ \textbf{\textit{bandwidth}} :> 0 : \text{INT} \uparrow \textit{(Mb/s)} \end{bmatrix}$$

Fonte: Próprio Autor

Figura 40 – Assinatura Contextual de STORAGE

$$\text{STORAGE*} \begin{bmatrix} \textbf{\textit{size}} :> 0 : \text{INT} \uparrow, \textit{(GB)} \\ \textbf{\textit{latency}} :> \text{INT} \downarrow, \textit{(ns)} \\ \textbf{\textit{bandwidth}} :> 0 : \text{INT} \uparrow \textit{(Mb/s)} \end{bmatrix}$$

Fonte: Próprio Autor

ras contextuais estão nas figuras 30, 38, 39 e 40, com exceção de OPERATINGSYSTEM, cuja assinatura é vazia nesse nível.

A assinatura contextual de PROCESSOR já foi discutida na seção anterior (Seção 5.1.3). Em relação à assinatura ACCELERATOR, há parâmetros para restringir o número de unidades do acelerador (***count***), o seu fabricante (***manufacturer***), o seu tipo[5] (***type***), a sua arquitetura dentro de um mesmo tipo de acelerador (***architecture***), o seu modelo (***model***) e sua capacidade de memória própria (***memory_size***). Dependendo do tipo de acelerador, outros parâmetros podem ser incluídos na assinatura de componentes abstratos derivados de ACCELERATOR, tais como representando o número de núcleos internos de processamento, que é de caracterização bastante diversa entre aceleradores de diferentes tipos. Por sua vez, os contratos de MEMORY e STORAGE permitem restringir o tamanho (***size***) e desempenho dos sistemas de memória volátil e armazenamento persistente, de modo que o desempenho desses sistemas é caracterizado pela sua latência (***latency***) e sua largura de banda (***bandwidth***). Embora possuam o mesmo conjunto

[5] e.g. GPU, FPGA, MIC, etc

Figura 41 – Assinatura Contextual de INTERCONNECT

$$\text{INTERCONNECT*} \begin{bmatrix} \textbf{\textit{startup\_time}} :> \text{INT} \downarrow, \textit{(ns)} \\ \textbf{\textit{node\_latency}} :> \text{INT} \downarrow, \textit{(ns)} \\ \textbf{\textit{bandwidth}} :> 0 : \text{INT} \uparrow \textit{(Mb/s)}, \\ \textbf{\textit{topology}} :> \text{NETWORKTOPOLOGY} \end{bmatrix}$$

Fonte: Próprio Autor

Figura 42 – Assinatura Contextual de PERFORMANCE

$$\text{PERFORMANCE} \begin{bmatrix} \textbf{\textit{peak\_flops}} :> 0 : \text{INT} \uparrow, \textit{(GFlops)} \\ \textbf{\textit{hpl\_flops}} :> 0 : \text{INT} \uparrow, \textit{(GFlops)} \\ \textbf{\textit{hpcg\_flops}} :> 0 : \text{INT} \uparrow \textit{(GFlops)} \end{bmatrix}$$

Fonte: Próprio Autor

de parâmetros de contexto, optou-se por separá-los em qualificadores distintos, uma vez que parâmetros mais específicos podem ser incluídos em categorias derivadas de CLUSTER para representar propriedades particulares de seus sistemas de memória e armazenamento.

O parâmetro ***network*** permite restringir características da rede de interconexão entre os nós de processamento do cluster, através de um contrato de INTERCONNECT, cuja assinatura está na Figura 41. Além de parâmetros descrevendo o tempo de *startup* (***startup_time***), a latência de nó (***node_latency***) e largura de banda (***bandwidth***) da rede, há o parâmetro ***topology***, permitindo fazer restrições a respeito da topologia de interconexão da rede. Na assinatura NETWORKTOPOLOGY, embora a princípio com um conjunto vazio de parâmetros de contexto para os propósitos desta Tese, poderiam ser incluídos parâmetros que descrevem características da topologia que frequentemente são relevantes para implementação de operações de comunicação, tais como *diâmetro*, *conectividade*, *largura do canal*, *taxa do canal*, *tamanho da bisseção*, *largura de banda da bisseção*, dentre outros (GRAMA, 2003).

O parâmetro ***storage*** restringe características do armazenamento persistente global, acessível aos nós do cluster através da rede de interconexão. A assinatura do seu tipo, STORAGE (Figura 40), foi discutida anteriormente.

Os parâmetros ***performance*** e ***power*** permitem fazer suposições a respeito do desempenho e da eficiência energética do cluster, usando os *benchmarks* consolidados pelo ranque Top500: HPL (*High Performance Linpack*) e HPCG (*High Performance Conjugate Gradients*). Portanto, um contrato de PERFORMANCE, cuja assinatura está apresentada na Figura 42, possui

Figura 43 – Assinatura Contextual de POWER

$$\text{POWER}\begin{bmatrix} \textbf{\textit{hpl\_total}} :> \text{INT} \downarrow, \textit{(Watt)} \\ \textbf{\textit{hpcg\_total}} :> \text{INT} \downarrow, \textit{(Watt)} \\ \textbf{\textit{hpl\_efficiency}} :> 0 : \text{INT} \uparrow, \textit{(GFlops/Watt)} \\ \textbf{\textit{hpcg\_efficiency}} :> 0 : \text{INT} \uparrow \textit{(GFlops/Watt)} \end{bmatrix}$$

Fonte: Próprio Autor

parâmetros que permitem fazer restrições sobre a capacidade de processamento em termos de operações de ponto-flutuante por unidade de tempo. Enquanto ***peak_flops*** representa o pico de desempenho teórico do cluster, os parâmetros ***hpl_flops*** e ***hpcg_flops*** representam, respectivamente, o desempenho medido através dos benchmarks HPL e HPCG. Por sua vez, um contrato de POWER, cuja assinatura encontra-se na Figura 43, possui parâmetros para os valores absolutos (***hpl_total*** e ***hpcg_total***) e relativos (***hpl_efficiency*** e ***hpcg_efficiency***) do custo energético do cluster tanto para o HPL quanto para ao HPCG. O valor relativo é uma medida de eficiência energética, em termos de GFlops/Watt. Essa medida é usada na classificação do ranque Green500 [6], que classifica as máquinas classificadas no Top500 de acordo com a eficiência energética.

Finalmente, o parâmetro **cost** representa o custo monetário de uso do cluster pela aplicação, desde o momento em que é instanciado até o momento quando é liberado. Como discutido antes, esse valor deve ser calculado em um componente sistema a partir do modelo de desempenho do componente computação sobre a plataforma virtual.

Na Figura 44, é apresentado um perfil de plataforma virtual que representa a menor instância da HPC Shelf possível para um mantenedor hipotético associado ao CENAPAD-UFC, e que possui um único acelerador computacional. Admite-se, portanto, que CENAPAD-UFC-MICRO-GPU é declarado nominalmente como um subtipo de CLUSTER no catálogo de componentes. Admite-se ainda que PROCESSORINTELXEON-2470, LINUXREDHAT, GPU-TESLA-K20M, NETWORK-HCA-400EX-D-GS são contratos contextuais de componentes concretos que representam o processador, sistema operacional, tipo de acelerador e tipo de interconexão do cluster em questão, com argumentos de contexto definidos, muito embora omitidos deste texto.

Assim como os usuários especialistas podem utilizar os recursos de plataformas de execução expostos por mantenedores por meio da HPC Shelf, também podem utilizar seus próprios recursos, bastando instalar um *Backend* da HPC Shelf e assumir o papel de mantenedor,

[6] <http://www.green500.org>

Figura 44 – Contrato contextual CENAPAD-UFC-MICRO-GPU

$$\text{CENAPAD-UFC-MICRO-GPU} \begin{bmatrix} \textbf{\textit{node-count}} = 2, \\ \textbf{\textit{node-processor-count}} = 2, \\ \textbf{\textit{node-memory-size}} = 96000, \\ \textbf{\textit{node-storage}} = 250000, \\ \textbf{\textit{node-processor}} = \text{PROCESSORINTELXEON-2470}, \\ \textbf{\textit{node-operating\_system}} = \text{LINUXREDHAT}, \\ \textbf{\textit{node-accelerator}} = \text{GPU-TESLA-K20M}, \\ \textbf{\textit{network}} = \text{NETWORK-HCA-400EX-D-GS}, \\ \textbf{\textit{locale}} = \text{FORTALEZA}, \\ \textbf{\textit{virtualized}} = \text{FALSE}, \\ \textbf{\textit{dedicated}} = \text{FALSE} \end{bmatrix}$$

Fonte: Próprio Autor

criando um ou mais perfis de plataformas virtuais para representar seus próprios recursos computacionais. Vale ressaltar que, atualmente, existe uma implementação de um *Backend* capaz de instanciar plataformas virtuais em uma máquina local baseada em Linux, porém sem usar recurso de virtualização, além de outras três, respectivamente voltadas a instanciação de plataformas virtuais em um cluster com suporte a tecnologia OpenStack, bem como sobre os serviços Amazon EC2 e Google Cloud Platform.

5.2 Arcabouço de Plataformas Virtuais sobre Amazon EC2

No Alite, a modelagem de plataformas virtuais permite a definição estática de plataformas de *clusters* locais (*on-premise*) em centros de pesquisa, seja com seu uso em modo nativo ou virtualizado. Além disso, o Alite suporta a modelagem de plataformas de perfis estáticos oferecidos por provedores de IaaS. Nesta seção, apresentamos um estudo de caso de modelagem de um arcabouço de plataformas virtuais sobre a infraestrutura oferecida pelo serviço EC2 da nuvem da Amazon, o qual pode servir de base para arcabouços análogos aplicados a outros serviços IaaS.

5.2.1 Fundamentos, Premissas e Nomenclatura

De acordo com Belusso *et al.* (2018), os principais provedores de IaaS, no que se refere à quantidade de usuários, são os provedores *Amazon Elastic Compute Cluster* (EC2), o *Google Compute Engine* e o *Microsoft Azure* (SERVICES, 2019a; ENGINE, 2019; MICROSOFT, 2019). Com base nessa afirmação, bem como nos estudos apresentados na Seção 2.2, tais

provedores foram investigados para determinação de categorias de plataformas virtuais sobre nuvens computacionais que oferecem serviços IaaS.

De modo geral, provedores IaaS suportam plataformas de computação paralela através da aglomeração de instâncias de máquinas virtuais que podem se comunicar através de uma rede, formando *clusters*, independente de suas localizações geográficas, distinguindo-se nos aspectos influenciados pela adoção de tecnologia virtualização, existência de serviços de elasticidade e capacidade de instanciar máquinas virtuais no escopo de uma mesma rede, oferecendo garantias de desempenho de comunicação. Para demonstrar a viabilidade de expressar, por meio do sistemas de contratos contextuais do Alite, plataformas virtuais sobre provedores IaaS, apresentaremos a assinatura contextual de uma categoria de plataformas virtuais para o serviço EC2 do provedor Amazon, bem como um conjunto de perfis de plataformas virtuais sobre tal categoria, representando plataformas virtuais que podem ser construídas a partir de tipos de instâncias suportadas pelo EC2.

Dada a natureza elástica dos serviços IaaS e a definição estática dos perfis de plataforma do Alite, as plataformas modeladas apresentam um *hardware* inicialmente estático, mas que poderiam ser alteradas dinamicamente pelo *Backend* durante a execução do componente, a fim de atender requisitos de QoS e de custo da aplicação. A elasticidade de componentes na HPC Shelf é um assunto que foi abordado na Tese de Doutorado de João Marcelo U. de Alencar (ALENCAR, 2017; ALENCAR; Carvalho Junior, 2019), mas não será tratada nesta Tese.

Seguindo o que foi discutido na Seção 5.1.1, uma vez que seja necessário definir um perfil de plataforma personalizado em um provedor IaaS, um mantenedor de plataformas deve criar tal perfil e inseri-lo no catálogo de componentes para que esse possa ser escolhido pelo algoritmo de resolução do Alite.

Na definição dos tipos de máquinas virtuais que suportam, tanto o *Amazon EC2* quanto o *Microsoft Azure* trabalham com tipos de instâncias que relacionam o propósito da máquina, as capacidades de CPU, a memória de interconexão e os processadores/coprocessadores utilizados. Por sua vez, o *Google Compute Engine* define quantidades pré-definidas com base no propósito da máquina, quantidades de processadores/núcleos e capacidade de memória, possibilitando a escolha dos processadores/coprocessadores de acordo com um conjunto de regras de disponibilidade por perfil disponível.

O modelo de tarifação de máquinas virtuais depende do provedor, não existindo uma regra ou arcabouço geral. A previsão de custos deve ser feita através de uma calculadora

disponibilizada pelo próprio provedor. De acordo com Belusso *et al.* (2018), diversos trabalhos já propuseram modelos de previsão de custos para ambientes de nuvens de IaaS, o que não é uma tarefa simples, dadas as características de localização, capacidade e modelos de compra, fatores que afetam diretamente no custo por hora da máquina virtual. Um provedor de IaaS apresenta uma diversidade de provedores físicos de máquinas virtuais, onde cada provedor possui uma tabela de preços diferenciada. Dependendo do modelo de contratação das máquinas virtuais, o custo pode ser reduzido. Esses provedores, de modo geral, apresentam a tarifação mais alta no modelo sob demanda (*on-demand*), em que a máquina virtual é tarifada somente por hora utilizada. Outros modelos, como o *spot* da Amazon EC2, permitem que, partindo de um perfil de instância, localização e preço máximo pré-definidos pelo consumidor, a contratação seja definida nos moldes de um leilão de recursos, tornando a execução susceptível a aguardar disponibilidade da oferta do recurso requisitado por um preço dentro do limite definido, ou até mesmo a sua interrupção, quando o preço aumentar acima do limiar máximo. Nos três provedores anteriormente citados, existe a possibilidade de reservar instâncias por um período de tempo pré-definido, o que reduz o preço de acordo com o período contratado, chamadas de instâncias reservadas nos provedores Amazon EC2 e Microsoft Azure e instâncias de uso contínuo no *Google Compute Engine*. O *Google Compute Engine* ainda apresenta redução de custos para instâncias de uso prolongado, proporcional ao tempo que a máquina virtual é utilizada dentro do mês, começando em 25% do mês, até 100% do mês (ENGINE, 2019)

Cada provedor IaaS apresenta um conjunto de peculiaridades no provimento de recursos computacionais que alteram o desempenho de uma instância, e cada provedor cria sua nomenclatura para tais serviços. A Amazon EC2 disponibiliza um serviço de armazenamento de blocos chamado *Amazon Elastic Block Store* (Amazon EBS), que pode ser anexado às instâncias e provê armazenamento persistente independente do ciclo de vida da instância, estando localizado na mesma região de disponibilidade. No que se refere ao armazenamento persistente na própria instância, a EC2 também disponibiliza, para algumas instâncias, discos magnéticos (HDD), drives de estado sólido (SSD) e unidades NVMe, sendo que essas últimas apresentam desempenho superior aos SSDs convencionais. No entanto, os dados serão perdidos se a instância for encerrada, parada ou se apresentar defeito em alguma unidade de armazenamento. Existem outros serviços de armazenamento persistentes, mas fogem ao escopo usual de computação em *clusters*, onde tarefas são enviadas para serem processadas e retornam os resultados ao final do processamento.

Além do armazenamento, a característica chamada *grupo de alocação* (*placement group*)[7], no serviço EC2, é relevante para os perfis do Alite, pois permite algumas restrições físicas em relação ao local de instanciação. São disponibilizados grupos de alocação com as seguintes estratégias de alocação: *clusterização*, *particionamento* e *espalhamento*. Na estratégia de clusterização, todas as instâncias pertencentes estarão localizadas fisicamente na mesma região de disponibilidade, interligadas por rede de alto desempenho e baixa latência. Já na estratégia de particionamento, o serviço EC2 garante que instâncias não compartilhem *hardware* subjacente, promovendo isolamento. Por fim, a estratégia de espalhamento distribui as instâncias buscando aumentar a disponibilidade.

Em relação à rede de interconexão, o serviço EC2 também disponibiliza serviços especializados, como é o caso do *Elastic Network Adapter* (ENA)[8], um serviço de rede avançada que provê alto desempenho de pacotes por segundo com latências consistentemente baixas. Nesse serviço, alguns tipos de instâncias apresentam um modelo de créditos de E/S, em que o desempenho elevado está associado ao uso intermitente de rede e tem seus créditos recarregados em momento de consumo abaixo do valor de referência (*baseline*) para o uso de E/S.

Uma vez que, no Alite, o mantenedor de plataformas é responsável por determinar os perfis de plataforma virtual a serem registrados no catálogo de componentes, ele deve ter profundo conhecimento do modelo de tarifação e das limitações contratuais do provedor IaaS. Portanto, é de sua responsabilidade fazer escolhas como o tipo de instância, serviços adicionais, restrição de grupos de alocação e custos de execução. O Alite terá conhecimento apenas dos parâmetros que determinam quantidades e nível de desempenho, não sendo consciente, por exemplo, da tarifação de instâncias de uso intermitente.

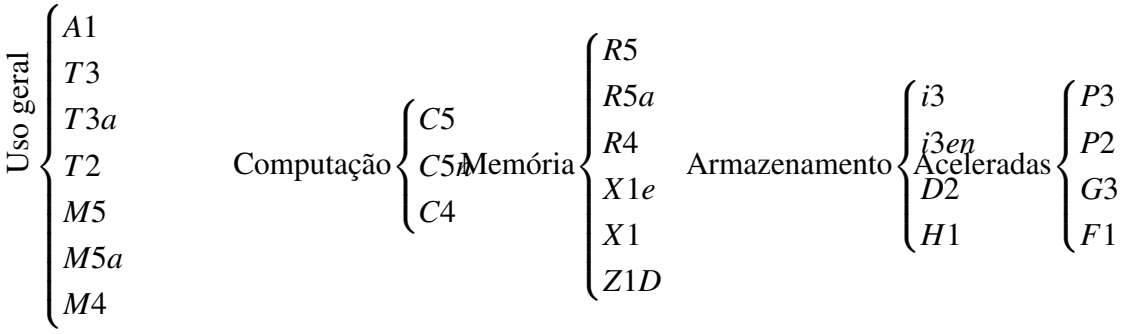
Nas próximas seções, serão introduzidos os tipos de instâncias, e serão discutidos os parâmetros disponibilizados pelo provedor Amazon EC2, usando informações extraídas a partir de sua página *WEB* (SERVICES, 2019a). Tais parâmetros constituem a base para construção de uma assinatura contextual para a categoria de plataformas virtuais instanciadas sobre o serviço EC2.

---

[7] <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html>

[8] <https://aws.amazon.com/blogs/aws/elastic-network-adapter-high-performance-network-interface-for-amazon-ec2>

Figura 45 – Resumo das famílias da nuvem Iaas da Amazon



Fonte: Próprio Autor

5.2.2 Tipos de Instância

O serviço EC2 da Amazon foi escolhido para demonstrar a expressividade do sistema de contratos contextuais no suporte a instanciação de perfis de plataformas virtuais sobre a infraestrutura oferecida por provedores IaaS em geral.

O serviço EC2 disponibiliza o suporte à criação de instâncias de máquinas virtuais de vários *modelos*, de forma a atender usuários com diferentes requisitos. Para isso, tais modelos são agrupados segundo uma hierarquia. No nível mais alto, estão as *famílias* de instâncias, descritas a seguir:

- instâncias de uso geral;
- instâncias para aplicações com requisitos severos de capacidade e velocidade de acesso do sistema de memória;
- instâncias para aplicações com requisitos de computação intensiva;
  - baseadas no uso das CPUs;
  - baseadas no uso de aceleradores (GPU ou FPGA);
- instâncias para aplicações com requisitos severos de armazenamento não-volátil.

Cada família de instâncias reúne *tipos* de instâncias com características comuns em relação às capacidades do computador hospedeiro[9]. A Figura 45 resume as famílias e os tipos de instâncias, as quais terão suas características detalhadas, no nível dos modelos de cada tipo, nas tabelas 7, 8, 9, 10, 11, 12 e 13.

As instâncias de uso geral tem capacidade de moderado desempenho. A capacidade de processamento parte de 1 núcleo virtual, com instâncias de até 96 núcleos. As quantidades de

[9] <https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/instance-types.html> e <https://aws.amazon.com/ec2/instance-types>

memória principal variam entre $500MB$ e $384GB$, enquanto o armazenamento interno varia desde instâncias sem armazenamento persistente próprio, utilizando volumes EBS para armazenamento em rede, até as instâncias que possuem SSD com até $4 \times 900GB$. Nessa família de instâncias, existem os tipos de instâncias de desempenho com capacidade de intermitência, chamadas de T, onde picos de processamento acima do valor de referência de desempenho são suportados, consumindo créditos de desempenho. Quando o percentual de processamento estiver acima de um valor de referência, são consumidos créditos de vCPU com uma taxa de verificação em milissegundos. No restante do período de utilização, durante o período de uma hora, quando houver ociosidade ou o processamento estiver abaixo do valor de referência, há o acúmulo de créditos de vCPU. Assim, para usos equilibrados, esses tipos de instâncias são adequados. Se exceder as quantidades de créditos de vCPU disponíveis, as instâncias serão tarifadas em uma taxa extra. O período de aquisição de consumo e de aquisição de créditos também pode ser estendido para o menor valor entre 24 horas ou o tempo em que a instância estiver ativa, chamado de instância ilimitada.

Por sua vez, as instâncias otimizadas para computação apresentam alto desempenho de processamento, sendo por isso consideradas instâncias apropriadas para requisitos de HPC. Essas instâncias oferecem atualmente de 2 a 72 núcleos de processamento virtual, variando de $4GB$ a $192GB$ de memória principal e com armazenamento persistente interno de $50GB$ até $2 \times 900GB$ SSD, quando possui armazenamento interno persistente.

As instâncias otimizadas em memória principal apresentam quantidade elevada de memória, atualmente variando entre $16GB$ e $768GB$ por instância.

Cada tipo de instância de computação baseada em aceleradores especifica o tipo de GPU ou FPGA que suporta, enquanto o modelo determina suas quantidades em cada instância. Atualmente, são disponibilizadas GPUs dos modelos Tesla V100, K80 e M60, da NVIDIA, ou ainda FPGAs Virtex UltraScale+ VU9P, da Xilinx, variando desde 4 núcleos virtuais até 96 núcleos, com memória principal entre $30,5GB$ e $768GB$.

Por fim, as instâncias otimizadas para armazenamento apresentam atualmente de 2 à 96 vCPU e armazenamento de até $16TB$ de HDD ou até $60TB$ de SSD.

Independente do tipo de instância, o serviço EC2 disponibiliza um serviço de armazenamento persistente em blocos de alta capacidade de I/O e baixa latência, com *backup* redundante na mesma localização de disponibilização e facilidade para migração entre localizações. Alguns tipos de instâncias apresentam, por padrão, otimização com uso do EBS, garantindo maior

desempenho de comunicação entre os volumes de armazenamento e as instâncias. São elas: C5, C4, M5, M4, P3, P2, G3 e D2. Instâncias dos demais tipos podem ter acesso à essa otimização através do pagamento de custos adicionais.

5.2.3 Caracterização dos Modelos de Instâncias

No serviço EC2, de acordo com as informações fornecidas em Services (2019b), Services (2019a), há, para cada tipo de instância, um conjunto padrão de parâmetros que os usuários podem usar a fim de escolher o modelo de instância mais adequado para as suas necessidades. Os seguintes parâmetros são comuns a todos os tipos de instância:

- **Modelo**, que corresponde à a denominação do modelo de instância, composta de seu tipo e seu tamanho, separados por um ponto (e.g. para o tipo instância $t3$, há os modelos $t3.nano$, $t3.micro$, $t3.small$, $t3.medium$, $t3.large$, $t3.xlarge$, $t3.2xlarge$);
- **vCPUs**, referente à quantidade de CPUs virtuais, podendo cada CPU virtual corresponder a um núcleo de processamento físico (*core*) ou a um núcleo de processamento lógico, como em processadores Intel com recursos *Hyper-Threading*;
- **Processador**, que especifica o tipo do processador usado pela instância, o qual limita o número de núcleos e *threads* por núcleo (CPUs virtuais) suportados;
- **Memória (GiB)**, representando a quantidade de memória principal alocada para a instância, sendo expressa em Gibibyte (GiB), equivalente a 2^{30} bytes;
- **Armazenamento**, representando o tipo de armazenamento local não-volátil da instância, podendo ser do tipo HD ou SSD;
- **Performance de rede (Gbps)**, representando a capacidade de largura de banda em rede em *cluster* quando as instâncias estiverem na mesma região de localização.

Cada modelo de instância pode estabelecer faixas de valores ou valores fixos para cada parâmetro. Por exemplo, certas instâncias podem variar o número de vCPUS, dependendo do modelo do processador físico suportado pela instância, configurando-se o número de núcleos e de *threads* por núcleo dentro de certos limites máximos. Por sua vez, a quantidade de memória associada a cada modelo de instância é fixa, enquanto a capacidade de armazenamento é sempre variável. Finalmente, o desempenho de rede, em termos de largura de banda, não pode sempre ser garantido dentro de um valor fixo (e.g. “10 Gigabit”), mas também através de um limiar (e.g. “Up to 10Gigabit”) ou qualitativamente (e.g. “Low to Moderate”, “Moderate”, etc).

Dentre o conjunto de parâmetros específicos de determinados tipos de instância está

Figura 46 – Assinatura Contectual de CLUSTER-EC2

$$\text{CLUSTER-EC2*}\begin{bmatrix} \textbf{\textit{maintainer}} :> \text{MAINTAINERID-EC2}, \\ \textbf{\textit{placement\_group}} :> \text{PLACEMENTGROUP}, \\ \textbf{\textit{virtual}} = \text{TRUE}, \\ \textbf{\textit{node}} :> \text{NODE-EC2}, \\ \textbf{\textit{interconnect}} :> \text{INTERCONNECT-EC2}, \\ \textbf{\textit{storage}} :> \text{STORAGE-EC2-PERSISTENT} \end{bmatrix} \text{from CLUSTER}$$

Fonte: Próprio Autor

a **largura de banda dedicada do EBS (Mbps)**, referente a instâncias de diferentes famílias (M4, M5, M5a, C4, C5, C5n, R5, R5a, X1, X1e e P3), que reflete a capacidade de transferência em megabits por segundo (Mbps) entre a instância e o serviço de armazenamento EBS. Além disso, os seguintes parâmetros são referentes a instâncias de computação acelerada:

- **Modelo GPU**, que determina o modelo da GPU empregada por instâncias de um mesmo modelo;
- **nGPU** ou **nFPGA**, representando a quantidade de aceleradores disponíveis para cada modelo de instância;
- **Memória GPU (GiB)**, representando a quantidade total de memória das GPUs, determinada pelo modelo da GPU e quantidade de GPUs do modelo de instância;
- **P2P entre GPUs**, que indica o tipo de comunicação par-a-par entre as GPUs.

5.2.4 Categoria e Perfis de Plataformas Virtuais para Amazon EC2

Para suportar os nós da Amazon EC2, a assinatura CLUSTER precisou ser estendida. Para isso, foi criada a assinatura CLUSTER-EC2, apresentada na Figura 46, que estende a assinatura CLUSTER, definindo uma plataforma virtual homogênea, com *n* nós de um mesmo modelo de instância, organizados em um *placement group* do tipo clusterização. Além disso, possui um armazenamento persistente independente do ciclo de vida das máquinas virtuais. Por sua vez, o armazenamento interno de cada nó depende do modelo de instância e da personalização do mantenedor. O mantenedor de plataformas deve ser criterioso na configuração das plataformas a fim de garantir o desempenho esperado pelos sistemas de computação paralela através dos perfis de plataformas virtuais solicitados, evitando prejuízos tanto para o usuário quanto para o próprio mantenedor.

Para cada modelo de instância, é associado um perfil de plataforma virtual (contrato),

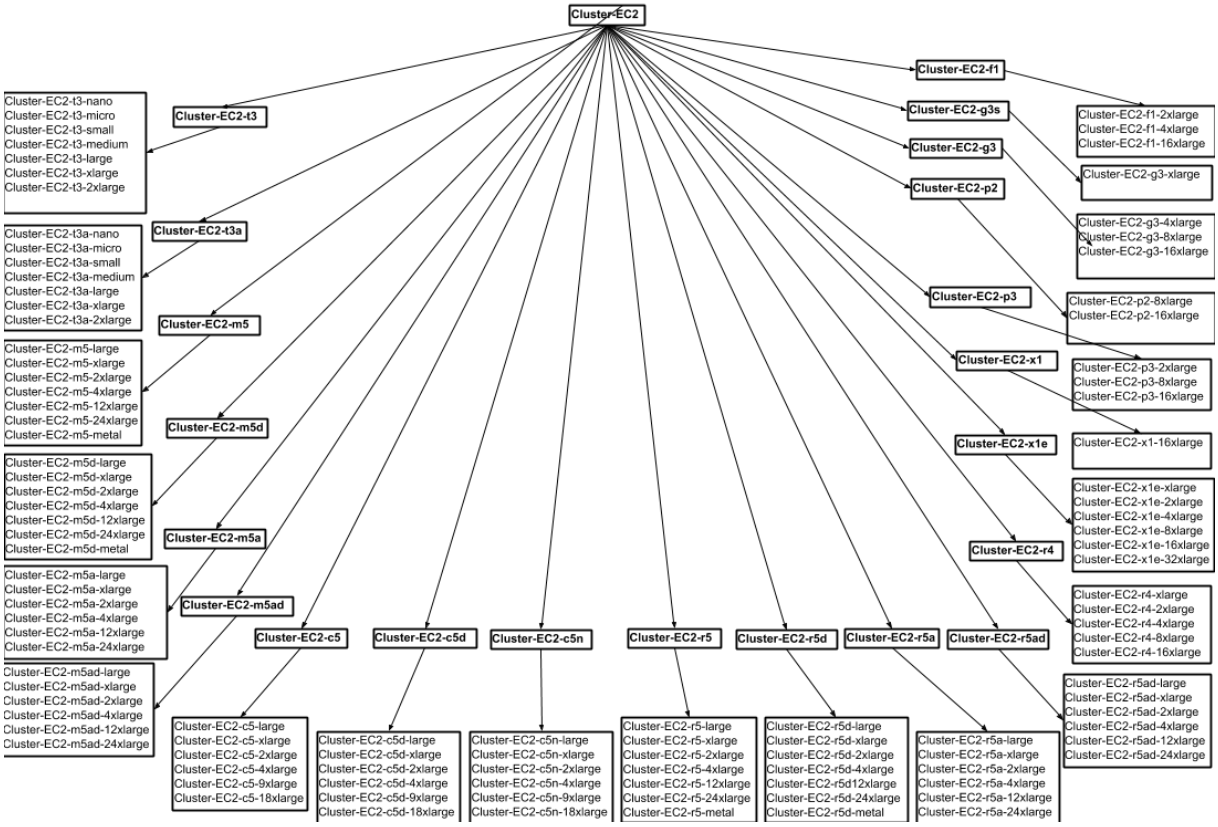
atribuindo-se, aos parâmetros de contexto de CLUSTER-EC2, argumentos que representam as características particulares do modelo de instância. Vale lembrar que, de acordo com o algoritmo de resolução, quando um sistema de computação paralela requisita uma plataforma virtual da categoria CLUSTER, um componente concreto que representa uma plataforma virtual EC2, ou seja, associado a um perfil de CLUSTER-EC2, pode ser escolhido, caso o perfil de plataforma virtual submetido seja compatível (supertipo).

Note que um sistema de computação paralela pode também restringir a requisição de plataformas virtuais à infraestrutura EC2, submetendo um perfil da categoria CLUSTER-EC2, ao invés de CLUSTER. Além disso, pode restringir a requisição um modelo de instância EC2 particular, associando o qualificador que representa o nome do perfil ao parâmetro ***name*** de CLUSTER ou CLUSTER-EC2. Entretanto, para que isso seja possível, é necessário derivar categorias de plataformas para cada modelo de instância a partir de CLUSTER-EC2. Isso é feito também para tipos de instâncias. Por exemplo, a categoria CLUSTER-EC2-T3 é derivada a partir de CLUSTER-EC2, representando o tipo de instância *t*3, enquanto a categoria CLUSTER-EC2-T3-LARGE é derivada a partir de CLUSTER-EC2-T3, representando o modelo *t*3.*large*. Dessa forma, parâmetros de contexto que possuem valorações fixas para um determinado tipo de instância devem ter suas valorações fixadas na categoria CLUSTER-EC2-T3, no caso do tipo de instância *t*3. Enfim, componentes concretos que representam plataformas virtuais baseadas em modelos de instância *t*3.*large* deve ter perfil da categoria CLUSTER-EC2-T3-LARGE, atribuindo-se argumentos a parâmetros que são mantidos em aberto por servirem exatamente para diferenciar componentes concretos distintos. No caso do arcabouço EC2 apresentado nesta Tese, as categorias associadas aos modelos de instâncias, como CLUSTER-EC2-T3-LARGE, não possuem esse tipo de parâmetro de contexto, de modo que há um único componente concreto, e portanto um único perfil, associado à categoria. A Figura 47 ilustra a hierarquia de subtipos entre as categorias de plataformas virtuais para a infraestrutura EC2.

Na assinatura contextual de CLUSTER-EC2 (Figura 48), é acrescentado o parâmetro ***placement_group***, o qual define o tipo de grupo de alocação (*placement group*) da plataforma virtual [10]. Devido à natureza da própria HPC Shelf, optou-se por fixar o parâmetro *placement group* como de *clusterização* em todos os perfis associados aos modelos de instância. Apesar disso, decidimos mantê-lo na assinatura de CLUSTER-EC2, antecipando possíveis extensões onde *placement groups* de outros tipos possam ser úteis.

[10] <https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/placement-groups.html>

Figura 47 – Hierarquia de subtipos entre as categorias de plataformas virtuais para a infraestrutura EC2



Fonte: Próprio Autor

Figura 48 – Assinatura Contextual de NODE-EC2

$$\text{NODE-EC2}\ \begin{bmatrix} \textbf{\textit{count}} = 1, \\ \textbf{\textit{vcpu\_count}} :> 1 : \text{INT} \uparrow, \\ \textbf{\textit{storage}} = \text{STORAGE-EC2-INSTANCE}, \\ \textbf{\textit{cost\_per\_hour}} :> \text{REAL} \downarrow, \\ \textbf{\textit{network\_throughput}} :> 0 : \text{INT} \uparrow \end{bmatrix}\ \text{from NODE}$$

Fonte: Próprio Autor

Figura 49 – Assinatura Contextual de STORAGE-EC2-PERSISTENT

$$\text{STORAGE-EC2-PERSISTENT}\ [\textbf{\textit{cost\_per\_gigabyte}} :> \text{REAL} \downarrow]\ \text{from STORAGE}$$

Fonte: Próprio Autor

Além disso, são acrescentados os seguintes parâmetros a assinatura contextual NODE-EC2, derivada de NODE:

- ***vcpu_count***, o qual define a quantidade de vCPU de cada instância, uma vez que, mesmo com um processador com maior capacidade, somente é liberada uma quantidade pré-determinada de vCPUs para cada instância de máquina virtual, geralmente com mapeamento um-para-um com *hyper-threads* (SERVICES, 2019a).
- ***network_throughput***, o qual define a capacidade de comunicação em rede da instância, parâmetro essencial para cálculo de predição de tempo de execução, que é utilizada no processo de escolha de plataforma de execução.
- ***cost_per_hour***, o qual define o custo a ser cobrado por hora pelo uso de determinada instância. Esse valor é calculado diretamente no site do provedor IaaS e depende das configurações e recursos adicionais incluídos pelo mantenedor de plataformas nas instâncias.

Parâmetros que tem tarifas variáveis devem ser usados de maneira criteriosa, pois caso seja sub-provisionado, o custo de execução será maior que o valor recebido para executar e incorrerá prejuízo ao mantenedor, visto que ele está intermediando a contratação do serviço de nuvem IaaS. Um exemplo disso é o uso de instâncias de uso geral dos tipos T[11], que podem ter custos adicionais quando o processamento estiver acima de uma linha base de desempenho. Caso a aplicação a ser executada tenha uma característica de uso intensivo de CPU, ao final da execução, o custo terá sido em torno de 1,5 vezes o custo de uma instância de capacidade fixa de processamento. O mantenedor também pode fazer o reaproveitamento de uma instância, contratando uma instância do tipo reservada ou dedicada e disponibilizar para ser reutilizada quantas vezes forem possíveis dentro do período de contratação com o provedor de nuvem, de modo a poder maximizar seu lucro ou ofertar um serviço a custo reduzido em relação às instâncias sob demanda. Essa estratégia de compartilhamento também pode ser utilizada para reaproveitar períodos de ociosidade de instâncias de um mantenedor, a fim de reduzir os custos de execução de suas aplicações.

Finalmente, o parâmetro ***cost_per_gigabyte*** é introduzido na assinatura de STORAGE-EC2-PERSISTENT, apresentada na Figura 49, representando o custo por gigabyte (GB) de armazenamento.

No cálculo do parâmetro ***cost***, herdado da assinatura CLUSTER, além do custo individual das instâncias (***node-cost_per_hour***), são considerados os custos de todos serviços

[11] <https://aws.amazon.com/pt/ec2/instance-types/>

adicionais incluídos pelo mantenedor, como é o caso do armazenamento persistente independente de instância. O mantenedor de plataformas será responsável pela definição do valor a ser cobrado, podendo inclusive prever parâmetros que serão conhecidos somente em tempo de resolução do contrato da aplicação. No caso do armazenamento independente de instância (***storage***), usa-se o valor do parâmetro ***storage-cost*** multiplicado pela quantidade total ***storage-size***, a qual será especificada no respectivo parâmetro da aplicação. Caso ***storage-size*** não seja fornecido pelo contrato da aplicação, o valor padrão para o perfil, definido previamente pelo mantenedor de plataforma será utilizado. Deve-se ainda lembrar que o valor do parâmetro ***estimated_cost***, o qual representa o custo total da execução de um sistema computacional e encontra-se na assinatura do componente de computação do sistema, é calculado pela composição dos custos dos componentes de computação propriamente dito e o custo da plataforma virtual, dado pelo valor (argumento) calculado para o parâmetro ***cost*** da assinatura CLUSTER.

O parâmetro ***node-processor-count*** teve sua valoração numérica fixada com o valor 1, pois nos tipos de instâncias da EC2, não há conhecimento da quantidade de processadores efetivamente envolvidas na máquina virtual, do computador *host*. Assim, usa-se o produto dos valores dos parâmetros ***node-count*** e ***node-vcpu_count*** para calcular a quantidade de núcleos disponíveis na plataforma virtual.

O parâmetro ***storage*** foi fixado como um contrato do qualificador abstrato STORAGE-EC2-PERSISTENT, que define um tipo genérico de armazenamento não temporário, independente de ciclo de vida da máquina virtual. Há três subtipos para STORAGE-EC2-PERSISTENT, os quais permitem escolher um dos serviços disponíveis pela Amazon: STORAGE-S3 (*Amazon Simple Storage Service*[12]), STORAGE-EBS (*Amazon Elastic Block Store*[13]) e STORAGE-EFS (*Amazon Elastic File System*[14]).

O parâmetro ***interconnect*** define o tipo de rede de interconexão. No caso da Amazon EC2, os tipos de interconexão são próprios e derivados da assinatura INTERCONNECT-EC2, derivada de INTERCONNECT.

O parâmetro ***node*** é fixado pelo qualificador NODE-EC2, derivado de NODE, de cuja assinatura vem os parâmetros que descrevem um modelo de instância EC2. Na assinatura de NODE-EC2 o parâmetro ***storage***, proveniente de NODE, é fixado em STORAGE-EC2-INSTANCE, o qual descreve um tipo de armazenamento temporário em nível de bloco para a instância[15].

[12] <https://aws.amazon.com/s3>
[13] <https://aws.amazon.com/ebs>
[14] <https://aws.amazon.com/efs>
[15] <https://docs.aws.amazon.com/pt_br/AWSEC2/latest/UserGuide/InstanceStorage.html>

Figura 50 – Assinatura Contextual de CLUSTER-EC2-P3-2XLARGE

$$\begin{bmatrix} \textbf{\textit{name}} = \text{CLUSTER-EC2-P3-2XLARGE}, \\ \textbf{\textit{placement\_group}} :> \text{CLUSTERING}, \\ \textbf{\textit{virtualization\_type}} = \text{HVM}, \\ \textbf{\textit{node}} = \text{NODE-EC2-P3-2XLARGE}, \\ \textbf{\textit{storage-size}} = 50, \\ \textbf{\textit{storage-bandwidth}} = 1500 \end{bmatrix} \quad \begin{bmatrix} \textbf{\textit{name}} = \text{NODE-EC2-P3-2XLARGE}, \\ \textbf{\textit{vcpu\_count}} = 8, \\ \textbf{\textit{processor}} = \text{PROCESSORINTELXEON\_2686V4}, \\ \textbf{\textit{accelerator}} = \text{NVIDIA\_V100}, \\ \textbf{\textit{accelerator-count}} = 1, \\ \textbf{\textit{storage-size}} = 128, \\ \textbf{\textit{operating\_system}} = \text{LINUX} \\ \textbf{\textit{memory-size}} = 61, \\ \textbf{\textit{cost\_per\_hour}} = 3.06 \end{bmatrix}$$

Fonte: Próprio Autor

Portanto, para as plataformas virtuais EC2 desse modelo, o tempo de vida do armazenamento dos nós de processamento vale apenas para enquanto a instância estiver ativa, ao contrário do armazenamento global do cluster, que é persistente por meio de um dos serviços S3, EBS e EFS. Assim como para CLUSTER-EC2, componentes abstratos para tipos e modelos de instâncias EC2 também são criados como subtipos de NODE-EC2 (e.g. NODE-EC2-T3-LARGE).

A Figura 50 apresenta, à esquerda, o perfil de plataformas virtuais CLUSTER-EC2-P3-2XLARGE, a partir do qual são criadas instâncias do modelo *p3.2xlarge*. À direita da mesma figura, de forma a complementar a definição de CLUSTER-EC2-P3-2XLARGE, é apresentado o contrato NODE-EC2-P3-2XLARGE, que define as propriedades dos nós (instâncias EC2) do cluster. Nota-se que alguns parâmetros de contexto permanecem em abertos, como o tipo da interconexão (***network***) e o tipo de armazenamento global persistente (***storage***). Em uma possível implementação, admite-se que os valores desses parâmetros podem ser escolhidos explicitamente, pela aplicação, ou implicitamente, pelo mantenedor, quando a aplicação não determina o uso de um serviço específico (por exemplo, quando o perfil CLUSTER-EC2-P3-2XLARGE é escolhido a partir de um contrato de CLUSTER). Em uma implementação alternativa, poderiam ser registrados perfis para todas as escolhas possíveis de ***network*** e ***storage***, ficando a cargo do sistema de resolução de contratos, quando não há escolha explícita de valores para esses parâmetros por parte da aplicação, a escolha do melhor perfil de acordo com os critérios de qualidade e custo definidos pela aplicação. Nesse caso, CLUSTER-EC2-P3-2XLARGE seria uma assinatura contextual de tais perfis alternativos.

As tabelas de 7 até 13 apresentam argumentos de contexto que definem os perfis para os modelos de instâncias do serviço EC2, de acordo com as características publicadas em sua documentação.

Tabela 7 – Tabela com enquadramento dos tipos de instâncias de uso geral da Amazon EC2 t3, t3a, e m5

name	*vcpu_count*	*processor*	*processor-cache_L3-size (MiB)*	*processor-core-cache_L2-size (MiB)*	*processor-core-cache_L1-size (MiB)*	*processor-core-clock (GHz)*	*processor-family*	*processor-manufacturer*	*processor-microarchitecture*	*memory-size (GiB)*	*storage*	*storage-size(GiB)*	*storage-bandwidth (Gbps)*	*operating_system*	*cost_per_hour (USD)*	*network_throughput (Gbit/s)*
t3.nano	2	Skylake E5 2686 v5	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	6	EC2-SSD	0,5		AWS-AMI	0,0052	5*
t3.micro	2	Skylake E5 2686 v5	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	12	EC2-SSD	1		AWS-AMI	0,0104	5*
t3.small	2	Skylake E5 2686 v5	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	24	EC2-SSD	2		AWS-AMI	0,0208	5*
t3.medium	2	Skylake E5 2686 v5	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	24	EC2-SSD	4		AWS-AMI	0,0416	5*
t3.large	2	Skylake E5 2686 v5	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	36	EC2-SSD	8		AWS-AMI	0,0832	5*
t3.xlarge	4	Skylake E5 2686 v5	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	96	EC2-SSD	16		AWS-AMI	0,1664	5*
t3.2xlarge	8	Skylake E5 2686 v5	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	192	EC2-SSD	32		AWS-AMI	0,3328	5*
t3a.nano	2	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	0,5	EC2-EBS	128*		AWS-AMI	0,0047	5*
t3a.micro	2	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	1	EC2-EBS	128*		AWS-AMI	0,0094	5*
t3a.small	2	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	2	EC2-EBS	128*		AWS-AMI	0,0188	5*
t3a.medium	2	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	4	EC2-EBS	128*		AWS-AMI	0,0376	5*
t3a.large	2	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	8	EC2-EBS	128*		AWS-AMI	0,0752	5*
t3a.xlarge	4	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	16	EC2-EBS	128*		AWS-AMI	0,1504	5*
t3a.2xlarge	8	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	32	EC2-EBS	128*		AWS-AMI	0,3008	5*
m5.large	2	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	8	EC2-EBS	128*	3.500	AWS-AMI	0,096	10
m5.xlarge	4	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	16	EC2-EBS	128*	3.500	AWS-AMI	0,192	10
m5.2xlarge	8	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	32	EC2-EBS	128*	3.500	AWS-AMI	0,384	10
m5.4xlarge	16	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	64	EC2-EBS	128*	3.500	AWS-AMI	0,768	10
m5.12xlarge	48	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	192	EC2-EBS	128*	7.000	AWS-AMI	2,304	10
m5.24xlarge	96	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	384	EC2-EBS	128*	14.000	AWS-AMI	4,608	25
m5.metal	96	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	384	EC2-EBS	128	14.000	AWS-AMI	4,608	25

Fonte: Próprio Autor

Tabela 8 – Tabela com enquadramento das instâncias de uso geral da Amazon EC2, dos tipos de instâncias m5d à m5ad

name	*vcpu_count*	*processor*	*processor-cache_L3-size (MiB)*	*processor-core-cache_L2-size (MiB)*	*processor-core-cache_L1-size (MiB)*	*processor-core-clock (GHz)*	*processor-family*	*processor-manufacturer*	*processor-microarchitecture*	*memory-size (GiB)*	*storage*	*storage-size(GiB)*	*storage-bandwidth (Gbps)*	*operating_system*	*cost_per_hour (USD)*	*network_throughput (Gbit/s)*
m5d.large	2	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	8	EC2-SSD	75		AWS-AMI	0,113	10
m5d.xlarge	4	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	16	EC2-SSD	150		AWS-AMI	0,226	10
m5d.2xlarge	8	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	32	EC2-SSD	300		AWS-AMI	0,452	10
m5d.4xlarge	16	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	64	EC2-SSD	600		AWS-AMI	0,904	10
m5d.12xlarge	48	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	192	EC2-SSD	1800		AWS-AMI	2,712	10
m5d.24xlarge	96	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	384	EC2-SSD	3600		AWS-AMI	5,424	25
m5d.metal	96	Xeon® Platinum 8175	33	24	1,50	3,1	Platinum	Intel	Skylake	384	EC2-SSD	3600		AWS-AMI	5,424	25
m5a.large	2	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	8	EC2-EBS	128*	2.120	AWS-AMI	0,086	10
m5a.xlarge	4	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	16	EC2-EBS	128*	2.120	AWS-AMI	0,172	10
m5a.2xlarge	8	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	32	EC2-EBS	128*	2.120	AWS-AMI	0,344	10
m5a.4xlarge	16	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	64	EC2-EBS	128*	2.120	AWS-AMI	0,688	10
m5a.12xlarge	48	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	192	EC2-EBS	128*	5.000	AWS-AMI	2,064	10
m5a.24xlarge	96	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	384	EC2-EBS	128*	10.000	AWS-AMI	4,128	20
m5ad.large	2	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	8	EC2-SSD	75		AWS-AMI	0,103	10
m5ad.xlarge	4	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	16	EC2-SSD	150		AWS-AMI	0,206	10
m5ad.2xlarge	8	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	32	EC2-SSD	300		AWS-AMI	0,412	10
m5ad.4xlarge	16	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	64	EC2-SSD	600		AWS-AMI	0,824	10
m5ad.12xlarge	48	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	192	EC2-SSD	1800		AWS-AMI	2,472	10
m5ad.24xlarge	96	AMD EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	384	EC1-SSD	3600		AWS-AMI	4,944	20

Fonte: Próprio Autor

Tabela 9 – Tabela com enquadramento das instâncias otimizadas para computação da Amazon EC2

name	*vcpu_count*	*processor*	*processor-cache_L3.size*	*processor-core-cache_L2-size (MiB)*	*processor-core-cache_L1-size (MiB)*	*processor-core-clock (GHz)*	*processor-family*	*processor-manufacturer*	*processor-microarchitecture*	*memory-size (GiB)*	*storage*	*storage-size(GiB)*	*storage-bandwidth (Gbps)*	*operating_system*	*cost_per_hour (USD)*	*network_throughput (Gbit/s)*
c5.large	2	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	4	EC2-EBS	128*	3500	AWS-AMI	0,085	10
c5.xlarge	4	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	8	EC2-EBS	128*	3500	AWS-AMI	0,17	10
c5.2xlarge	8	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	16	EC2-EBS	128*	3500	AWS-AMI	0,34	10
c5.4xlarge	16	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	32	EC2-EBS	128*	3500	AWS-AMI	0,68	10
c5.9xlarge	36	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	72	EC2-EBS	128*	7000	AWS-AMI	1,53	10
c5.18xlarge	72	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	144	EC2-EBS	128*	14000	AWS-AMI	3,06	25
c5d.large	2	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	4	EC2-SSD	50	3500	AWS-AMI	0,096	10
c5d.xlarge	4	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	8	EC2-SSD	100	3500	AWS-AMI	0,192	10
c5d.2xlarge	8	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	16	EC2-SSD	200	3500	AWS-AMI	0,384	10
c5d.4xlarge	16	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	32	EC2-SSD	400	3500	AWS-AMI	0,768	10
c5d.9xlarge	36	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	72	EC2-SSD	900	7000	AWS-AMI	1,728	10
c5d.18xlarge	72	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	144	EC2-SSD	1800	14000	AWS-AMI	3,456	25
c5n.large	2	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	5.25	EC2-EBS	128*	3500	AWS-AMI	0,108	25
c5n.xlarge	4	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	10.5	EC2-EBS	128*	3500	AWS-AMI	0,216	25
c5n.2xlarge	8	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	21	EC2-EBS	128*	3500	AWS-AMI	0,432	25
c5n.4xlarge	16	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	42	EC2-EBS	128*	3500	AWS-AMI	0,864	25
c5n.9xlarge	36	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	96	EC2-EBS	128*	7000	AWS-AMI	1,944	50
c5n.18xlarge	72	Platinum 8124M	24,75	18	1,13	3	Platinum	Intel	Skylake	192	EC2-EBS	128*	14000	AWS-AMI	3,888	100

Fonte: Próprio Autor

Tabela 10 – Tabela com enquadramento dos tipos de instâncias r5, r5d e r5a, otimizadas para memória da Amazon EC2

name	*vcpu_count*	*processor*	*processor-cache_L3.size*	*processor-core-cache_L2-size (MiB)*	*processor-core-cache_L1-size (MiB)*	*processor-core-clock (GHz)*	*processor-family*	*processor-manufacturer*	*processor-microarchitecture*	*memory-size (GiB)*	*storage*	*storage-size(GiB)*	*storage-bandwidth (Gbps)*	*operating_system*	*cost_per_hour (USD)*	*network_throughput (Gbit/s)*
r5.large	2	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	16	EC2-EBS	128*	3500	AWS-AMI	0,126	10
r5.xlarge	4	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	32	EC2-EBS	128*	3500	AWS-AMI	0,252	10
r5.2xlarge	8	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	64	EC2-EBS	128*	3500	AWS-AMI	0,504	10
r5.4xlarge	16	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	128	EC2-EBS	128*	3500	AWS-AMI	1,008	10
r5.12xlarge	48	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	384	EC2-EBS	128*	7000	AWS-AMI	3,024	10
r5.24xlarge	96	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	768	EC2-EBS	128*	14000	AWS-AMI	6,048	25
r5.metal	96	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	768	EC2-EBS	128*	14000	AWS-AMI	6,048	25
r5d.large	2	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	16	EC2-SSD	75	3500	AWS-AMI	0,144	10
r5d.xlarge	4	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	32	EC2-SSD	150	3500	AWS-AMI	0,288	10
r5d.2xlarge	8	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	64	EC2-SSD	300	3500	AWS-AMI	0,576	10
r5d.4xlarge	16	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	128	EC2-SSD	600	3500	AWS-AMI	1,152	10
r5d.12xlarge	48	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	384	EC2-SSD	1800	7000	AWS-AMI	3,456	10
r5d.24xlarge	96	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	768	EC2-SSD	3600	14000	AWS-AMI	6,912	25
r5d.metal	96	Platinum 8175	33	24	1,5	3,1	Platinum	Intel	Skylake	768	EC2-SSD	3600	14000	AWS-AMI	6,912	25
r5a.large	2	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	16	EC2-EBS	128*	2120	AWS-AMI	0,113	10
r5a.xlarge	4	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	32	EC2-EBS	128*	2120	AWS-AMI	0,226	10
r5a.2xlarge	8	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	64	EC2-EBS	128*	2120	AWS-AMI	0,452	10
r5a.4xlarge	16	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	128	EC2-EBS	128*	2120	AWS-AMI	0,904	10
r5a.12xlarge	48	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	384	EC2-EBS	128*	5000	AWS-AMI	2,712	10
r5a.24xlarge	96	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	768	EC2-EBS	128*	10000	AWS-AMI	5,424	20
r5ad.large	2	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	16	EC2-SSD	75	Até 2.120	AWS-AMI	0,131	10
r5ad.xlarge	4	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	32	EC2-SSD	150	Até 2.120	AWS-AMI	0,262	10
r5ad.2xlarge	8	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	64	EC2-SSD	300	Até 2.120	AWS-AMI	0,524	10
r5ad.4xlarge	16	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	128	EC2-SSD	600	2120	AWS-AMI	1,048	10
r5ad.12xlarge	48	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	384	EC2-SSD	1800	5000	AWS-AMI	3,144	10
r5ad.24xlarge	96	EPYC 7571	8	0,5	0,09	2,5	Zen	AMD	Epyc	768	EC2-SSD	3600	10000	AWS-AMI	6,288	20

Fonte: Próprio Autor

Tabela 11 – Tabela com enquadramento dos tipos de instâncias r4 e x1e, otimizadas para memória da Amazon EC2

name	*vcpu_count*	*processor*	*processor-cache_L3-size (MiB)*	*processor-core-cache_L2-size (MiB)*	*processor-core-cache_L1-size (MiB)*	*processor-core-clock (GHz)*	*processor-family*	*processor-manufacturer*	*processor-microarchitecture*	*memory-size (GiB)*	*storage*	*storage-size(GiB)*	*storage-bandwidth (Gbps)*	*operating_system*	*cost_per_hour (USD)*	*network_throughput (Gbit/s)*
r4.large	2	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	15,25	EC2-EBS	128*		AWS-AMI	0,133	10*
r4.xlarge	4	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	30,5	EC2-EBS	128*		AWS-AMI	0,266	10*
r4.2xlarge	8	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	61	EC2-EBS	128*		AWS-AMI	0,532	10*
r4.4xlarge	16	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	122	EC2-EBS	128*		AWS-AMI	1	10*
r4.8xlarge	32	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	244	EC2-EBS	128*		AWS-AMI	2	10
r4.16xlarge	64	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	488	EC2-EBS	128*		AWS-AMI	4	25
x1e.xlarge	4	Xeon E7-8880 v3	45	4,5	1,13	2,3	E7-8880 v3	Intel	Haswell	122	EC2-SSD	120		AWS-AMI	0,834	10*
x1e.2xlarge	8	Xeon E7-8880 v3	45	4,5	1,13	2,3	E7-8880 v3	Intel	Haswell	244	EC2-SSD	240		AWS-AMI	1,668	10*
x1e.4xlarge	16	Xeon E7-8880 v3	45	4,5	1,13	2,3	E7-8880 v3	Intel	Haswell	488	EC2-SSD	480		AWS-AMI	3,336	10*
x1e.8xlarge	32	Xeon E7-8880 v3	45	4,5	1,13	2,3	E7-8880 v3	Intel	Haswell	976	EC2-SSD	960		AWS-AMI	6,672	10*
x1e.16xlarge	64	Xeon E7-8880 v3	45	4,5	1,13	2,3	E7-8880 v3	Intel	Haswell	1952	EC2-SSD	1920		AWS-AMI	13,344	10
x1e.32xlarge	128	Xeon E7-8880 v3	45	4,5	1,13	2,3	E7-8880 v3	Intel	Haswell	3904	EC2-SSD	3840		AWS-AMI	26,688	25
x1.16xlarge	64	Xeon E7-8880 v3	45	4,5	1,13	2,3	E7-8880 v3	Intel	Haswell	976	EC2-SSD	1920		AWS-AMI	6,669	10

Fonte: Próprio Autor

Tabela 12 – Tabela com enquadramento das instâncias aceleradas da Amazon EC2

name	*vcpu_count*	*processor*	*processor-cache_L3-size (MiB)*	*processor-core-cache_L2-size (MiB)*	*processor-core-cache_L1-size (MiB)*	*processor-core-clock (GHz)*	*processor-family*	*processor-manufacturer*	*processor-microarchitecture*	*memory-size (GiB)*	*storage*	*storage-size(GiB)*	*storage-bandwidth (Gbps)*	*operating_system*	*cost_per_hour (USD)*	*network_throughput (Gbit/s)*
p3.2xlarge	8	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	61	EC2-EBS	128*	1500	AWS-AMI	3,06	10
p3.8xlarge	32	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	244	EC2-EBS	128*	7000	AWS-AMI	12,24	10
p3.16xlarge	64	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	488	EC2-EBS	128*	14000	AWS-AMI	24,48	25
p2.8xlarge	32	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	488	EC2-EBS	128*		AWS-AMI	7,2	10
p2.16xlarge	64	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	732	EC2-EBS	128*		AWS-AMI	14,4	25
g3s.xlarge	4	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	30,5	EC2-EBS	128*		AWS-AMI	1,14	10*
g3.4xlarge	16	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	122	EC2-EBS	128*		AWS-AMI	2,28	10*
g3.8xlarge	32	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	244	EC2-EBS	128*		AWS-AMI	4,56	10
g3.16xlarge	64	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	488	EC2-EBS	128*		AWS-AMI	0,75	25
f1.2xlarge	8	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	122	EC2-SSD	470		AWS-AMI	1,65	10*
f1.4xlarge	16	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	244	EC2-SSD	940		AWS-AMI	3,3	10*
f1.16xlarge	64	E5 2686 v4	45	4,5	1,13	2,5	E5-2686 v4	Intel	Broadwell	976	EC2-SSD	3760		AWS-AMI	13,2	25

Fonte: Próprio Autor

Tabela 13 – Tabela com parâmetros específicos das instâncias aceleradas da Amazon EC2

name	*accelerator-count*	*accelerator*	*accelerator-model*	*accelerator-memory_size*	*accelerator-manufacturer*
p3.2xlarge	1	NVIDIA Tesla V100	V100	16	NVIDIA
p3.8xlarge	4	NVIDIA Tesla V100	V100	64	NVIDIA
p3.16xlarge	8	NVIDIA Tesla V100	V100	128	NVIDIA
p2.8xlarge	8	NVIDIA K80	K80	96	NVIDIA
p2.16xlarge	16	NVIDIA K80	K80	192	NVIDIA
g3s.xlarge	1	NVIDIA Tesla M60	M60	8	NVIDIA
g3.4xlarge	1	NVIDIA Tesla M61	M60	8	NVIDIA
g3.8xlarge	2	NVIDIA Tesla M62	M60	16	NVIDIA
g3.16xlarge	4	NVIDIA Tesla M63	M60	32	NVIDIA
f1.2xlarge	1	Virtex UltraScale+ VU9P	VU9P		Xilinx
f1.4xlarge	2	Virtex UltraScale+ VU9P	VU9P		Xilinx
f1.16xlarge	8	Virtex UltraScale+ VU9P	VU9P		Xilinx

Fonte: Próprio Autor

5.3 Estudo de Caso: Multiplicação de Matrizes Baseado em BLAS

O estudo de caso para validação do Alite é o algoritmo de multiplicação de matrizes, uma operação comumente utilizada em bibliotecas de álgebra linear, e que são utilizadas em uma diversidade de aplicações científicas. Na implementação dessas bibliotecas, destacam-se os projetos C-XSC[16] (***C*** *– e****X****tension for* ***S****cientific* ***C****omputation*), que tem o foco voltado para a precisão e validação dos cálculos, e a biblioteca BLAS (***B****asic* ***L****inear* ***A****lgebra* ***S****ubprograms*)[17] e suas variantes, considerada um padrão "de facto", possuindo ênfase em desempenho e rotinas definidas por grupos funcionais divididos em três níveis. No nível 1, encontram-se as operações vetor-vetor. Por sua vez, no nível 2, são definidas as operações matriz-vetor. Finalmente, no nível 3, as operações entre matrizes. Foi escolhida a biblioteca BLAS para ter sua multiplicação de matrizes validada, devido ao compartilhamento do interesse em desempenho com a HPC Shelf.

Existem diversas implementações da biblioteca BLAS atendendo a diferentes plataformas de execução, tipos de dados ou formatos de matrizes, de modo a explorar, de forma otimizada, os diversos padrões de paralelismo (DONGARRA *et al.*, 1990; KOç; GAN, 1992). Dentre esses padrões, destacam-se as arquiteturas de memória compartilhada, de memória distribuída e de processamento vetorial, buscando sempre alcançar o máximo poder de processamento oferecido pela máquina alvo.

Para arquiteturas de memória compartilhada, as implementações buscam otimizar o uso das hierarquias de memória (memórias *cache* e principal) e recursos dos processadores como o uso de TLB (*Translation Lookaside Buffer*) para suporte à memória virtual e registradores para operações de coprocessamento aritmético. Essas otimizações desenvolvidas nas implementações específicas visam principalmente parametrizar os algoritmos de forma a favorecer os efeitos das hierarquias de memórias, com o correto dimensionamento e a escolha de qual nível de memória *cache* irá armazenar cada dado, podendo ainda otimizar o reuso dos dados, como ocorre de maneira explícita na biblioteca BLIS (***B****LAS-like* ***L****ibrary* ***I****nstantiation* ***S****oftware*), GotoBLAS e OpenBLAS, ou, de maneira implícita, na biblioteca ATLAS (LOW *et al.*, 2016). O pacote LAPACK (***L****inear* ***A****lgebra* ***Pack****age*) oferece implementações de resolvedores sistemas de equações lineares, sobre implementações conhecidas do BLAS. Existe ainda uma implementação genérica, implementada na linguagem Fortran, que pode ser compilada para a plataforma alvo. Por sua generalidade, essa implementação apresenta desempenho limitado em relação às implementações

[16] <http://www2.math.uni-wuppertal.de/wrswt/xsc/cxsc/apidoc/html/index.html>
[17] <http://www.netlib.org/blas/>

otimizadas disponibilizadas pelos fabricantes dos processadores e coprocessadores (GOTO; GEIJN, 2008). Já as implementações para arquiteturas SIMD, onde destaca-se o processamento vetorial, os padrões de acesso e particionamento são definidos de acordo com a arquitetura (LOW *et al.*, 2016). Destaca-se a biblioteca MAGMA (*Matrix Algebra on GPU and Multicore Arquitectures*), que executa sobre a CUBLAS, implementação do BLAS desenvolvida pela NVIDIA para executar de forma otimizada nas suas GPUs (BARRACHINA *et al.*, 2008).

Para aplicações que executam em plataformas de memória distribuída, as matrizes de entrada são particionadas em blocos que são alocados nas unidades de processamento. Uma das implementações mais disseminadas é o ScaLAPACK (***Sca****lable* ***LAPACK***), que, a partir do LAPACK, promove os mecanismos de alto nível chamados *drivers* para implementação MIMD de solução de sistemas lineares. O ScaLAPACK utiliza o BLACS (***B****asic* ***L****inear* ***A****lgebra* ***C****ommunication* ***S****ubprograms*) para comunicação eficiente e o PBLAS (***P****arallel* ***B****asic* ***L****inear* ***A****lgebra* ***S****ubprograms*), que por sua vez utiliza uma implementação do BLAS para as subrotinas, principalmente nos níveis 1 e 2. A comunicação através do BLACS permite suporte às principais bibliotecas de memória distribuída, como MPI e PVM (CHOI *et al.*, 1996).

De acordo com a definição da interface BLAS, o componente de multiplicação de matrizes pode tratar matrizes gerais, simétricas, triangulares e hermitianas, sendo possível que as células das matrizes sejam números de ponto flutuantes de precisão simples ou dupla, representando números reais ou complexos. Além de Fortran, existe uma diversidade de implementações do BLAS em outras linguagens (DONGARRA *et al.*, 1990).

Na HPC Shelf, o objetivo desse estudo de caso é explorar o mecanismo de escolha de componentes, bem como o resultado na variação dos contratos a serem resolvidos. Nesse caso, tem-se basicamente a estrutura de dados para representar as matrizes de entrada e de saída, bem como o componente que efetivamente executa a computação.

A assinatura de um componente abstrato chamado BLAS3MM, cujos componentes concretos estão associados a cada uma das subrotinas de multiplicação de matrizes da interface BLAS, é apresentada na Figura 51. A Tabela 14, lista tais subrotinas representa os parâmetros de contexto (colunas) que permitem a seleção dentre elas. Dentre os componentes concretos de BLAS3MM, optou-se por modelar o desempenho de uma versão paralela do DGEMM (multiplicação de matrizes genéricas reais de ponto flutuante de precisão dupla).

Na Figura 52, está representado o contrato contextual de um componente concreto hipotético de BLAS3MM implementado usando a subrotina DGEMM, contendo os argumentos

Tabela 14 – Resumo dos componentes de multiplicação de matrizes do BLAS definidos por Dongarra *et al.* (1990)

matrix_type	***domain***	***float_point_precision***	*subroutine name*
GENERALMATRIX	REAL	SINGLE	**SGEMM**
	REAL	DOUBLE	**DGEMM**
	COMPLEX	SINGLE	**CGEMM**
	COMPLEX	DOUBLE	**ZGEMM**
SYMMETRICMATRIX	REAL	SINGLE	**SSYMM**
	REAL	DOUBLE	**DSYMM**
	COMPLEX	SINGLE	**CSYMM**
	COMPLEX	DOUBLE	**ZSYMM**
TRIANGULARMATRIX	REAL	SINGLE	**STRMM**
	COMPLEX	SINGLE	**CTRMM**
	COMPLEX	DOUBLE	**ZTRMM**
HERMITIANMATRIX	COMPLEX	SINGLE	**CHEMM**
	COMPLEX	DOUBLE	**ZHEMM**

Fonte: Próprio Autor

Figura 51 – Assinatura Contextual do Componente Abstrato BLAS3MM

$$\text{BLAS3MM} \left[\begin{array}{l} \textbf{\textit{k\_size}} <: \text{INT} \downarrow, \\ \textbf{\textit{n\_size}} <: \text{INT} \downarrow, \\ \textbf{\textit{m\_size}} <: \text{INT} \downarrow, \\ \textbf{\textit{matrix\_type}} <: \text{MATRIXTYPE}, \\ \textbf{\textit{domain}} <: \text{DOMAINTYPE}, \\ \textbf{\textit{precision}} <: \text{PRECISIONTYPE}, \\ \textbf{\textit{platform}} <: \text{CLUSTER} \\ \textbf{\textit{flop\_count}} :> \text{INT} \uparrow, \\ \textbf{\textit{time}} :> 0{:}\text{INT} \downarrow, \\ \textbf{\textit{efficiency}} :> 0{:}\text{INT} \uparrow, \\ \textbf{\textit{speedup}} :> 1{:}\text{INT} \uparrow, \\ \textbf{\textit{power}} :> 0{:}\text{INT} \downarrow, \\ \textbf{\textit{cost}} :> 0{:}\text{INT} \downarrow \end{array}\right]$$

Fonte: Próprio Autor

Figura 52 – Contrato Contextual de BLAS3MM

$$\text{BLAS3MM} \begin{bmatrix} \textbf{\textit{matrix\_pattern}} = \text{GENERICMATRIX}, \\ \textbf{\textit{domain}} = \text{REAL}, \\ \textbf{\textit{precision}} = \text{DOUBLE}, \\ \textbf{\textit{platform}} = \text{CLUSTER} \end{bmatrix}$$

Fonte: Próprio Autor

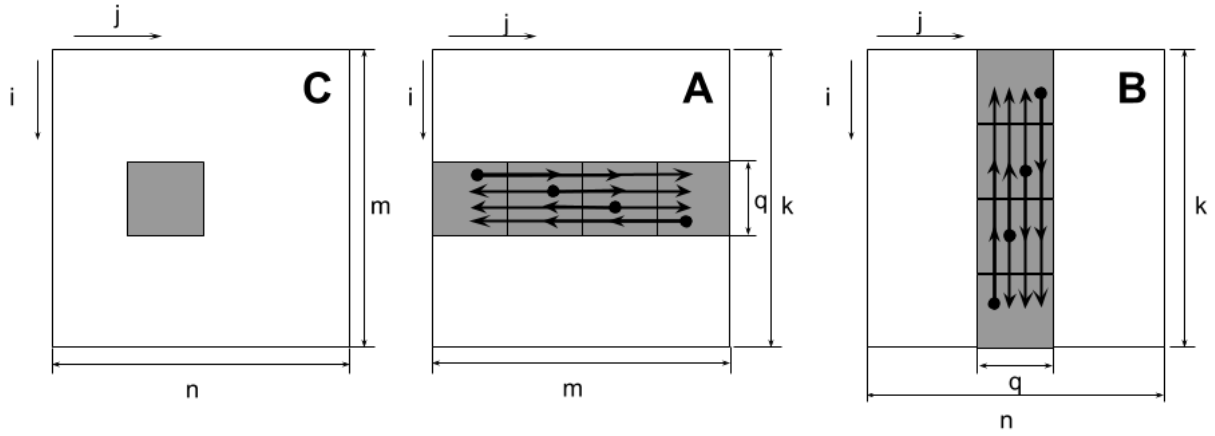
Figura 53 – Contrato Contextual de BLAS3MM

$$\text{BLAS3MM} \begin{bmatrix} \textbf{\textit{k\_size}} = 10^3, \\ \textbf{\textit{n\_size}} = 10^3, \\ \textbf{\textit{m\_size}} = 10^3, \\ \textbf{\textit{matrix\_pattern}} = \text{GENERICMATRIX}, \\ \textbf{\textit{domain}} = \text{REAL}, \\ \textbf{\textit{precision}} = \text{DOUBLE}, \\ \textbf{\textit{platform}} = \text{CLUSTER}, \\ \textbf{\textit{platform-node-accelerator}} = \text{GPU-TESLA-V100}, \\ \textbf{\textit{platform-node-accelerator.count}} = 1+ \\ \textbf{\textit{efficiency}} = 0.7, \\ \textbf{\textit{cost}} = 10 \end{bmatrix}$$

Fonte: Próprio Autor

de contexto suficientes para distingui-lo das demais subrotinas do BLAS, porém sem fazer maiores suposições a respeito da plataforma de execução, dimensões das matrizes de entrada ou compromissos de qualidade e custo. Por sua vez, a Figura 53 apresenta um contrato de BLAS3MM para ser submetido ao algoritmo de resolução, restringindo que os candidatos deverão executar uma multiplicação de matrizes reais arbitrárias de números de ponto flutuante de dupla precisão, de dimensões da ordem de 10^3, usando como plataforma de execução um *cluster* cujos nós de processamento sejam dotados de uma GPU do modelo Tesla V100, da fabricante NVIDIA. Além disso, requer eficiência de pelo menos 70%, enquanto o custo de execução não deve ultrapassar 10 unidades monetárias. O componente concreto que implementa o contrato contextual da Figura 52 poderia ser escolhido nesse contexto, caso um cluster com GPUs compatíveis não esteja disponível, graças a estratégia de generalização de contratos do algoritmo de resolução, desde que seu modelo de desempenho determine uma eficiência igual ou maior a 70% e custo menor ou igual a 10 unidades monetárias.

Figura 54 – Particionamento das Matrizes A, B e C

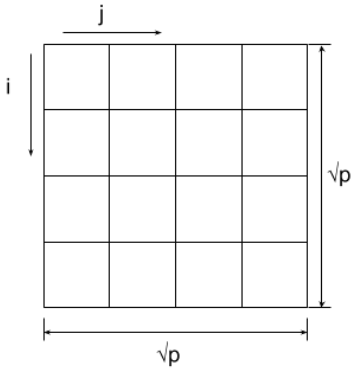


Fonte: Próprio Autor

5.3.1 Parâmetros de Qualidade

Na Seção 4.1.8, os parâmetros de contexto de qualidade foram apresentados para permitir que as aplicações façam suposições a respeito de características de desempenho dos componentes de computação que utiliza em seus sistemas de computação paralela. Além disso, tem papel na classificação dos componentes sistema candidatos. Os parâmetros de qualidade podem também ser utilizados por mecanismos de elasticidade, como o mecanismo proposto por Alencar (2017) em sua Tese de Doutorado, dentro do contexto da HPC Shelf (ALENCAR; Carvalho Junior, 2019). Assume-se que uma uma implementação da HPC Shelf oferece um conjunto de parâmetros de qualidade básicos a serem suportados em componentes de computação, o qual pode evoluir com o passar do tempo, podendo incluir novos parâmetros a fim de refletir, por exemplo, avanços tecnológicos no estado-da-arte.

Figura 55 – Grade dos p Nós de Processamento



Fonte: Próprio Autor

Nesta seção, são apresentados parâmetros de qualidade para este estudo de caso: a aceleração (*speedup*), a eficiência e o tempo de execução. Como discutido na Seção 4.1.8, parâmetros de qualidade podem ser calculados durante a resolução dos contratos, a partir dos parâmetros que descrevem as propriedades do componente de computação (parâmetros de aplicação) e da plataforma virtual (parâmetros de plataforma) que formam o componente sistema. Por esse motivo, no contrato implementado por componentes de computação, parâmetros de qualidade são associados a funções de estimativa, que calculam os argumentos necessários durante o processo de resolução. Quando essas funções não estão presentes, os valores *default* para os parâmetros são utilizados. Por exemplo, para estimativas de aceleração, eficiência e tempo de execução, esses valores são 1, 0 e $+\infty$, respectivamente, de modo que tais componentes tendem a ter classificação penalizada em relação a componentes que fornecem funções de estimativas, ainda que conservadoras. Além disso, um mecanismo de reputação poderia garantir que componentes que não cumprem os argumentos de qualidade expressos em seus contratos (SLOs) sofram uma tendência a serem rebaixados na classificação.

Para o cálculo de estimativas de aceleração (S), eficiência (E) e tempo de execução (T_p), é fundamental um modelo de desempenho que permita o cálculo de dois valores: o tempo de execução sequencial sobre um nó de processamento da plataforma virtual (T_1) e o tempo de sobrecarga da execução paralela (T_o). Tais valores dependem do conhecimento do desenvolvedor do componente acerca de detalhes do algoritmo utilizado pelo componente e sua implementação utilizando um certo artefato de programação paralela, bem como de propriedades da plataforma virtual expostas na assinatura do componente abstrato CLUSTER. Para o cálculo de T_1, são necessários os argumentos que permitem calcular o tamanho da computação a ser realizada, bem como os argumentos de perfis de plataformas virtuais que descrevem propriedades do sistema de memória, possivelmente envolvendo todos os níveis da hierarquia, e dos processadores e aceleradores empregados nos nós de processamento. Por sua vez, T_o, o tempo de sobrecarga do paralelismo sobre a plataforma virtual, pode ter várias fontes, sendo os mais importantes: o desbalanceamento da carga de trabalho entre os nós de processamento, levando a tempos de ociosidade dos nós da plataforma virtual; o tempo gasto em operações de comunicação e sincronização entre processos; e o tempo de execução de trechos não-paralelizáveis do código. Portanto, as propriedades relacionadas à interconexão entre os nós de processamento da plataforma virtual são especialmente necessárias para o cálculo de T_o.

De posse dos valores de T_1 e T_o, o valor de T_p pode ser estimado por $T_p = T_o + \frac{T_1}{p}$,

onde p corresponde ao número de nós de processamento.

Supõe-se que é de interesse e responsabilidade do desenvolvedor de componentes incluir parâmetros de contexto que permitam calcular a quantidade de computação que será executada, essencial para o cálculo de T_1. No caso do componente BLAS3MM, os parâmetros que permitem são ***k_size***, ***m_size*** e ***n_size***. Quando não presentes, há impossibilidade de calcular T_1 e, portanto, as funções de estimativa para S, E e T_p. Quando presentes mas não associadas a valores por aplicações em sistemas de computação paralela antes de executar a solução de um problema, possivelmente pela impossibilidade de determinar estaticamente tais valores, a função de estimativa tende a calcular argumentos que levam a escolhas arbitrárias de componentes sistema em relação aos parâmetros de qualidade em questão. Por exemplo, sem levar em consideração restrições de aceleração, eficiência e/ou tempo de execução.

Uma outra forma, mais otimista e portanto menos realística, de estimar o valor de T_1 é utilizando a razão entre a quantidade de operações relevantes a serem realizada pelo algoritmo sequencial paralelizado W por C, um valor para a capacidade de processamento dessas operações em um nó da plataforma virtual. Dessa forma, $T_1 = \frac{W}{C}$. C pode ser tomado tanto como a vazão de execução das operações em questão teórica quanto como aquela obtida através de benchmarks. Por exemplo, em se tratando de operações de ponto flutuante, o HPL ou HPCG, utilizados pelo Top500, constituem alternativas aceitas perante a comunidade. Entretanto, a métrica adequada é bastante dependente da natureza da computação executada pelo componente. Por exemplo, é comum que o tempo de execução de uma computação não seja restringida pelo número de operações aritméticas, inteira ou ponto flutuante, mas pelos acessos à memória, mesmo em computação científica.

A aceleração (S) representa a relação entre o tempo de execução sequencial e o tempo de execução paralela, sobre p nós de processamento, ou seja $S = \frac{T_1}{T_p} = \frac{pT_1}{T_1+T_o}$. Logo, o resultado esperado está no intervalo $[1, p]$. Em casos excepcionais, a aceleração pode estar fora desse intervalo. No caso $S < 1$, mais comum, não há ganho em termos de tempo de execução paralela em relação a execução sequencial, devido a alta sobrecarga (T_o). Por sua vez, no caso $S > p$, há um fenômeno conhecido como super-speedup, situação quando a sobrecarga da execução paralela é negativa, o que pode ser causada por diferentes fatores: menor trabalho realizado pela versão paralela do algoritmo, melhor residência dos dados em cache na versão paralela, dentre outros.

Finalmente, a eficiência E corresponde ao tempo da computação paralela investido

em computação essencial, ou seja, operações de computação realizadas pela versão sequencial. A eficiência pode ser definida por $E = \frac{S}{p} = \frac{T_1}{T_1+T_o}$.

5.3.1.1 Modelo de Desempenho para DGEMM

A multiplicação de matrizes definida na biblioteca BLAS implementa a operação $C = \alpha A * B + \beta C$, onde α e β são escalares, A e B são as matrizes de entrada e C é a matriz de entrada/saída. O Algoritmo 1, apresentado por Grama (2003), utiliza as matrizes A, B, C de tamanhos $m \times k$, $k \times n$, e $m \times n$, respectivamente. Usa-se particionamento em uma grade bidimensional chamada P, que totaliza p processadores, ou seja, uma grade de $\sqrt{p} \times \sqrt{p}$ processadores, conforme é representado na Figura 55. Os blocos de dados de A são de tamanho $(m/q) \times (k/q)$, enquanto os blocos de B são de tamanho $(k/q) \times (n/q)$ e, por fim, os blocos de C são de tamanho $(m/q) \times (n/q)$. Logo, cada tarefa de processador possui $\frac{m*n*k}{q}$ operações de soma e de multiplicação. Como temos p processadores em uma grade quadrada, e existe um mapeamento de uma tarefa por processador, o tamanho de q é dado de modo que $q = \sqrt{p}$. Esse componente tem como restrição que $m \geq q$, $n \geq q$ e $k \geq q$, para satisfazer as suas funções de particionamento, e os tamanhos de m, n e k devem ser múltiplos de q, pois não trata os casos em que a divisão dos blocos cria os últimos blocos com menos linhas ou colunas que os demais. Assim como a implementação do PDGEMM do PBLAS (Parallel BLAS)(BLACKFORD *et al.*, 1996), cada nó $P_{i,j}$ já possui os blocos $A_{i,j}$, $B_{i,j}$ e $C_{i,j}$, para $i = 0, ..., \sqrt{p}$ e $j = 0, ..., \sqrt{p}$.

Na Figura 54, são apresentadas as comunicações de uma possível implementação do PDGEMM, onde ocorre uma comunicação *all-to-all broadcast* de cada bloco $A_{i,j}$ para os demais nós da mesma linha da grade $P_{i,j=0..\sqrt{p}-1}$, bem como uma comunicação *all-to-all broadcast* de cada bloco $B_{i,j}$ aos demais nós $P_{i=0..\sqrt{p}-1,j}$. Após, $P_{i,j}$ adquire os escalares α e β através de operações de *one-to-all broadcast* para cada $P_{i,j}$ e realiza as computações de multiplicação e

adição das linhas 3 e 5.

Algoritmo 1: Multiplicação de Matrizes em Blocos (GRAMA, 2003)

```
for i = 0, ..., q − 1 do
    for j = 0, ..., q − 1 do
        C_{i,j} := β × C_{i,j};
        for l = 0, ..., k − 1 do
            C_{i,j} := C_{i,j} + α × A_{i,l} × B_{l,j};
        end
    end
end
```

Nessa implementação baseada em passagem de mensagens (memória distribuída) de DGEMM, o tempo de sobrecarga de execução paralela (T_o) corresponde ao tempo de comunicação (T_{comm}) entre as unidades de processamento. Considere o tempo de comunicação do *All-to-all broadcast* sendo $t_s \times log_2(p) + t_w \times m(p-1)$ (GRAMA, 2003). Já para o envio dos escalares α e β, o custo é de $(t_s + t_w \times m)log(p)$. Substituindo-se p pela quantidade de nós em uma linha/coluna da grade de nós, teremos $\sqrt{p}$ comunicações do mesmo bloco para sua linha na matriz A e coluna na matriz B e substituindo m pelo tamanho dos blocos $\frac{m \times k}{\sqrt{p}}$ e $\frac{k \times n}{\sqrt{p}}$, respectivamente A e B. Considerando ainda que são duas operações de *All-to-all broadcast* e duas operações *One-to-all broadcast*, chegamos a:

$$\begin{aligned} T_{comm} = T_o = & 2 \times (t_s + t_w \times 16) \times log_2(p) + (t_s + t_w \times \frac{m \times k \times 16}{\sqrt{p}} \times \sqrt{p} - 1) + \\ & (t_s + t_w \times \frac{k \times n \times 16}{\sqrt{p}} \times \sqrt{p} - 1) + \frac{2 \times m \times k \times 16}{p} \times t_m \end{aligned} \tag{5.1}$$

onde 16 é o tamanho de representação de um número de ponto flutuante de precisão dupla em *bytes*, supondo uma arquitetura baseada em palavra de 64 bits.

Na modelagem do tempo de processamento paralelo (T_P), é considerada a Equação 5.2, onde t_s é o tempo de *startup* e t_w é o tempo de transferência por palavra, que depende da taxa de transferência do canal. Se a taxa de transferência é de r palavras por segundo, então cada palavra demora $t_w = \frac{1}{r}$ para percorrer a rede.

$$T_p = \frac{(\frac{m \times n \times k}{p} * 4)}{C_i} + (\frac{m \times n \times k}{p} \times 16 \times t_m) + T_{comm} \tag{5.2}$$

, onde C_i é a capacidade de computação do nó de processamento.

Na Equação 5.2, o primeiro termo representa o tempo de computação, onde a linha 3 possui uma multiplicação de um escalar do tipo ponto flutuante de precisão dupla, duas leituras de memória e uma atribuição do valor resultante na memória com o mesmo tipo de dado, executando essa linha $\frac{m}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ vezes. Já na linha 5, ocorre uma soma, duas multiplicações, quatro leituras de memória e uma atribuição, em todos casos com números de precisão dupla, executando $\frac{m}{q} \times \frac{n}{q} \times k$ vezes essa linha. Na equação, foi considerado que todas as operações demandam a mesma quantidade, por conta de simplificação na representação.

Para se prever de forma mais detalhada o desempenho desse componente, deve-se modelar a hierarquia de memórias e eventualmente trabalhar com a parametrização adequada de modo a preservar valores recorrentes nas memórias *cache* e trabalhar com um padrão de acesso que permita reduzir as faltas em memória *cache* para acessos à posições contíguas em memória. Diversas implementações consideram essa abordagem, como ocorre nas bibliotecas BLIS, GotoBLAS e OpenBLAS. Porém, para o propósito desta seção, considerou-se o tempo informado na plataforma para acesso à memória principal. Um modelo de desempenho que detalhe de forma aprofundada os diversos níveis de acesso à memória, considerando todos detalhes da hierarquia de memória, pode ser modelado e suportado pelo Alite, bastando para isso a inclusão dos respectivos parâmetros/argumentos nas assinaturas e contratos dos componentes. Além do acesso à memória, a presente modelagem considerou o custo de instruções de ponto flutuante com base na capacidade teórica de execução de operações de ponto flutuante por segundo dos processadores presentes nas informações dos processadores. Em um cenário ideal, deveria ser cadastrado o mapeamento do tempo de execução de cada instrução em termos de ciclo de relógio, para que combinado com a frequência do CPU, tenha a predição efetuada de forma precisa. Porém, esse tipo de detalhamento é comumente explorado em sistemas de tempo real, onde cada tarefa tem seu tempo restrito por uma garantia temporal necessária devido à criticidade das aplicações, mas essa informação dificilmente é encontrada, devendo ser parametrizada por *hardware* externo (SHAW, 2003).

Dessa forma, no Alite, o parâmetro que custo de comunicação T_{comm} é composto pelos tempos t_s e $t_w m$ informados pelo desenvolvedor do componente através de duas funções que consideram as computações e as características da rede de interconexão. Todos os parâmetros de contexto usados na modelagem estão resumidos na Tabela 15.

Tabela 15 – Resumo dos parâmetros envolvidos na modelagem de desempenho do componente BLAS3MM.

Parâmetro	Descrição
m	quantidade de linhas da matriz A identificada no contrato do componente a ser resolvido
n	quantidade de colunas da matriz B identificada no contrato do componente a ser resolvido
k	quantidade de colunas de A e de linhas de B identificada no contrato do componente a ser resolvido
t_s	tempo de *startup* identificada no contrato da plataforma
t_w	tempo de envio de uma palavra de dados pela rede, calculado como o inverso da taxa de transferência da rede de comunicação, identificada no contrato da plataforma
C_i	capacidade de computação de um nó da plataforma, calculado a partir das capacidades teórica de cada processador e multiplicada pela quantidade de processadores por nó identificada no contrato da plataforma
p	quantidade de nós da plataforma, identificada no contrato da plataforma
t_m	tempo de acesso à memória, identificada no contrato da plataforma
MPI_DOUBLE	constante definida em âmbito da HPC Shelf

Fonte: Próprio Autor

5.3.2 Parâmetros de Custo

O modelo de custo usado no Alite foi definido com base na utilização de cada componente de *hardware* por unidade de tempo, de forma análoga ao que é praticado no *framework* CloudSim, e outros ambientes de nuvens públicas como o Amazon EC2, pois permite quantificar a utilização da infraestrutura e ainda permite que, mesmo com a elasticidade, os custos possam ser quantificados. Esse modelo de tarifação homogeniza o tratamento dos custos, e facilita a integração com o mecanismo de elasticidade (CALHEIROS *et al.*, 2011).

Como a HPC Shelf é um ambiente de nuvem que trabalha em múltiplos níveis de abstração e divisão de responsabilidades, cada componente deve ter um custo associado para ser considerado no momento do cálculo do custo final da execução da aplicação do usuário especialista. Analogamente, o custo energético de um nó é calculado com base na quantidade de núcleos de processamento e a presença de aceleradores computacionais, podendo variar de acordo com a ação dos mecanismos de elasticidade.

Ressalta-se que, embora seja definida a possibilidade de trabalhar com granularidade fina de quantificação de custos, os mantenedores podem definir de forma personalizada o cálculo de seus custos. Mesmo quando definido em granularidade fina de tarifação, o Alite não contempla

o acompanhamento dinâmico de custos, ficando a cargo do mecanismo de elasticidade o controle das alterações e da tarifação dinâmica.

Para a validação do Alite, foram definidos os parâmetros gerais de custo financeiro e custo energético. O custo financeiro do componente sistema é definido pela composição dos custos dos componentes de *software* utilizados e o custo da plataforma virtual. O custo do perfil de plataforma por sua vez, é composto pelo produto do tempo previsto de execução pela soma do produto das quantidades de cada unidade básica de execução (quantidade de CPUs, memória, armazenamento, acelerador computacional) pelo seu custo unitário.

$$C_n = C_c \times c + C_m \times m + C_a \times a \tag{5.3}$$

A Equação 5.3 define o cálculo do custo de cada nó da plataforma de execução (C_n), onde C_c, c, C_m, m, C_a e a, são respectivamente: o custo por vcpu (cpu virtual), a quantidade de vcpus, o custo de cada GB de memória principal, a quantidade de memória principal em GB, o custo de cada acelerador computacional e a quantidade de aceleradores.

$$C_p = C_n \times m + C_s \tag{5.4}$$

Com base no custo unitário por nó, pode-se calcular o custo da plataforma de execução por unidade de tempo (C_p) através da equação 5.4, onde m é a quantidade de nós da plataforma e C_s é o custo de armazenamento da plataforma.

$$C_e = C_p \times T_i + C_b \tag{5.5}$$

Uma vez conhecido o custo da plataforma, é possível calcular o custo total de execução (C_e) através da Equação 5.5, onde C_b representa o custo total de *software* do componente e T_i representa o tempo estimado da i-ésima alternativa de componente.

$$C_y = (m \times E_n) \times T_i \tag{5.6}$$

Por fim, o custo energético é calculado pela equação 5.6, onde E_n representa o consumo energético de cada nó.

5.3.3 Parâmetros de Ranqueamento

Os parâmetros de ranqueamento são definidos de forma que seja possível classificar os resultados da resolução dos contratos contextuais. Cada parâmetro de ranqueamento representa a aplicação de um método de escolha baseada em múltiplos critérios, representando uma política de alocação de recursos da nuvem.

Muitos esforços vem sendo empregados em pesquisa para alcançar uma forma adequada de escolher implementações de *software* e *hardware* para executar as aplicações. Existem diversas técnicas que podem ser aplicadas para possibilitar a análise dos candidatos disponíveis para escolha da melhor alternativa. A área de pesquisa operacional concentra-se em estudar esses problemas e propor uma série de métodos que vem sendo empregados nas mais diversas áreas da engenharia, economia, dentre outas, conforme identificado por Velasquez e Hester (2013). Nesse processo de análise dos candidatos, diversos critérios são analisados e ponderados para modelar adequadamente o problema, e para dificultar essa análise, muitas vezes os critérios apresentam interesses conflitantes e/ou diferentes unidades de medida.

Majoritariamente, nos provedores SaaS que possuem recursos computacionais próprios, ou ainda com suporte à integração com ambientes de nuvem IaaS externos através de perfis homogêneos de *hardware* virtualizado, tem-se o uso de métodos que consideram o histórico das execuções para auxiliar no processo de escolha de componentes, como é o caso de algoritmos evolutivos, algoritmos genéticos, dentre outros. Uma visão geral dos métodos pode ser obtida em (AMALARETHINAM; BEENA, 2014). Além das pesquisas em escolha de qual aplicação irá executar em qual ambiente de nuvem, existem trabalhos que buscam calcular quando executar a aplicação na nuvem mais é vantajoso que executar em um *cluster* local (GUPTA *et al.*, 2016).

Como forma de demonstrar o emprego de técnicas de classificação de componentes junto ao Alite, foram considerados, para fins dos experimentos deste trabalho, métodos de escolha empregados quando existem múltiplos critérios (MCMD[18]), provenientes da subárea de pesquisa operacional, que serão apresentadas na próxima seção. A escolha por métodos que não utilizam o histórico das execuções ou necessitam de treinamento para identificar o comportamento se deu pela natureza dinâmica e heterogênea que apresentam os recursos computacionais manipulados pelo Alite, diferente das demais iniciativas que apresentam um conjunto fixo de possíveis recursos.

É importante enfatizar que o Alite não está limitado ao emprego da técnica de

[18] *Multiple Criteria Decision Making*

classificação utilizada nos experimentos detalhados a seguir. Outras técnicas de escolha podem ser incorporadas ao Alite, sendo este um estudo de caso com propósitos de avaliação do arcabouço proposto.

5.3.3.1 Tomada de Decisão com Múltiplos Critérios

A escolha com múltiplos critérios é uma subárea da pesquisa operacional, chamada de *Multiple Criteria Decision Making* (MCDM) ou *Multiple Criteria Decision Analysis*, onde a classificação ou escolha da melhor opção baseada em múltiplos critérios é feita por um computador (DYER *et al.*, 1992; XU; YANG, 2001). Dentro da MCDM, os métodos são agrupados de acordo com a quantidade de alternativas em avaliação. A sub-área *Multi-attribute decision-making* é voltada para alternativas discretas e a área *Multi-objective decision-making* é adequada quando um número teoricamente contínuo de alternativas é definido através de um vetor de variáveis de decisão, como é o caso dos sistemas computacionais considerados pelo Alite. O desenvolvimento das técnicas de análise de decisão de multi-critérios tem seu histórico iniciado por volta do ano 1970, e tendo como base várias escolas, que definiam a forma de promover as escolhas. Dessas escolas, destacam-se as escolas Americana e Francesa (Europeia). Os métodos da escola Americana, de modo geral, são baseados na comparação par-a-par entre alternativas, o que interfere negativamente no tempo total de execução da classificação. Já nos métodos da escola Francesa, geralmente são utilizadas probabilidades e ponderação dos critérios para efetuar a classificação (SALOMON, 2004).

Para o entendimento dos métodos de decisão multi-critérios, é necessária a definição de dois conceitos, as alternativas e os critérios:

- **a**lternativas, que representam os elementos a serem ranqueados, os quais, neste trabalho, correspondem aos componentes sistema candidatos para execução, calculada pela fase de seleção da resolução de contratos contextuais;
- **c**ritérios, que representam cada atributo das alternativas analisado no processo de avaliação. Neste trabalho, representam propriedades expressas em parâmetros de contexto, qualidade e custo, a respeito de componentes sistema, que serão considerados na classificação das alternativas.

De acordo com Xu e Yang (2001), os critérios considerados na análise podem ser quantitativos ou qualitativos. No entanto, somente os critérios quantitativos são de interesse no processo de escolha de componentes do Alite, pois a interação entre os *Frontend's* e o Alite é

automatizada.

Ao se considerar diferentes métricas, alguns critérios podem ser conflitantes, de forma a não ser possível satisfazer a ambos simultaneamente. Além disso, como discutido na Seção 4.3.2.2, um critério quantitativo pode ser ascendente (ou de benefício), quando o aumento do seu valor reflete em uma melhor opção, ou descendente (ou de custo), quando a diminuição de seu valor reflete na melhor opção (XU; YANG, 2001). Essas situações são tratadas neste trabalho de modo a delegar à HPC Shelf o correto tratamento de valores conflitantes.

Seja a matriz de decisão da apresentada na Tabela 16, contendo M alternativas e N critérios. Seja A_i, para $i \in \{1,2,\ldots,M\}$, cada alternativa da lista que necessita ser classificada, C_j, para $j \in \{1,2,\ldots,N\}$, cada critério de classificação, e W_j cada peso associado ao critério C_j. Considere, ainda, a_{ij}, cada valor associado ao critério C_j da alternativa A_i. Por fim:

$$\sum_{j=1}^{N} W_j = 1$$

Tabela 16 – Matriz de Decisão M

	C_1	C_2	C_3	$\ldots$	C_N
Alternativa	W_1	W_2	W_3	$\ldots$	W_N
A_1	a_{11}	a_{12}	a_{13}	$\ldots$	a_{1N}
A_2	a_{21}	a_{22}	a_{23}	$\ldots$	a_{2N}
A_3	a_{31}	a_{32}	a_{33}	$\ldots$	a_{3N}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
A_M	a_{M1}	a_{M2}	a_{M3}	$\ldots$	a_{MN}

Fonte: (TRIANTAPHYLLOU; SáNCHEZ, 1997).

O método MCDM mais simples, antigo e disseminado é a soma ponderada, *Weighted Sum Model* (WSM). Outro método, largamente utilizado é uma extensão do WSM, o produto ponderado *Weighted Product Model* (WPM). A união desses dois métodos forma o WASPAS, um método que combina WSM e WPM e faz uma ponderação entre as notas dos dois para cada alternativa para gerar uma nota final, ou isolar um dos dois métodos dependendo de um valor λ predefinido. Ambos os métodos são exemplos que podem ser usados no Alite. Nos últimos anos, uma diversidade de métodos vem sendo desenvolvidos, cada um possuindo como objetivo uma categoria de problemas específica. Dentre esses métodos, além do WASPAS, destacam-se o TOPSIS e o VIKOR (TZENG; HUANG, 2011). Todos esses métodos trabalham com critérios conflitantes e em diferentes unidades, classificando-os como multidimensionais.

O método de soma ponderada consiste do somatório dos valores dos critérios de cada alternativa multiplicados pelo peso associado ao mesmo, conforme é possível observar na equação 5.7 (FÜLÖP, 2005).

$$Rank_i = \sum_{j=1}^{N} a_{ij} * w_j \tag{5.7}$$

Por sua vez, o método WPM difere do WSM pelo uso do produtório ao invés do somatório e do cálculo de razões par-a-par, conforme podemos observar na Equação 5.8 (TRIANTAPHYLLOU; SáNCHEZ, 1997), onde $k \in \{1, 2, \ldots, M\}$.

$$R\left(\frac{a_i}{a_k}\right) = \prod_{j=1}^{N} \left(\frac{a_{i,j}}{a_{k,j}}\right)^{W_j} \tag{5.8}$$

O ranqueamento é feito de acordo com o valor da razão R. Quando seu resultado for maior que 1, as duas alternativas comparadas devem ter suas posições trocadas. Para se trabalhar com variáveis que diferentes dimensões e diferentes objetivos, os valores da matriz de decisão são normalizados.

Finalmente, o método TOPSIS (*Technique for Order Preference by Similarity to Ideal Solutions*) compõe o melhor valor de cada critério em relação a todas alternativas, bem como o pior e faz a ordenação considerando como melhor alternativa a que fique mais próxima do melhor valor e mais distante do pior valor de cada critério. O método *Multicriteria Optimization and Compromise Solution* (VIKOR) introduz um índice ranqueamento multicritério baseado na medição particular da proximidade em relação à "solução ideal" $F*$. Mateo (2012) descreve os processos de cálculo de cada uma das técnicas citadas.

O Alite foi prototipado de modo que os diversos métodos de escolha multicritério possam ser utilizados, e através da integração do Alite com a linguagem R e seu vasto repositório de métodos MCDM, os métodos utilizados podem ser facilmente substituídos.

5.4 Avaliação de Desempenho

Foram conduzidos estudos de desempenho para avaliar o Alite em relação a dois aspectos:

- o tempo de resolução de contratos contextuais, da *seleção* à *classificação*;
- a qualidade da classificação usando técnicas MCMD descritas na Seção 5.3.3.1;

Para o estudo experimental, o ambiente de testes foi composto por um computador com processador Intel(R) Core(TM) $i5-2500$ CPU @ 3.30GHz com 4 núcleos físicos e 4 lógicos, possuindo 6MB de memória *cache* e 8GB de memória principal (7GB disponível para a aplicação). O sistema operacional usado foi o Ubuntu 14.04.4 LTS (kernel $3.13.0-83-$ *generic*x86_64), juntamente com o banco de dados PostgreSQL 9.6, o Apache Tomcat/7.0.54, o Axis2 e o Oracle Java(TM) SE Runtime Environment (build $1.8.0\_77-b03$). O computador cliente possui um processador Intel(R) Core(TM) $i5-4210U$ CPU @ 1.70GHz com 2 núcleos físicos e 2 lógicos, possuindo 3 MB de memória *cache* e 12 GB de memória principal. O sistema operacional usado foi o Ubuntu 16.04.3 LTS (kernel $4.4.0-96-$ *generic*x86_64).

5.4.1 Tempo de Resolução de Contratos Contextuais

Embora na HPC Shelf os processamentos realizados por aplicações sejam definidos através da composição dinâmica de sistemas de computação paralela, a validação do Alite irá considerar a resolução dos componentes isoladamente, pois o interesse da avaliação reside essencialmente na dinâmica de resolução de contratos contextuais. Vale ressaltar que o controle das requisições de resolução de componentes da aplicação realizados a partir de sistemas de computação paralela gerenciados pelo SAFe, bem como o controle da execução que ocorrerá através da interação entre o SAFe e o *Backend*, já foi abordado na Tese de Doutorado de Jefferson Silva (SILVA, 2016; de Carvalho Junior *et al.*, 2019).

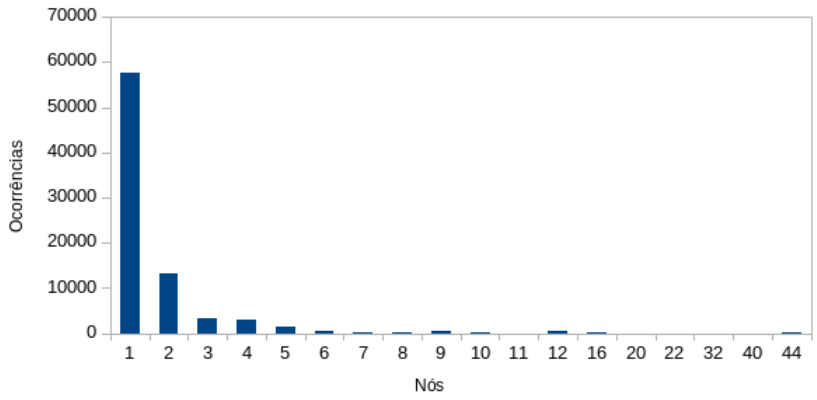
Para os estudos de caso, foram registrados no Alite diversos perfis de plataformas de execução baseados em configurações factíveis nos *clusters* CENAPAD-UFC e Santos Dumont-LNCC pertencentes ao Sistema Nacional de Processamento de Alto Desempenho (SINAPAD[19]), e no *cluster* de pequeno porte do MDCC-UFC. Essas três infraestruturas computacionais, do ponto de vista da HPC Shelf, são vistos como três usuários *mantenedores* distintos. Para cada mantenedor, foram definidas diferentes configurações de perfis de plataforma para duas avaliações distintas, a primeira de desempenho e a segunda de qualidade do algoritmo de resolução do Alite. Esses perfis foram definidos quantitativamente com base nas configurações recorrentes de execução no CENAPAD-UFC.

A fim de definir a quantidade de perfis de plataformas distintos necessária para validar o tempo de resolução de contratos contextuais foi realizado um estudo sobre o histórico da quantidade de nós de execução do CENAPAD-UFC, a fim de determinar a quantidade de

[19] <https://www.lncc.br/sinapad/>

perfis que já foram solicitados no período que compreende desde o início da operação em 2011 até junho de 2017. Foram observados 47 diferentes perfis de acordo com diferentes quantidades de CPUs virtuais (vCPU). No entanto, como o processo de requisição de recursos computacionais tem o nó como menor fração, esses perfis foram agrupados de acordo com a quantidade de CPUs virtuais por nó, reduzindo para o total de 18 variações de quantidade de nós para o CENAPAD-UFC. O gráfico da Figura 56 apresenta a frequência de ocorrências de cada quantidade de nós que tiveram pelo menos uma execução no período observado. Observa-se que ocorrem majoritariamente execuções com somente um nó processador, chegando a 5 nós em alguns casos, enquanto o restante das configurações tiveram poucas ocorrências ou nenhuma. Em adição à esse histórico de execução, outros dois perfis contendo aceleradores computacionais do tipo GPU foram considerados, os quais não estavam disponíveis no sistema de fila de tarefas na época do estudo, totalizando assim 20 perfis para o CENAPAD-UFC.

Figura 56 – Frequência de Perfis utilizados no CENAPAD-UFC



Fonte: Próprio Autor

Uma vez que a HPC Shelf foi concebida para ser utilizada em um ambiente multi-organizacional, e possivelmente de escala global, optou-se por considerar a quantidade de perfis identificada para o CENAPAD-UFC como base e padronizar para os demais mantenedores de plataforma na avaliação de desempenho. Para gerar a carga, foram cadastrados perfis de plataforma fictícios, baseados em características dos perfis já existentes, evitando que eles sejam rejeitados no processo de validação. A quantidade de perfis gerada foi definida com base nos 20 perfis das plataformas identificados no estudo sobre o histórico de execução do CENAPAD-UFC, e 50 centros computacionais distintos (9 CENAPADS e 41 universidades federais, considerando

um *cluster* por universidade), totalizando 1000 perfis cadastrados. Também foi analisada a diferença entre os tempos de requisição na rede local do servidor e de requisições através da internet.

Para avaliar o tempo de resolução do algoritmo, definimos três contratos de componentes BLAS3MM (Seção 5.3). Esses contratos tem a sua quantidade de correspondências após a resolução controlada através do parâmetro que define a localização dos perfis de plataformas. Assim, é possível retornar todas as plataformas compatíveis com o componente de *software*, um subconjunto delas ou até mesmo nenhuma correspondência. O contrato *A* não possui plataforma compatível registrada no Alite, evitando a maior parte do trabalho computacional do algoritmo de resolução. Por sua vez, *B* e *C* restringem alguns atributos e nenhum atributo, respectivamente, de modo que todas as plataformas virtuais serão selecionados na carga *C* para formação dos componentes sistema candidatos. Para cada contrato, definimos uma carga de resolução variando a quantidade de perfis de plataformas registradas no serviço *Core* em 16 perfis na carga 1, 100 perfis na carga 2 e 1000 perfis na carga 3.

A avaliação da execução do algoritmo de resolução considerou os parâmetros de qualidade, custo e ranqueamento identificados nas seções 5.3.1, 5.3.2 e 5.3.3.

Figura 57 – Tempo de execução no servidor dos contratos A, B e C

Carga	A	B	C
1	0,116s	3,80s	3,75s
2	0,122s	22,03s	22,19s
3	0,123s	216,43s	217,45s

Fonte: Próprio Autor

Os tempos da Figura 57 correspondem ao tempo de construção da lista de candidatos realizadas na máquina servidora do serviço de resolução do *Core.* As cargas 1, 2 e 3 representam, respectivamente, 16, 100 e 1000 diferentes plataformas e 2 diferentes componentes Multiplicação de Matizes registrados no *Core* que satisfazem as condições dos contratos A, B e C.

É relevante ressaltar que, além da implementação do mecanismo de resolução de contratos não ser totalmente otimizada, o *hardware* utilizado como servidor para executar o Alite foi modesto em relação ao que seria possível utilizar em uma infraestrutura de computação que hospedasse o serviço Core. Assim, é esperado que o uso em ambiente de produção responderia à requisições de resolução contratos em tempos menores do que os observados nos testes de desempenho apresentados nesta seção.

Tabela 17 – Plataformas registradas no Alite para avaliação qualitativa
max width=

nome	quantidade de nós	CPUs por nó	núcleos por CPU	GPU
CENAPAD-UFC-Micro	2	2	6	-
CENAPAD-UFC-Small	4	2	6	-
CENAPAD-UFC-Medium	8	2	6	-
CENAPAD-UFC-Large	12	2	6	-
CENAPAD-UFC-Micro-GPU	2	2	8	1 × K20M
CENAPAD-SD-MESCA2	1	16	12	-
CENAPAD-SD-B710-Micro	2	2	12	-
CENAPAD-SD-B715-GPU-Micro	2	2	12	2 × K40
CENAPAD-SD-B715-MIC-Micro	2	2	12	-
MDCC-GPU-Micro	1	2	10	1 × K40
MDCC-MIC-Micro	1	2	10	-
MDCC-CPU-Micro	1	2	10	-
MDCC-CPU-Small	2	2	10	-
MDCC-CPU-Medium	4	2	10	-
MDCC-CPU-Large	8	2	10	-

Fonte: Próprio Autor

5.4.2 Avaliação da Qualidade da Classificação

Para a avaliação qualitativa, foram registradas as plataformas representadas na Tabela 17, com suas respectivas características.

O Alite gera uma lista de todos os sistemas computacionais que são compatíveis com os contratos da aplicação e que tenham plataforma virtuais compatíveis registrada no *Core* na etapa de seleção, sendo portanto alternativas seguras para serem utilizadas. O aspecto qualitativo do algoritmo de classificação do Alite é avaliado quanto à qualidade da classificação da lista de sistemas computacionais resultante da etapa de classificação. A solução de classificação do Alite permite que métodos de escolha baseada em múltiplos critérios (DYER *et al.*, 1992) possam ser utilizados para ranquear as alternativas. A avaliação consiste em comparar os resultados obtidos pelo algoritmo de cálculo, de modo a identificar se o comportamento dos métodos preserva as características conhecidas. Ressalta-se que essa implementação do Alite gera uma matriz de decisão para cada parâmetro de ranqueamento, de modo a fornecer um formato de saída de dados compatível com ferramentas de avaliação de escolhas baseadas em múltiplos critérios. Dessa forma, a validação será considerada satisfatória se o escore dos métodos utilizados estiverem de acordo com as características definidas no parâmetro de ranqueamento.

O processo de análise da estratégia de classificação foi desenvolvido comparando-se diversas matrizes de decisão, cada uma ponderando diferentes critérios. Os pesos das 6 primeiras

matrizes foram definidos de modo que as prioridades estivessem ordenadas e que um peso de maior prioridade fosse o dobro do peso de menor prioridade subsequente. O mesmo ocorre em relação aos demais critérios. Na sétima matriz, os critérios foram ponderados igualitariamente. Por sua vez, nas 3 matrizes seguintes, foram considerados somente um critério. Finalmente, nas últimas 6 matrizes, foram consideradas todas combinações de matrizes par-a-par, com os pesos definidos de forma que um critério tenha o dobro de peso do outro. Os resultados dessa avaliação serão apresentados na Seção 5.4.

Ao mesmo tempo em que variaram os pesos dos critérios, também foram aplicados os principais métodos da área de análise multicritério que não são baseados na comparação par-a-par.

Para a validação da qualidade da classificação dos componentes sistema candidatos, o mesmo contrato utilizado para a validação do tempo de resolução foi aplicado a um conjunto de perfis de plataformas reais, as quais foram registradas no Alite. A partir da lista de sistemas computacionais gerada com esses perfis, foram extraídos os parâmetros calculados que representam custo financeiro, consumo energético e tempo de execução. Para a aplicação dos métodos de escolha com múltiplos critérios, foram definidas dez matrizes de decisão, com diferentes pesos para cada um dos parâmetros de qualidade calculados. A Tabela 18 apresenta as diversas variações de pesos utilizadas na validação, bem como pode ser observado que todas as combinações de alternativas e pesos foram realizadas com os 3 parâmetros.

Já a Tabela 19 apresenta os valores calculados para cada parâmetro de contexto. Ressalta-se que nessa tabela há um valor discrepante dos demais por conta de ser um perfil especial do supercomputador Santos Dumont chamado CENAPAD-SD-MESCA2, que possui 240 núcleos e 6 TB de RAM.

Em posse da matriz de alternativas, podemos classificar os resultados do *Core* através de bibliotecas de MCDM disponibilizadas no repositório de bibliotecas da linguagem R[20], restrito aos métodos baseados em matriz de alternativas. Optou-se por não considerar as demais estratégias que utilizam algoritmos evolutivos, ou que simplesmente necessitam do histórico de execuções para produzir uma base de dados utilizados no processo de decisão.

As tabelas 20 à 29 apresentam as matrizes de decisão de cada parâmetro de ranqueamento. Nessas tabelas, há a variação dos pesos dos parâmetros que representam uma matriz de comparação dos coeficientes de correlação de Spearman, adequado para comparar

[20] <https://cran.r-project.org/web/packages/MCDM/index.html>

conjuntos ordenados. Os coeficientes foram calculados com base na comparação dos métodos de classificação identificados na Seção 5.3.3, juntamente com as classificações feitas manualmente por três especialistas humanos (P_1, P_2 e P_3).

O pesquisador P_1 é especialista em processamento paralelo em GPU, com vasto conhecimento em programação SIMD. Já o pesquisador P_2 é especialista em modelos de especificação formal de componentes em *workflows* de componentes. Por fim, o pesquisador P_3 é especialista em ambientes de execução paralela, gerenciando um *cluster* de grande porte na cidade de Fortaleza. Todos os perfis de pesquisadores são graduados em Ciência da Computação, com mestrado e doutorado em Computação.

Com todos os pesquisadores, o problema foi exposto com um conjunto de informações padronizado. Cada pesquisador recebeu:

- O contrato resolvido pelo Alite.
- Lista de sistemas computacionais resultantes da resolução do contrato.
- Uma tabela sumarizando os valores dos parâmetros de Qualidade e de Custo usados como critérios de escolha multicritério, com os seus respectivos pesos.

Ressalta-se que nenhuma informação sobre os resultados dos algoritmos foi repassada aos pesquisadores, a fim de não influenciar as suas escolhas.

Como resultado, os pesquisadores devolveram a lista ranqueada de acordo com suas próprias estratégias. Embora as estratégias avaliadas no Alite não tenham sido divulgadas aos pesquisadores, os especialistas 1 e 2 utilizaram um método de ordenação semelhante ao algoritmo da ordenação lexicográfica, onde os pesos definem a ordem de prioridade e a estratégia restringe ao critério de maior peso, a prioridade da ordenação. Esse método possui como desvantagem os casos em que os pesos dos critérios 2 e 3 superem a diferença absoluta de valores juntamente com os respectivos pesos. Além disso, a forma como aplicaram o método não considerou a natureza multidimensional dos critérios, que pode gerar compensação dos valores. Já o especialista 3 usou uma etapa de normalização dos valores dos critérios para tratar a natureza multidimensional de tais critérios, porém ainda utiliza a ordem lexicográfica.

Embora tenham sido comparados nesse experimento, devido à integração com o interpretador da linguagem R, outros métodos podem ser utilizados, bastando importar a respectiva biblioteca. Os valores calculados representam dessa forma, sugestões para que os usuários especialistas possam escolher, mas que podem ser substituídas de acordo com as necessidades dos mesmos.

Tabela 18 – Tabela das variações dos pesos dos critérios de custo, energia e tempo, utilizados na validação da etapa de classificação do Alite.

Rank	Custo de execução	Consumo energético	Tempo de execução
0	4/7	2/7	1/7
1	4/7	1/7	2/7
2	1/7	4/7	2/7
3	2/7	4/7	1/7
4	1/7	2/7	4/7
5	2/7	1/7	4/7
6	1/3	1/3	1/3
7	1	0	0
8	0	0	1
9	0	1	0

Fonte: Próprio autor.

Tabela 19 – Valores dos Parâmetros de Ranqueamento do Alite

Alternativa	Custo de execução	Consumo energético	Tempo de execução
0	2,02	15,21	0,04
1	2,01	15,22	0,02
2	2,00	15,24	0,01
3	2,00	15,28	0,01
4	7,81	15,20	0,04
5	492,68	147,20	0,08
6	5,29	18,41	0,04
7	5,35	18,41	0,04
8	5,35	18,41	0,04
9	5,27	16,80	0,08
10	5,27	16,80	0,08
11	5,33	16,80	0,08
12	5,29	16,81	0,04
13	5,28	16,83	0,02
14	5,28	16,88	0,01
15	15,25	48,81	0,00

Fonte: Próprio autor.

Tabela 20 – Matriz de Correlação com pesos 4/7 para Custo Financeiro, 2/7 para Consumo Energético e 1/7 para Tempo de Execução

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,99	0,68	0,74	0,75	0,99	0,91
P_2	0,99	1,00	0,74	0,79	0,80	1,00	0,92
P_3	0,68	0,74	1,00	0,99	0,99	0,74	0,66
TOPSIS	0,74	0,79	0,99	1,00	0,99	0,79	0,71
VIKOR	0,75	0,80	0,99	0,99	1,00	0,80	0,72
WPM	0,99	1,00	0,74	0,79	0,80	1,00	0,92
TOPSIS Linear	0,91	0,92	0,66	0,71	0,72	0,92	1,00

Fonte: Próprio autor.

Tabela 21 – Matriz de Correlação com pesos 4/7 para Custo Financeiro, 1/7 para Consumo Energético e 2/7 para Tempo de Execução

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,99	0,54	0,72	0,67	0,74	0,78
P_2	0,99	1,00	0,61	0,77	0,72	0,79	0,81
P_3	0,54	0,61	1,00	0,95	0,98	0,92	0,82
TOPSIS	0,72	0,77	0,95	1,00	0,98	0,97	0,91
VIKOR	0,67	0,72	0,98	0,98	1,00	0,96	0,88
WPM	0,74	0,79	0,92	0,97	0,96	1,00	0,94
TOPSIS Linear	0,78	0,81	0,82	0,91	0,88	0,94	1,00

Fonte: Próprio autor.

Tabela 22 – Matriz de Correlação com pesos 1/7 para Custo Financeiro, 4/7 para Consumo Energético e 2/7 para Tempo de Execução

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,87	0,65	0,65	0,65	0,48	0,96
P_2	0,87	1,00	0,38	0,38	0,38	0,25	0,85
P_3	0,65	0,38	1,00	1,00	1,00	0,95	0,74
TOPSIS	0,65	0,38	1,00	1,00	1,00	0,96	0,74
VIKOR	0,65	0,38	1,00	1,00	1,00	0,96	0,74
WPM	0,48	0,25	0,95	0,96	0,96	1,00	0,58
TOPSIS Linear	0,96	0,85	0,74	0,74	0,74	0,58	1,00

Fonte: Próprio autor.

Tabela 23 – Matriz de Correlação com pesos 2/7 para Custo Financeiro, 4/7 para Consumo Energético e 1/7 para Tempo de Execução

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,95	0,56	0,56	0,56	0,69	0,86
P_2	0,95	1,00	0,49	0,49	0,49	0,61	0,79
P_3	0,56	0,49	1,00	1,00	1,00	0,95	0,85
TOPSIS	0,56	0,49	1,00	1,00	1,00	0,96	0,86
VIKOR	0,56	0,49	1,00	1,00	1,00	0,96	0,86
WPM	0,69	0,61	0,95	0,96	0,96	1,00	0,92
TOPSIS Linear	0,86	0,79	0,85	0,86	0,86	0,92	1,00

Fonte: Próprio autor.

Tabela 24 – Matriz de Correlação com pesos 1/7 para Custo Financeiro, 2/7 para Consumo Energético e 4/7 para Tempo de Execução

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,65	0,69	0,69	0,69	0,64	0,53
P_2	0,65	1,00	1,00	1,00	0,99	0,99	0,89
P_3	0,69	1,00	1,00	1,00	1,00	0,99	0,88
TOPSIS	0,69	1,00	1,00	1,00	1,00	0,99	0,88
VIKOR	0,69	0,99	1,00	1,00	1,00	0,99	0,89
WPM	0,64	0,99	0,99	0,99	0,99	1,00	0,88
TOPSIS Linear	0,53	0,89	0,88	0,88	0,89	0,88	1,00

Fonte: Próprio autor.

Tabela 25 – Matriz de Correlação com pesos 2/7 para Custo Financeiro, 1/7 para Consumo Energético e 4/7 para Tempo de Execução

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,60	0,60	0,62	0,62	0,65	0,51
P_2	0,60	1,00	1,00	1,00	1,00	0,96	0,79
P_3	0,60	1,00	1,00	1,00	1,00	0,96	0,79
TOPSIS	0,62	1,00	1,00	1,00	1,00	0,97	0,82
VIKOR	0,62	1,00	1,00	1,00	1,00	0,97	0,82
WPM	0,65	0,96	0,96	0,97	0,97	1,00	0,86
TOPSIS Linear	0,51	0,79	0,79	0,82	0,82	0,86	1,00

Fonte: Próprio autor.

Tabela 26 – Matriz de Correlação com pesos 1/3 para Custo Financeiro, 1/3 para Consumo Energético e 1/3 para Tempo de Execução

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,79	0,62	0,65	0,65	0,76	0,55
P_2	0,79	1,00	0,67	0,70	0,70	0,67	0,68
P_3	0,62	0,67	1,00	1,00	1,00	0,93	0,87
TOPSIS	0,65	0,70	1,00	1,00	1,00	0,94	0,89
VIKOR	0,65	0,70	1,00	1,00	1,00	0,94	0,89
WPM	0,76	0,67	0,93	0,94	0,94	1,00	0,83
TOPSIS Linear	0,55	0,68	0,87	0,89	0,89	0,83	1,00

Fonte: Próprio autor.

Tabela 27 – Matriz de Correlação com pesos 1 para Custo

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	1,00	1,00	0,99	0,99	0,99	0,99
P_2	1,00	1,00	1,00	0,99	0,99	0,99	0,99
P_3	1,00	1,00	1,00	0,99	0,99	0,99	0,99
TOPSIS	0,99	0,99	0,99	1,00	1,00	1,00	1,00
VIKOR	0,99	0,99	0,99	1,00	1,00	1,00	1,00
WPM	0,99	0,99	0,99	1,00	1,00	1,00	1,00
TOPSIS Linear	0,99	0,99	0,99	1,00	1,00	1,00	1,00

Fonte: Próprio autor.

Tabela 28 – Matriz de Correlação com pesos 1 para Tempo

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	0,99	0,64	0,64	0,64	0,64	0,64
P_2	0,99	1,00	0,65	0,65	0,65	0,65	0,65
P_3	0,64	0,65	1,00	1,00	1,00	1,00	1,00
TOPSIS	0,64	0,65	1,00	1,00	1,00	1,00	1,00
VIKOR	0,64	0,65	1,00	1,00	1,00	1,00	1,00
WPM	0,64	0,65	1,00	1,00	1,00	1,00	1,00
TOPSIS Linear	0,64	0,65	1,00	1,00	1,00	1,00	1,00

Fonte: Próprio autor.

Tabela 29 – Matriz de Correlação com pesos 1 para Energia

	P_1	P_2	P_3	TOPSIS	VIKOR	WPM	TOPSIS Linear
P_1	1,00	1,00	0,99	0,99	0,99	0,99	0,99
P_2	1,00	1,00	1,00	1,00	1,00	1,00	1,00
P_3	0,99	1,00	1,00	1,00	1,00	1,00	1,00
TOPSIS	0,99	1,00	1,00	1,00	1,00	1,00	1,00
VIKOR	0,99	1,00	1,00	1,00	1,00	1,00	1,00
WPM	0,99	1,00	1,00	1,00	1,00	1,00	1,00
TOPSIS Linear	0,99	1,00	1,00	1,00	1,00	1,00	1,00

Fonte: Próprio autor.

5.4.3 Análise e Discussão dos Resultados

Nos testes foram geradas diferentes cargas para o Alite a fim de identificar possíveis tempos para a resolução dos contratos contextuais que inviabilizassem seu uso em um ambiente de nuvem computacional com as demandas de HPC. A implementação se mostrou adequada à carga proposta para validação, onde foram cadastradas até 1000 plataformas com 3 contratos candidatos para a resolução, chegando a um total de 2005 componentes sistema candidatos.

O protótipo desenvolvido neste trabalho utiliza uma única linha de execução, tanto para resolução de um único contrato quanto para atendimento de múltiplas requisições. Logo, em um ambiente real, seu desempenho pode ser bastante melhorado através da exploração do paralelismo, tanto na implementação do algoritmo de resolução quanto no atendimento de múltiplas requisições de resolução de contratos.

A escolha por cadastrar uma grande quantidade de plataformas foi feita de modo a simular um ambiente realístico, como vemos em diversas bibliotecas científicas que para cada tarefa específica, apresentam poucas possibilidades, conforme pôde ser observado ao verificar a biblioteca BLAS.

Para avaliar a etapa de classificação, foram comparadas diversas técnicas de analise multicritério para classificação do Alite com a classificação manual de três especialistas de HPC. As tabelas 20 à 29 apresentam as matrizes de correlação das classificações, onde estão representados os coeficientes de correlação entre cada estratégia de classificação utilizado pelo Alite e os especialistas. Nessas tabelas, também é possível identificar a correlação entre a ordenação dos diversos métodos avaliados.

Nas tabelas 20, 21, 24, 26, 27, 28 e 29, os métodos apresentaram coeficiente de correlação mais forte que nos demais casos. Isso aconteceu pelo fato de que os valores analisados não interferem na classificação final. Por sua vez, nas tabelas 22, 23 e 25, a correlação entre os

métodos automáticos e os especialista foi reduzida, por conta das estratégias dos especialistas de não considerar a compensação dos valores que ocorre quando o valor de um critério é muito maior que outro. Dessa forma, a diferença acaba sobrepondo os pesos dos critérios, levando a uma ordenação incorreta dos sistemas computacionais. Nesse sentido, considerando os valores da matriz de alternativas da Tabela 19, a razão entre as médias dos valores de cada critério, excluindo a alternativa 5, por ser muito discrepante.

Dessa forma, sugere-se o método WPM, o qual mostrou-se adequado para a classificação dos sistemas computacionais, mesmo sendo um método de baixa demanda computacional, pois as diferenças entre os valores das primeiras alternativas respeitou os pesos definidos na priorização dos critérios.

Por fim, a tarifação do uso dos componentes considerou tanto os componentes de *software* quanto os de *hardware*, combinando os custos unitários com a previsão do tempo de resolução dos componentes, assim, um modelo que considere a real utilização dos recursos é um trabalho a ser realizado futuramente. O novo modelo deve considerar tanto as quantidades inicialmente alocadas, quanto todas variações causadas pelo mecanismo de elasticidade do *Backend.*

6 CONCLUSÃO

Esta tese apresentou o Alite, um mecanismo para escolha de componentes de sistemas de computação paralela que leva em consideração a afinidade entre o *software*, representado por componentes de computação que implementam algoritmos paralelos, e o *hardware*, representado por componentes que representam plataformas virtuais para as quais tais algoritmos são desenvolvidos, bem como o alinhamento com políticas de alocação de recursos da própria plataforma de componentes subjacente. Esse mecanismo é aplicado sobre a plataforma HPC Shelf, voltada a oferta de serviços de construção e execução de sistemas de programação paralela de larga escala, ou seja, capaz de engajar um conjunto de plataformas de computação paralela possivelmente dispersas geograficamente e em diferentes infraestruturas, sejam nuvens de serviços IaaS ou plataformas físicas.

A concepção do Alite foi motivada pela ausência de uma solução que atenda a demandas de usuários de sistemas HPC que ainda causam relutância ao uso dos ambientes de nuvens computacionais destinados a tais sistemas (VECCHIOLA *et al.*, 2009; AHUJA; MANI, 2012; NETTO *et al.*, 2018). Nesse sentido, destacamos os relatórios anuais do projeto *UberCloud Experiment*[1] (GENTZSCH; YENIER, 2013; GENTZSCH; YENIER, 2014; GENTZSCH; YENIER, 2015; GENTZSCH; YENIER, 2016; GENTZSCH; YENIER, 2018), o qual tem avaliado, ao longo dos anos, o uso de sistemas HPC em nuvens computacionais através de um conjunto abrangente de estudos de caso com aplicações reais. Tais relatórios enumeram os seguintes fatores que dificultam a adoção da execução de aplicações dos usuários de HPC em ambientes de nuvens computacionais:

1. a dificuldade em prever os custos da execução;
2. a falta de controle sobre a localização geográfica das plataformas de execução;
3. a segurança e o custo do envio dos dados até as plataformas;
4. a degradação de desempenho das plataformas devido à sobrecargas adicionadas pela virtualização do hardware;
5. o acesso aos dados resultantes da computação;
6. o gerenciamento de licenças de software.

A partir dos fatores identificados, definimos as questões de pesquisa apresentadas no Capítulo 1, que tinham como objetivo nortear a concepção de um ambiente de nuvem que transpassasse os obstáculos que representam a dificuldade na adoção do modelo de nuvens

[1] <https://community.theubercloud.com/hpc-experiment>

computacionais pelos usuários de HPC. As questões identificadas para abordagem nesta Tese são restritas às características do processo de escolha de componentes de *software* e *hardware*, pois a HPC Shelf é uma contribuição coletiva do grupo de pesquisa em Computação de Alto Desempenho vinculado ao Programa de Pós-Graduação em Ciência da Computação (MDCC) da Universidade Federal do Ceará.

Assim, esta Tese de Doutorado tem como principal contribuição a concepção de uma estratégia de escolha de componentes voltada a ambientes CBHPC (Computação de Alto Desempenho Baseada em Componentes) baseados em nuvem, chamada Alite de modo que a escolha ocorra de forma automática, tanto para os componentes que representam *software* quanto para os componentes que representam as plataformas de execução e, através um conjunto de restrições, possibilite a escolha das melhores combinações de componentes de acordo com as prioridades do ambiente de nuvem e/ou do usuário. Essa estratégia orientada a componentes permite que a evolução dos recursos de *hardware* sejam incorporadas de forma natural, sem a necessidade de extensão do Alite.

Partiu-se de uma questão geral, sobre a viabilidade de utilizar ambientes de nuvens computacionais para executar aplicações de alto desempenho por parte dos usuários convencionais de HPC. Como resultado deste trabalho, pode-se afirmar que a resposta para essa pergunta é afirmativa. Através da concepção e do desenvolvimento da HPC Shelf por parte de esforços coletivos, conseguiu-se conceber um ambiente de nuvem que rompe as barreiras identificadas como empecilho por parte dos usuários de HPC. Em comparação às demais inciativas que vem sendo desenvolvidas, a HPC Shelf foi além e rompeu a maioria das restrições fortes sobre o conjunto restrito de aplicações e/ou de plataformas de execução, observado na maioria dos trabalhos relacionados.

Além disso, identificamos subquestões relacionadas a esta questão, as quais estão elencadas a seguir.

1. Quais os fatores que atualmente inviabilizam a adoção das nuvens computacionais pelos usuários de HPC ?
2. Como minimizar ou evitar os fatores identificados?
3. Como proporcionar uma escolha de componentes que respeite todas as restrições dos envolvidos no processo de execução, avaliando as métricas de desempenho e custo para guiar as escolhas?

Para responder essas questões, foram estudadas iniciativas existentes com o objetivo

de prover ambientes de nuvens computacionais para HPC, bem como iniciativas que objetivassem apenas diminuir a distância entre provedores de nuvens computacionais e usuários de HPC (NETTO *et al.*, 2018). Para essa finalidade, durante a execução desta pesquisa, os relatórios do projeto Ubercloud Experiment tornaram-se importantes guias. O Alite oferece as seguintes respostas aos principais pontos, enumerados anteriormente, que dificultam a adoção de nuvens dentro do contexto HPC, respectivamente:

1. Através de modelos de desempenho e de custo incorporados de forma integrada aos contratos contextuais, a previsão do custo de execução de sistemas de computação paralela pode ser mais precisa. Além disso, através de um sistema de reputação incorporado ao mecanismo de classificação, desenvolvedores de componentes e mantenedores de plataformas são naturalmente motivados a desenvolver modelos mais precisos, o quanto seja possível, dentro das restrições do sistema de contratos, a fim de que os recursos que disponibilizam na nuvem (e.g. HPC Shelf) sejam selecionados e seus contratos sejam obedecidos.
2. Parâmetros de contexto podem ser utilizados para restringir o escopo de localização geográfica para instanciação de plataformas virtuais. Essa possibilidade é reforçada tendo em vista que os grandes provedores IaaS, tais como Amazon EC2, Google Cloud Platform e Microsoft Azure, já incluem a localização geográfica como uma restrição para instanciação de máquinas virtuais.
3. O custo de desempenho no envio de grandes massas de dados a plataformas virtuais é inerente a sistemas HPC em nuvens computacionais. Entretanto, a própria HPC Shelf oferece mecanismos para tratar desse problema, permitindo o desenvolvimento de componentes de ligação customizados para lidar com segurança de dados sensíveis e eficiência de comunicação, evitando a comunicação desnecessária e permitindo a sobreposição entre computação e comunicação de acordo com o padrão de paralelismo adotado pelo sistema. Além disso, componentes conectores podem ser usados para levar a computação, total ou parcialmente, ao local onde os dados estão armazenados (fontes de dados), buscando reduzir o impacto da comunicação.
4. A virtualização do hardware é um outro aspecto inerente de sistemas HPC em nuvens. Porém, mais uma vez, a HPC Shelf oferece meios para lidar com a sobrecarga da virtualização. No caso da criação de plataformas virtuais, mantenedores podem manter uma *cache* de máquinas virtuais, de acordo com a demanda, as quais podem ser utilizadas e

reutilizadas evitando a sobrecarga da sua inicialização a cada execução de um sistema de computação paralela, de forma transparente ao sistema. Além disso, não há restrição que impeça que máquinas físicas possam ser utilizadas diretamente como plataformas virtuais na HPC Shelf, sendo uma alternativa de serviço que pode ser oferecido por mantenedores. Para isso, parâmetros de contexto podem ser utilizados para restringir o uso da virtualização.

5. Componentes especializados em visualização de dados recebidos através de componentes especializados de ligação entre o componente application e os componentes de solução, podem ser empregados para facilitar o acesso aos dados resultantes do processamento de sistemas de computação paralela por parte das aplicações.

A segunda questão de pesquisa refere-se à forma como representar os recursos diferentes dos provedores de plataformas de execução, possivelmente heterogêneos. Para isso, o sistema de contratos contextuais possibilita a modelagem de características arquiteturais de plataforma virtuais através de seus parâmetros de contexto. Tal expressividade foi explorada nos estudos de caso das seções 5.1 e 5.2, com a definição de categorias de plataforma através da modelagem de conjunto abrangente de suposições arquiteturais. Com base em perfis de plataformas virtuais, o nome específico dado a contratos contextuais de componentes da espécie plataforma virtual na HPC Shelf, sistemas de computação paralela podem guiar a escolha de plataformas virtuais adequadas para os interesses da aplicação e de acordo com a afinidade de componentes de computação por plataformas virtuais que atendam às suas suposições de implementação. Nesse contexto, os parâmetros de qualidade e custo, introduzidos pelo Alite, desempenham um papel importante, permitindo realizar o casamento entre modelos de desempenho e custo de componentes de computação e plataformas virtuais, sob a perspectiva de componentes sistema, lidando com características heterogêneas de plataformas modernas de computação paralela.

Com as respostas das questões anteriores, pode-se concluir que o Alite possibilita que aplicações possam ser modeladas para serem executadas na HPC Shelf, ou outro ambiente de nuvem para CBHPC, respeitando todos os requisitos dos usuários de HPC, e diferente das demais inciativas em desenvolvimento, sem impor limites de quais aplicações estão disponíveis ou em que conjunto de plataformas físicas podem ser executadas. Assim, respondendo à ultima questão de pesquisa que questionava se o Alite seria capaz de sobrepor as dificuldades identificadas pelos usuários de HPC.

6.1 Trabalhos Futuros

No processo de concepção e desenvolvimento do Alite, foram identificados problemas que consideramos oportunidades de novas funcionalidades a serem introduzidas no Alite em futuras extensão. Nesta seção, são apresentados alguns desses problemas identificados, aqueles que podem constituir motivação para trabalhos de pesquisa adjacentes.

6.1.1 Parâmetros de Negação

Na modelagem atual da HPC Shelf, os parâmetros de contexto permitem a representação de listas brancas (*white lists*) de valores para filtragem do conjunto de componentes que os satisfazem. Para todos os demais valores, os componentes são rejeitados. No processo de avaliação do Alite, identificou-se como uma limitação a ausência de parâmetros de contexto que representassem listas negras (*black list*), constituindo uma restrição de negação, de modo que o parâmetro de negação poderia representar um intervalo de valores cujos componentes que os satisfazem devem ser rejeitados pelo algoritmo de resolução.

Os parâmetros de negação tem especial impacto sobre aspectos legais, uma reivindicação dos usuários de HPC no que se refere à localização da nuvem computacional, pois, conforme descrito na Seção 2.6, existem aplicações que não podem ser executadas em determinadas regiões. Dessa forma, um exemplo do uso desse recurso seria poder definir que o contrato não pode ser executado em determinada cidade, país ou até continente.

Deve ser estudado o efeito da inclusão deste tipo de restrição no Alite, a fim de garantir a preservação da segurança de tipos.

6.1.2 Modelo Dinâmico de Tarifação

Para os estudos de caso apresentados nesta Tese, o modelo de tarifação aplica-se em tempo de escolha de componentes, a fim de permitir a noção de previsão de custo para a execução de tarefas na HPC Shelf. No entanto, com a incorporação de mecanismos de elasticidade tais quais os propostos por Alencar (2017) (ALENCAR; Carvalho Junior, 2019), a HPC Shelf deve possuir um mecanismo de tarifação que considere dinamicamente o custo da execução para ser utilizada como um ambiente de nuvem computacional pública. Portanto, a modelagem do modelo de tarifação dinâmica incorporado ao Alite e ao *Backend* é um trabalho a ser desenvolvido. Juntamente com a tarifação, contribuiria para o próximo trabalho identificado:

o uso de histórico de execução dos componentes.

6.1.3 Classificação Baseada em Histórico de Execução

A tarifação calculada na resolução da etapa de seleção do Alite é um dos parâmetros considerados no processo de classificação dos sistemas computacionais. Usar o histórico de valores de parâmetros como o custo final da execução, é uma forma de ajustar o Alite para ter resultados mais precisos no momento da classificação dos resultados. Dessa forma, sugere-se como trabalho futuro a inclusão de métodos de classificação evolutivos, baseados no histórico das execuções anteriores.

6.1.4 Métodos MCDM Hierárquicos

Uma outra mudança no processo de classificação que pode ser explorada é a mudança dos métodos atuais por métodos MCDM hierárquicos, como é o caso do AHP, que considerem a existência de critérios e subcritérios, assim ampliando o alcance da etapa de classificação.

REFERÊNCIAS

AHUJA, S. P.; MANI, S. The State of High Performance Computing in the Cloud. **Journal of Emerging Trends in Computing and Information Sciences**, 2012.

ALENCAR, J. M. U.; Carvalho Junior, F. H. de. On the Elasticity of Parallel Components in a Cloud of High Performance Computing Services. In: **2019 Symposium on High Performance Computing Systems (WSCAD'2019)**. Campo Grande, MT: SBC, 2019.

ALENCAR, J. M. U. d. **Reconfiguração Elástica de Componentes Paralelos em Nuvens de Serviços de Computação de Alto Desempenho**. Doutorado — Programa de Pós-Graduação em Ciência da Computação, 2017.

ALKHAMISI, A.; QURESHI, M. R. A Proposal of Case Based Reasoning System for the Appropriate Selection of Components Using CBD. **International Journal of Information Technology and Computer Science**, v. 5, p. 43–55, 08 2013.

ALLAN, B. A.; ARMSTRONG, R. C.; WOLFE, A. P.; RAY, J.; BERNHOLDT, D. E.; KOHL, J. A. The CCA Core Specification in a Distributed Memory SPMD Framework. **Concurrency and Computation: Practice and Experience**, Wiley, v. 14, n. 5, p. 323–345, 2002.

AMALARETHINAM, D. G.; BEENA, T. L. A. Cloud Scheduling - A Survey. **International Journal of Computer Applications**, Foundation of Computer Science, v. 97, n. 13, 2014.

AMEDRO, B.; BAUDE, F.; D. Caromel; DELBE, C.; FILALI, I.; HUET, F.; MATHIAS, E.; O. Smirnov. An Efficient Framework for Running Applications on Clusters, Grids and Clouds. In: **Cloud Computing: Principles, Systems and Applications**. London: Springer, 2010. chapter 10, p. 163–178.

ANDERSON, T. E.; CULLER, D. E.; PATTERSON, D. A.; TEAM the N. A Case for Networks of Workstations. **IEEE Micro**, v. 15, n. 1, p. 54–64, fev. 1995.

ARMSTRONG, R.; GANNON, D.; GEIST, A.; KEAHEY, K.; KOHN, S.; MCINNES, L.; PARKER, S.; SMOLINSKI, B. Toward a Common Component Architecture for High-Performance Scientific Computing. In: **Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing**. Washington, DC, USA: IEEE Computer Society, 1999. (HPDC '99). ISBN 0-7695-0287-3. Disponível em: http://dl.acm.org/citation.cfm?id=822084.823232. Acesso em: 10 setembro 2012.

ARMSTRONG, R.; KUMFERT, G.; MCINNES, L. C.; PARKER, S.; ALLAN, B.; SOTTILE, M.; EPPERLY, T.; TAMARA, D. The CCA Component Model For High-Performance Scientific Computing. **Concurrency and Computation: Practice and Experience**, Wiley, v. 18, n. 2, p. 215–229, 2006.

Bailey, D. H.; Barszcz, E.; Barton, J. T.; Browning, D. S.; Carter, R. L.; Dagum, L.; Fatoohi, R. A.; Frederickson, P. O.; Lasinski, T. A.; Schreiber, R. S.; Simon, H. D.; Venkatakrishnan, V.; Weeratunga, S. K. The nas parallel benchmarks summary and preliminary results. In: **Supercomputing '91:Proceedings of the 1991 ACM/IEEE Conference on Supercomputing**. Albuquerque, NM, USA: IEEE, 1991. p. 158–165.

BAKER, M.; BUYYA, R.; HYDE, D. Cluster Computing: A High Performance Contender. **IEEE Computer**, v. 42, n. 7, p. 79–83, jul. 1999.

BARRACHINA, S.; CASTILLO, M.; IGUAL, F. D.; MAYO, R.; QUINTANA-ORTI, E. S. Evaluation and Tuning of the Level 3 CUBLAS for Graphics Processors. In: **2008 IEEE International Symposium on Parallel and Distributed Processing**. Miami, FL, USA: IEEE, 2008. p. 1–8. ISSN 1530-2075.

BAUDE, F.; CAROMEL, D.; DALMASSO, C.; DANELUTTO, M.; GETOV, W.; HENRIO, L.; PREZ, C. GCM: A Grid Extension to Fractal for Autonomous Distributed Components. **Annals of Telecommunications**, v. 64, n. 1, p. 5–24, 2009.

BAXTER, R.; BOOTH, S.; BULL, M.; CAWOOD, G.; PERRY, J.; PARSONS, M.; SIMPSON, A.; TREW, A.; MCCORMICK, A.; SMART, G.; SMART, R.; CANTLE, A.; CHAMBERLAIN, R.; GENEST, G. The FPGA High-Performance Computing Alliance Parallel Toolkit. In: **Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on**. Los Alamos, CA: IEEE Computer Society, 2007. p. 301 –310.

BECKER, D. J.; STERLING, T.; SAVARESE, D.; DORBAN, J. E.; RANAWAK, U. A.; PACKER, C. V. Bewoulf: A Parallel Workstation for Scientific Computation. In: **1995 International Conference on Parallel Processing**. Boca Raton, FL, USA: CRC Press, Inc., 1995.

BELUSSO, C. M.; SAWICKI, S.; BASTO-FERNANDES, V.; FRANTZ, R. Z.; ROOS-FRANTZ, F. A Proposal of Infrastructure-as-a-Service Providers Pricing Model Using Linear Regression. **Revista Brasileira de Computação Aplicada**, v. 10, n. 2, p. 44–53, jul. 2018.

BEN-YEHUDA, O. A.; BEN-YEHUDA, M.; SCHUSTER, A.; TSAFRIR, D. The Resource-as-a-Service (RaaS) Cloud. In: **Proceedings of the 4th USENIX conference on Hot Topics in Cloud Computing**. Berkeley, CA, USA: USENIX Association, 2012. (HotCloud'12), p. 12–12.

BERTRAND, F.; BRAMLEY, R. DCA: A Distributed CCA Framework Based on MPI. In: **Proceedings of the 9th International Workshop on Highlevel Parallel Programming Models and Supportive Environments (HIPS'2004)**. Santa Fe, New Mexico: IEEE Computer Society, 2004. ISBN 0-7695-2151-7.

BLACKFORD, L. S.; CHOI, J.; CLEARY, A.; PETITET, A.; WHALEY, R. C.; DEMMEL, J.; DHILLON, I.; STANLEY, K.; DONGARRA, J.; HAMMARLING, S.; HENRY, G.; WALKER, D. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. In: **Proceedings of the 1996 ACM/IEEE Conference on Supercomputing**. Washington, DC, USA: IEEE Computer Society, 1996. (Supercomputing '96). ISBN 0-89791-854-1.

BLAIR, G.; COUPAYE, T.; STEFANI, J.-B. Component-Based Architecture: The Fractal Initiative. **Annals of Telecommunications**, Springer Paris, v. 64, p. 1–4. ISSN 0003-4347.

BOARD, O. A. R. **OpenMP: Simple, Portable, Scalable SMP Programming**. 1997. Disponível em: www.openmp.org. Acesso em: 12 Junho 2019.

BRUNETON, E.; COUPAYE, T.; LECLERCQ, M.; QUMA, V.; STEFANI, J.-B. The Fractal Component Model and Its Support In Java. **Software - Practice and Experience**, Wiley, v. 36, p. 1257–1284, 2006.

BUYYA, R.; BELOGLAZOV, A.; ABAWAJY, J. H. Energy-Efficient Management of Data Center Resources for Cloud Computing: A Vision, Architectural Elements, and Open Challenges. **CoRR**, abs/1006.0308, 2010.

BUYYA, R.; BROMBERG, J.; GOSCINSKI, A. **Cloud Computing Principles and Paradigms**. Danvers, MA: Wiley, 2011.

BUYYA, R.; GARG, S. K.; CALHEIROS, R. N. SLA-oriented Resource Provisioning for Cloud Computing: Challenges, Architecture, and Solutions. In: **Proceedings of the 2011 International Conference on Cloud and Service Computing**. Washington, DC, USA: IEEE Computer Society, 2011. (CSC '11), p. 1–10. ISBN 978-1-4577-1635-5.

CALEGARI, P.; LEVRIER, M.; BALCZYńSKI, P. Web Portals for High-performance Computing: A Survey. **ACM Trans. Web**, ACM, New York, NY, USA, v. 13, n. 1, p. 5:1–5:36, fev. 2019. ISSN 1559-1131.

CALHEIROS, R. N.; RANJAN, R.; BELOGLAZOV, A.; ROSE, C. A. F. D.; BUYYA, R. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. **Softw. Pract. Exper.**, John Wiley & Sons, Inc., New York, NY, USA, v. 41, n. 1, p. 23–50, jan. 2011. ISSN 0038-0644.

CARVALHO-JUNIOR, F. H.; CORREA, R. C. The Design of a CCA Framework with Distribution, Parallelism, and Recursive Composition. In: **Workshop on Component-Based High Performance Computing (CBHPC'2010)**. Brussels, Belgium: IEEE, 2010. p. 339–348. ISBN 9781424493470.

CARVALHO-JUNIOR, F. H. de; CORREA, R. C.; LINS, R.; SILVA, J. C.; ARAÚJO, G. A. High Level Service Connectors for Components-Based High Performance Computing. In: **Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing**. Gramado - RS - Brazil: IEEE, 2007. p. 237–244.

CARVALHO-JUNIOR, F. H. de; LINS, R. D. Haskell$_{\#}$: Parallel Programming Made Simple and Efficient. **Journal of Universal Computer Science**, Springer, v. 9, n. 8, p. 776–794, ago. 2003.

CARVALHO-JUNIOR, F. H. de; LINS, R. D. Separation of Concerns for Improving Practice of Parallel Programming. **INFORMATION, An International Journal**, International Information Institute, v. 8, n. 5, p. 621–638, set. 2005. ISSN 1343-4500.

CARVALHO-JUNIOR, F. H. de; LINS, R. D. An Institutional Theory for #-Components. **Electronic Notes in Theoretical Computer Science**, Springer Verlag, v. 195, p. 113–132, jan. 2008.

CARVALHO-JUNIOR, F. H. de; LINS, R. D.; CORRêA, R. C.; ARAúJO, G. A. Towards an Architecture for Component-Oriented Parallel Programming. **Concurrency and Computation: Practice and Experience**, John Wiley and Sons Ltd., Chichester, UK, v. 19, n. 5, p. 697–719, abr. 2007. ISSN 1532-0626.

CARVALHO-JUNIOR, F. H. de; REZENDE, C. A. A Case Study on Expressiveness and Performance of Component-Oriented Parallel Programming. **Journal of Parallel and Distributed Computing**, v. 73, n. 5, p. 557–569, maio 2013. ISSN 0743-7315.

CARVALHO-JUNIOR, F. H. de; REZENDE, C. A.; SILVA, J. C.; ALAM, W. G. Al; ALENCAR, J. M. U. Contextual Abstraction in a Type System for Component-Based High Performance Computing Platforms. **Science of Computer Programming**, v. 132, n. 1, p. 96–128, 2016. ISSN 0167-6423.

CASTRO, A. de; LIBORIO, J.; PANDOLFELLI, V. A Influência do Tipo de Cimento no Desempenho de Concretos Avançados Formulados a partir do Método de Dosagem Computacional. **Cerâmica**, SciELO Brasil, v. 57, p. 10–21, 2011.

CHOI, J.; DEMMEL, J.; DHILLON, I.; DONGARRA, J.; OSTROUCHOV, S.; PETITET, A.; STANLEY, K.; WALKER, D.; WHALEY, R. C. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers — Design Issues and Performance. In: DONGARRA, J.; MADSEN, K.; WAŚNIEWSKI, J. (Ed.). **Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science**. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. p. 95–106. ISBN 978-3-540-49670-0.

CHOO, K.-K. R. Cloud Computing: Challenges and Future Directions. **Trends e issues in crime and criminal justice**, 2010.

CLEMENTS, P.; NORTHROP, L. **Software Product Lines: Patterns and Practice**. Reading, MA: Addison Wesley, 2001.

COLE, M. **Algorithm Skeletons: Structured Management of Paralell Computation**. London: Pitman, 1989.

COLE, M. Bringing Skeletons out of the Closet: A Pragmatic Manifesto for Skeletal Parallel Programming. **Parallel Computing**, v. 30, n. 3, p. 389–406, 2004.

COOK, S. **CUDA Programming: A Developer's Guide to Parallel Computing with GPUs** . Waltham - MA, USA: Morgan Kaufmann, 2012. ISBN 9780124159334.

CRNKOVIC, I.; SENTILLES, S.; VULGARAKIS, A.; CHAUDRON, M. A Classification Framework for Software Component Models. **Software Engineering, IEEE Transactions on**, v. 37, n. 5, p. 593 –615, sept.-oct. 2011. ISSN 0098-5589.

DANTAS, A. B. d. O. **Certificação de Componentes em uma Plataforma de Nuvens Computacionais para Serviços de Computação de Alto Desempenho**. Doutorado — Programa de Pós-Graduação em Ciência da Computação, 2017.

de Carvalho Junior, F. H.; SILVA, J. C.; DANTAS, A. B. O. A Scientific Workflow Management System for Orchestration of Parallel Components in a Cloud of Large-Scale Parallel Processing Services. **Science of Computer Programming**, v. 173, p. 95–127, mar. 2019. ISSN 0167-6423.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. In: **Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6**. Berkeley, CA, USA: USENIX Association, 2004. (OSDI'04), p. 10–10.

DEAN, J.; GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. **Commun. ACM**, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782.

DEAN, J.; GHEMAWAT, S. MapReduce: A Flexible Data Processing Tool. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 1, p. 72–77, jan. 2010. ISSN 0001-0782.

DOERNER, W. **Application Analysis for Intel® MIC Architecture Suitability**. 2012. Disponível em: http://software.intel.com/en-us/articles/application-analysis-for-intel-mic-architecture-suitability. Acesso em: 10 setembro 2012.

DONGARRA, J. Basic Linear Algebra Subprograms Technical Forum Standard I. **International Journal of High Performance Applications and Supercomputing**, v. 16, n. 2, p. 115–199, feb 2002. ISSN 1094-3420.

DONGARRA, J.; OTTO, S. W.; SNIR, M.; WALKER, D. A Message Passing Standard for MPP and Workstation. **Communications of ACM**, v. 39, n. 7, p. 84–90, 1996.

DONGARRA, J. J.; CROZ, J. D.; HAMMARLING, S.; DUFF, I. S. A Set of Level 3 Basic Linear Algebra Subprograms. **ACM Transactions on Mathematical Software (TOMS)**, ACM, v. 16, n. 1, p. 1–17, 1990.

DURAN, A.; KLEMM, M. The Intel® Many Integrated Core Architecture. In: **2012 International Conference on High Performance Computing and Simulation (HPCS)**. Madrid, Spain: IEEE Computer Society, 2012. p. 365–366. ISBN 978-1-4673-2359-8.

DYER, J. S.; FISHBURN, P. C.; STEUER, R. E.; WALLENIUS, J.; ZIONTS, S. Multiple Criteria Decision Making, Multiattribute Utility Theory: The Next Ten Years. **Management Science**, v. 38, n. 5, p. 645–654, 1992.

ECMA International. **Common Language Infrastructure (CLI), Partitions I to VI**. 4. ed. Geneva, 2006. Disponível em: http://www.ecma-international.org/publications/standards/Ecma-335.htm. Acesso em 15 setembro 2019.

EKANAYAKE, J.; FOX, G. High Performance Parallel Computing with Clouds and Cloud Technologies. In: AVRESKY, D. R.; DIAZ, M.; BODE, A.; CICIANI, B.; DEKEL, E.; AKAN, O.; BELLAVISTA, P.; CAO, J.; DRESSLER, F.; FERRARI, D.; GERLA, M.; KOBAYASHI, H.; PALAZZO, S.; SAHNI, S.; SHEN, X. S.; STAN, M.; XIAOHUA, J.; ZOMAYA, A.; COULSON, G. (Ed.). **Cloud Computing**. Munich, Germany: Springer Berlin Heidelberg, 2010, (Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, v. 34). p. 20–38. ISBN 978-3-642-12636-9. 10.1007/978-3-642-12636-9_2.

ENGINE, G. C. **Google Compute Engine Website**. 2019. Disponível em: https://cloud.google.com/compute. Acesso em: 01 Junho 2019.

FAN, Z.; QIU, F.; KAUFMAN, A.; STOVER, S. Yoakum. GPU Cluster for High Performance Computing. In: **Proceedings of the 2004 ACM/IEEE conference on Supercomputing (SC'04)**. Washington, DC, USA: IEEE Computer Society, 2004. p. 47–47. ISBN 0-7695-2153-3.

FEITELSON, D. G.; RUDOLPH, L.; SCHWIEGELSHOHN, U.; SEVCIK, K. C.; WONG, P. Theory and Practice in Parallel Job Scheduling. In: SPRINGER. **Job scheduling strategies for parallel processing**. London, UK, UK, 1997. p. 1–34.

FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud Computing and Grid Computing 360-Degree Compared. In: **Grid Computing Environments Workshop, 2008. GCE '08**. Austin, TX, USA: IEEE, 2008. p. 1 –10.

FÜLÖP, J. Introduction to Decision Making Methods. In: CITESEER. **BDEI-3 Workshop**. Washington, 2005.

GARG, S.; VERSTEEG, S.; BUYYA, R. SMICloud: A Framework for Comparing and Ranking Cloud Services. In: **Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on**. Victoria, NSW, Australia: IEEE, 2011. p. 210–218.

GARG, S. K.; GOPALAIYENGAR, S. K.; BUYYA, R. SLA-based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter. In: **Proceedings of the 11th international conference on Algorithms and architectures for parallel processing - Volume Part I**. Berlin, Heidelberg: Springer-Verlag, 2011. (ICA3PP'11), p. 371–384. ISBN 978-3-642-24649-4.

GENTZSCH, W.; YENIER, B. **The UberCloud HPC Experiment: Compendium of Case Studies**. Los Altos, CA, USA, 2013.

GENTZSCH, W.; YENIER, B. **The UberCloud Experiment: Technical Computing in the Cloud - 2nd Compendium of Case Studies**. Los Altos, CA, USA, 2014.

GENTZSCH, W.; YENIER, B. **The UberCloud Experiment: Technical Computing in the Cloud - 3rd Compendium of Case Studies**. Los Altos, CA, USA, 2015.

GENTZSCH, W.; YENIER, B. **The UberCloud Experiment: Technical Computing in the Cloud - 4th Compendium of Case Studies**. Los Altos, CA, USA, 2016.

GENTZSCH, W.; YENIER, B. **The UberCloud Experiment: Technical Computing in the Cloud - 5th Compendium of Case Studies**. Los Altos, CA, USA, 2018.

GOTO, K.; GEIJN, R. A. v. d. Anatomy of High-Performance Matrix Multiplication. **ACM Trans. Math. Softw.**, ACM, New York, NY, USA, v. 34, n. 3, p. 12:1–12:25, maio 2008. ISSN 0098-3500.

GOVINDARAJU, M.; LEWIS, M. J.; CHIU, K. Design and Implementation Issues for Distributed CCA Framework Interoperability. **Concurrency and Computation: Practice and Experience**, John Wiley and Sons Ltd., Chichester, UK, v. 19, n. 5, p. 651–666, abr. 2007. ISSN 1532-0626.

GRAMA, A. **Introduction to Parallel Computing**. Boston, MA, USA: Addison-Wesley, 2003. (Pearson Education). ISBN 9780201648652.

GUO, C.; WU, H.; TAN, K.; SHIY, L.; ZHANG, Y.; LUZ, S. MapReduce: Simplified Data Processing on Large Clusters. **Proc. of OSDI 2004, San Francisco, CA, USA**, 2004.

GUPTA, A.; FARABOSCHI, P.; GIOACHIN, F.; KALE, L. V.; KAUFMANN, R.; LEE, B. S.; MARCH, V.; MILOJICIC, D.; SUEN, C. H. Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud. **IEEE Transactions on Cloud Computing**, v. 4, n. 3, p. 307–321, July 2016. ISSN 2168-7161.

HALL, M. W.; GIL, Y.; LUCAS, R. Self-Configuring Applications for Heterogeneous Systems: Program Composition and Optimization Using Cognitive Techniques. **Proceedings of the IEEE**, v. 96, n. 5, p. 849–862, 2008.

HAREL, D.; PNUELI, A. On the Development of Reactive Systems. In: APT, K. (Ed.). **Logics and Models of Concurrent Systems**. Heidelberg, Berlin: Springer Berlin Heidelberg, 1985, (NATO ASI Series, v. 13). p. 477–498.

HERBORDT, M. C.; VANCOURT, T.; GU, Y.; SUKHWANI, B.; CONTI, A.; MODEL, J.; DISABELLO, D. Achieving High Performance with FPGA-Based Computing. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 40, p. 50–57, March 2007. ISSN 0018-9162.

HUANG, K.-C.; HUANGA, T.-C.; CHANG, M.-J. T. H.-Y.; TUNG, Y.-H. Moldable Job Scheduling for HPC as a Service with Application Speedup Model and Execution Time Information. **Journal of Convergence**, v. 4, n. 4, 2013.

JHA, S.; KATZ, D. S.; LUCKOW, A.; MERZKY, A.; STAMOU, K. Understanding Scientific Applications for Cloud Environments. In: **Cloud Computing**. Hoboken, United States: John Wiley & Sons, Inc., 2011. p. 345–371. ISBN 9780470940105.

JUNIOR, F. H. de Carvalho; LINS, R.; CORREA, R. C.; ARAÚJO, G. A. Towards an Architecture for Component-Oriented Parallel Programming. **Concurrency and Computation: Practice and Experience**, v. 19, n. 5, p. 697–719, 2007. ISSN 1532-0626.

KäCHELE, S.; DOMASCHKA, J.; HAUCK, F. J. COSCA: an Easy-to-use Component-based PaaS Cloud System for Common Applications. In: **Proceedings of the First International Workshop on Cloud Computing Platforms**. New York, NY, USA: ACM, 2011. (CloudCP '11), p. 4:1–4:6. ISBN 978-1-4503-0727-7.

KHAN, A.; KHAN, K.; AMIR, M.; KHAN, M. A Component-Based Framework for Software Reusability. **International Journal of Software Engineering and its Applications**, v. 8, p. 13–24, 01 2014.

KOç Çetin K.; GAN, S. C. Parallel Matrix Multiplication on Networked Microcomputers. **Computers e Electrical Engineering**, v. 18, n. 2, p. 145 – 152, 1992. ISSN 0045-7906.

KRISHNAN, S.; GANNON, D. XCAT3: A Framework for CCA Components as OGSA Services. In: **Proceedings of the HIPS2004 - 9th International Workshop on Highlevel Parallel Programming Models and Supportive Environments**. Santa Fe, NM, USA: IEEE, 2004.

KUCHEN, H.; COLE, M. Algorithm Skeletons. **Parallel Computing**, Elsevier, v. 32, n. 7–8, p. 447–626, 2006.

KWONG, C.; MU, L.; TANG, J.; LUO, X. Optimization of Software Components Selection for Component-Based Software System Development. **Computers and Industrial Engineering**, v. 58, n. 4, p. 618 – 624, 2010. ISSN 0360-8352.

LEE, S.; MEREDITH, J. S.; VETTER, J. S. COMPASS: A Framework for Automated Performance Modeling and Prediction. In: **Proceedings of the 29th ACM on International Conference on Supercomputing**. New York, NY, USA: ACM, 2015. (ICS '15), p. 405–414. ISBN 978-1-4503-3559-1.

LI, A.; YANG, X.; KANDULA, S.; ZHANG, M. CloudCmp: Comparing Public Cloud Providers. In: **Proceedings of the 10th ACM SIGCOMM conference on Internet measurement**. New York, NY, USA: ACM, 2010. (IMC '10), p. 1–14. ISBN 978-1-4503-0483-2.

LI, R.; HU, H.; LI, H.; WU, Y.; YANG, J. MapReduce Parallel Programming Model: A State-of-the-Art Survey. **International Journal of Parallel Programming**, v. 44, n. 4, p. 832–866, 2016. ISSN 1573-7640.

LOW, T. M.; IGUAL, F. D.; SMITH, T. M.; QUINTANA-ORTI, E. S. Analytical Modeling Is Enough for High-Performance BLIS. **ACM Trans. Math. Softw.**, ACM, New York, NY, USA, v. 43, n. 2, p. 12:1–12:18, ago. 2016. ISSN 0098-3500.

LOWY, J. **Programming .NET Components: Design and Build .NET Applications Using Component-Oriented Programming**. Sebastopol, CA, USA: O'Reilly Media, 2005. ISBN 9780596553661.

MALAWSKI, M.; BUBAK, M.; BAUDE, F.; CAROMEL, D.; HENRIO, L.; MOREL, M. Interoperability of Grid Component Models: GCM and CCA Case Study. In: **CoreGRID'07**. Boston, MA, USA: Springer US, 2007. p. 95–105.

MALAWSKI, M.; KURZYNIEC, D.; SUNDERAM, V. MOCCA - Towards a Distributed CCA Framework for Metacomputing. In: **Proceedings of the Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models (HIPS-HPGC)**. Denver, USA: IEEE Computer Society, 2005. ISBN 0-7695-2312-9.

MALAWSKI, M.; MEIZNER, J.; BUBAK, M.; GEPNER, P. Component Approach to Computational Applications on Clouds. **Procedia Computer Science**, v. 4, n. 0, p. 432 – 441, 2011. ISSN 1877-0509.

MASUD, R. High Performance Computing with Clouds. Disponível em: http://www.pdfio.com/k-875146.html. Acesso em 15 setembro 12. 2011.

MATEO, J. R. S. C. **Multicriteria Analysis in the Renewable Energy Industry**. London: Springer Science & Business Media, 2012.

MICROSOFT. **Microsoft .NET Framework**. 2012. Disponível em: http://www.microsoft.com/net. Acesso em 11 setembro 2012.

MICROSOFT. **Microsoft Azure Website**. 2019. Disponível em: https://azure.microsoft.com/pt-br/pricing/details/virtual-machine-scale-sets. Acesso em: 01 Junho 2019.

MONTEIRO, M. **Introdução à Organização de Computadores**. Brasil: Livros Técnicos e Científicos, 2002. ISBN 9788521612919.

MOORE, G. E. Cramming More Components onto Integrated Circuits. In: **Electronics Magazine**. Mountainview, CA, USA: IEEE, 1965.

NAKAJIMA, J.; PALLIPADI, V. Enhancements for Hyper-Threading Technology in the Operating System: Seeking the Optimal Scheduling. In: **Proceedings of the 2nd conference on Industrial Experiences with Systems Software - Volume 2**. Berkeley, CA, USA: USENIX Association, 2002. (WIESS'02), p. 3–3.

NAPPER, J.; BIENTINESI, P. Can Cloud Computing Reach the Top500 ? In: **Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop**. New York, NY, USA: ACM, 2009. (UCHPC-MAW '09), p. 17–20. ISBN 978-1-60558-557-4.

NETTO, M. A. S.; CALHEIROS, R. N.; RODRIGUES, E. R.; CUNHA, R. L. F.; BUYYA, R. HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. **ACM Computing Surveys**, ACM, New York, NY, USA, v. 51, n. 1, p. 1–29, jan. 2018. ISSN 0360-0300.

NIEHORSTER, O.; BRINKMANN, A.; FELS, G.; KRUGER, J.; SIMON, J. Enforcing SLAs in Scientific Clouds. In: **Cluster Computing (CLUSTER), 2010 IEEE International Conference on**. Heraklion, Crete, Greece: IEEE, 2010. p. 178–187.

NULL, L.; LOBUR, J. **Princípios Básicos de Arquitetura e Organização de Computadores**. Tall Pine Drive, Sudbury, MA: Bookman, 2010. ISBN 9788577807666.

OBAIDA, M. A.; LIU, J.; CHENNUPATI, G.; SANTHI, N.; EIDENBENZ, S. Parallel Application Performance Prediction Using Analysis Based Models and HPC Simulations. In: **Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation**. New York, NY, USA: ACM, 2018. (SIGSIM-PADS '18), p. 49–59. ISBN 978-1-4503-5092-1.

OMG. **CORBA Component Model Specification, Version 4.0**. Needham, MA, USA, 2006.

PANDE, J.; GARCIA, C. J.; PANT, D. Optimal Component Selection for Component-Based Software Development Using Pliability Metric. **SIGSOFT Software Engineering Notes**, ACM, New York, NY, USA, v. 38, n. 1, p. 1–6, jan. 2013. ISSN 0163-5948.

PARASHAR, M.; LI, X.; PARKER, S. G.; DAMEVSKI, K.; KHAN, A.; SWAMINATHAN, A.; JOHNSON, C. R. Advanced Computational Infrastructures for Parallel/Distributed Adaptive Applications. In: PARASHAR, M.; LI, X. (Ed.). Danvers, MA, USA: Wiley Press, 2009. cap. The SCIJump Framework for Parallel and Distributed Scientific Computing.

PEPPLE, K. **Deploying OpenStack Creating Open Source Clouds**. Sebastopol, CA, USA: O'Reilly Media, 2011.

Pezy Super Computing. **Pezy**. 2015. Disponível em: http://pezy.co.jp. Acesso em 15 setembro 2015.

POHL, D. **Experimental Cloud-based Ray Tracing Using Intel® MIC Architecture for Highly Parallel Visual Processing**. 2012. Disponível em: http://software.intel.com/en-us/articles/cloud-based-ray-tracing/?wapkw=pohl. Acesso em: 10 setembro 2012.

POST, D. E.; VOTTA, L. G. Computational Science Demands a New Paradigm. **Physics Today**, v. 58, n. 1, p. 35–41, 2005.

RABHI, F. A.; GORLATCH, S. **Patterns and Skeletons for Parallel and Distributed Computing**. London, England: Springer, 2002.

REZENDE, C. A.; Carvalho Junior, F. H. de. MapReduce with Components for Processing Big Graphs. In: **2018 Symposium on High Performance Computing Systems (WSCAD'2018)**. São Paulo, SP: SBC, 2018. p. 108–115.

REZENDE, C. A. d. **Um Arcabouço Baseado em Componentes para Computação Paralela de Larga Escala sobre Grafos**. Doutorado — Programa de Pós-Graduação em Ciência da Computação, 2017.

SALOMON, V. A. P. **Desempenho da Modelagem do Auxílio à Decisão por Múltiplos critérios na Análise do Planejamento e Controle da Produção**. Tese (Doutorado) — Universidade de São Paulo, 2004.

SANDHU, R.; TOOSI, A. N.; BUYYA, R. An API for Development of User-Defined Scheduling Algorithms in Aneka PaaS Cloud Software: User Defined Schedulers in Aneka PaaS Cloud Software. In: **Handbook of Research on Cloud Computing and Big Data Applications in IoT**. Hershey, PA: IGI Global, 2018.

SANDOVAL, J. A.; COOPER, K. D. Dynamic Compilation for Component-Based High Performance Computing. In: **Proceedings of the 2009 Workshop on Component-Based High Performance Computing**. New York, NY, USA: ACM, 2009. (CBHPC '09). ISBN 978-1-60558-718-9.

SERVICES, A. W. **Amazon Elastic Compute Cloud (Amazon EC2) Website**. 2019. Disponível em: http://aws.amazon.com. Acesso em: 12 Junho 2019.

SERVICES, A. W. **Documentação do Amazon Elastic Compute Cloud**. 2019. Disponível em: https://docs.aws.amazon.com/ec2/index.html. Acesso em: 07 Julho 2019.

SHAW, A. **Sistemas e Software de Tempo Real**. Porto Alegre, Brasil: BOOKMAN COMPANHIA ED, 2003. ISBN 9788536301723.

SHI, J.; TAIFI, M.; KHREISHAH, A. Resource Planning for Parallel Processing in the Cloud. In: **High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on**. Washington, DC, USA: IEEE Computer Society, 2011. p. 828–833.

SILVA, J. C.; CARVALHO-JUNIOR, F. H. de. A Platform of Scientific Workflows for Orchestration of Parallel Components in a Cloud of High Performance Computing Applications. In: **Lecture Notes in Computer Science**: Proceedings of the the Proceedings of the XX Brazilian Symposium on Programming Languages (SBLP'2016). Maringá, Brazil: Springer, 2016. v. 9889, p. 156–170.

SILVA, J. de C. **Um Arcabouço para a Construção de Aplicações Baseadas em Componentes sobre uma Plataforma de Nuvem Computacional para Serviços de Computação de Alto Desempenho**. Doutorado — Programa de Pós-Graduação em Ciência da Computação, 2016.

SOUSA, F. R.; MOREIRA, L. O.; MACHADO, J. C. Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios. In: EDUFPI. **III Escola Regional de Computação Ceará, Maranhão e Piauí. ERCEMAPI**. Parnaíba, PI, Brazil, 2009. p. 150–175.

STALLINGS, W. **Arquitetura e Organização de Computadores**. 5a ed. ed. Upper Siddle River, New Jersey, USA: Prentice Hall, 2003.

STEEN, A. J. van der. Issues in Computational Frameworks. **Concurrency and Computation: Practice and Experience**, John Wiley and Sons, Ltd., v. 18, n. 2, p. 141–150, 2006.

STONE, J. E.; GOHARA, D.; SHI, G. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. **Computing in Science and Engineering.**, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, n. 3, p. 66–73, maio 2010. ISSN 15219615.

STURM, R.; MORRIS, W.; JANDER, M. **Foundations of Service Level Management**. Indianapolis, United States: Sams, 2000. (Sams Professional Series). ISBN 9780672317439.

TANENBAUM, A.; STEEN, M. van. **Sistemas Distribuídos: Princípios e Paradigmas**. Brasil: Pearson Prentice Hall, 2007. ISBN 9788576051428.

THOMPSON, S. **Haskell, The Craft of Functional Programming**. USA: Addison-Wesley Publishers Ltd., 1996. ISBN 0201882957.

TOOSI, A. N.; SINNOTT, R.; BUYYA, R. Resource Provisioning for Data-Intensive Applications with Deadline Constraints on Hybrid Clouds Using Aneka. **Future Generation Computer Systems**, v. 79, p. 765 – 775, 2018. ISSN 0167-739X.

TOP500.ORG. **TOP 500 Supercomputer**. 2019. Disponível em: http://www.top500.org. Acesso em: 04 março 2019.

TRIANTAPHYLLOU, E.; SáNCHEZ, A. A Sensitivity Analysis Approach for Some Deterministic Multi-Criteria Decision-Making Methods. **Decision Sciences**, Blackwell Publishing Ltd, v. 28, n. 1, p. 151–194, 1997. ISSN 1540-5915.

TZENG, G.-H.; HUANG, J.-J. **Multiple Attribute Decision Making: Methods and Applications**. Boca Raton, FL-US: Chapman and Hall/CRC, 2011.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A Break in the Clouds: Towards a Cloud Definition. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, dez. 2008. ISSN 0146-4833.

VECCHIOLA, C.; PANDEY, S.; BUYYA, R. High-Performance Cloud Computing: A View of Scientific Applications. In: **Proceedings of the 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks**. Washington, DC, USA: IEEE Computer Society, 2009. (ISPAN '09), p. 4–16. ISBN 978-0-7695-3908-9.

VELASQUEZ, M.; HESTER, P. T. An Analysis of Multi-Criteria Decision Making Methods. **International Journal of Operations Research**, v. 10, n. 2, p. 56–66, 2013.

WANG, A.; QIAN, K. **Component-Oriented Programming**. New York, NY, USA: Wiley, 2005. ISBN 9780471713692.

WEBER, R.; GOTHANDARAMAN, A.; HINDE, R.; PETERSON, G. Comparing Hardware Accelerators in Scientific Applications: A Case Study. **Parallel and Distributed Systems, IEEE Transactions on**, v. 22, n. 1, p. 58 –68, jan. 2011. ISSN 1045-9219.

WILDE, T.; KOHL, J. A. Debugging High-Performance Component-Based Applications. **Concurrency and Computation: Practice and Experience**, John Wiley and Sons Ltd., Chichester, UK, v. 19, n. 5, p. 667–684, abr. 2007. ISSN 1532-0626.

WOLSKI, R. **The Eucalyptus Story**. 2012. Disponível em: http://www.eucalyptus.com/about/story. Acesso em 15 setembro 2012.

XU, L.; YANG, J.-B. **Introduction to Multi-Criteria Decision Making and the Evidential Reasoning Approach**. Manchester, UK: Manchester School of Management, 2001. v. 0106. 1-21 p. ISBN 186115111.

YOUNGE, A.; LASZEWSKI, G. von; WANG, L.; LOPEZ-ALARCON, S.; CARITHERS, W. Efficient Resource Management for Cloud Computing Environments. In: **Green Computing Conference, 2010 International**. Chicago, IL, USA: IEEE, 2010. p. 357–364.

ZASPEL, P.; GRIEBEL, M. Massively Parallel Fluid Simulations on Amazon's HPC Cloud. In: **Network Cloud Computing and Applications (NCCA), 2011 First International Symposium on**. Toulouse, France: IEEE, 2011. p. 73–78.

ZHANG, K.; DAMEVSKI, K.; VENKATACHALAPATHY, V.; PARKER, S. SCIRun2: A CCA Framework for High Performance Computing. In: **Proceedings of the 9th International Workshop on Highlevel Parallel Programming Models and Supportive Environments (HIPS'2004)**. Santa Fe, NM, USA: IEEE Computer Society, 2004. p. 72–79. ISBN 0-7695-2151-7.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud Computing: State-Of-The-Art and Research Challenges. **Journal of Internet Services and Applications**, Springer London, v. 1, p. 7–18, 2010. ISSN 1867-4828. 10.1007/s13174-010-0007-6.