UNIVERSIDADE FEDERAL DO CEARÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MANOEL MARIANO SIQUEIRA JÚNIOR

GEDANIC: UM *FRAMEWORK* PARA O GERENCIAMENTO DE BANCO DE DADOS EM NUVEM BASEADO NAS INTERAÇÕES ENTRE CONSULTAS

FORTALEZA
2012

MANOEL MARIANO SIQUEIRA JÚNIOR

GEDANIC: UM *FRAMEWORK* PARA O GERENCIAMENTO DE BANCO DE DADOS EM NUVEM BASEADO NAS INTERAÇÕES ENTRE CONSULTAS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. José Maria da Silva Monteiro Filho

Coorientador: Prof. Dr. Javam de Castro Machado

FORTALEZA
2012

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Ciências e Tecnologia

S63g Siqueira Junior, Manoel Mariano.
Gedanic: um *framework* para gerenciamento de banco de dados em nuvem baseado nas interações entre consultas. / Manoel Mariano Siqueira Junior. – 2012.
95 f. : il. , enc. ; 30 cm.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2012.
Área de Concentração: Ciência da Computação.
Orientação: Prof. Dr. José Maria da Silva Monteiro Filho.
Coorientação: Prof. Dr. Javam de Castro Machado.

1. Banco de dados - gerência. 2. Computação em nuvem. 3. *Framework* (Programa de computação). I. Título.

CDD 005

MANOEL MARIANO SIQUEIRA JÚNIOR

GEDANIC: UM *FRAMEWORK* PARA O GERENCIAMENTO DE BANCO DE DADOS EM NUVEM BASEADO NAS INTERAÇÕES ENTRE CONSULTAS

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. Javam de Castro Machado
Universidade Federal do Ceará - UFC
Coorientador

Prof. Dr. José Antônio Fernandes de Macêdo
Universidade Federal do Ceará - UFC

Prof. Dr. Angelo Roncalli Alencar Brayner
Universidade de Fortaleza - UNIFOR

Aos meus pais e à minha noiva.

AGRADECIMENTOS

Agradeço a Deus por toda a força concedida durante meu percurso no mestrado. Tive muitos momentos de dificuldade e superação, que foram vencidos graças ao acalento, iluminação e bênçãos divinas concedidas no decorrer de todo esse trabalho árduo. Obrigado, Senhor!

Eu possuo uma imensa gratidão em se tratando da minha família. Foram muito compreensivos e pacientes com minha ausência para com eles e perceberam sabiamente que o mestrado exige uma dedicação necessária para meu desenvolvimento profissional e evolução intelectual. Em especial, menciono meu pai Manoel Mariano Siqueira e minha mãe Maria Antoneide Veras Siqueira, os quais me concederam palavras que me deram forças para persistir na conclusão desse grande desafio que foi o mestrado. Minhas irmãs Maria Albermária Veras Siqueira e Maria Zeneide Mota Veras Neta torceram pelo êxito desta dissertação e, assim, deixo também minha sincera gratidão. Não poderia deixar de mencionar também minha amada noiva, Antônia Cristiane Pinheiro de Melo, a qual foi muito paciente, acreditou no meu potencial e me proferiu palavras de conforto em momentos de desânimo e dificuldade.

Agradeço aos professores Javam de Castro Machado, José Antônio Fernandes de Macêdo e Angelo Roncalli Alencar Brayner, que compuseram a banca examinadora desta dissertação. Cada qual descreveu uma série de contribuições de forma a melhorar a qualidade deste trabalho.

Meu orientador, Prof. Dr. José Maria da Silva Monteiro Filho, merece destaque em meus agradecimentos, pois foi um orientador exemplar, que procurou me ajudar quando eu estava em momentos de dificuldade e fez o que estava em seu alcance para viabilizar a concretização desta dissertação. Sua ajuda foi indispensável para que o trabalho chegasse ao estado atual. Deixo, assim, meus profundos agradecimentos a este grande homem que além de competente profissional é uma pessoa à qual tenho grande admiração e respeito.

Durante o período em que cursei o mestrado, tive o privilégio de fazer grandes amizades, que muito me ajudaram no meu desenvolvimento profissional e como ser humano. Assim, gostaria de deixar parte dos meus agradecimentos ao Regis Pires, o qual possui uma personalidade singular e conhecimento fora do comum. Este amigo compartilhou comigo muito conhecimento técnico, que contribuiu bastante para o desenvolvimento desta dissertação. Agradeço também à Mônica da Silva, que me ajudou muito em algumas disciplinas do mestrado e, por estar estudando a mesma área de conhecimento que eu (computação em nuvem), contribuiu diretamente para minha evolução nesse tema. Meu amigo Diego Sá merece também meus agradecimentos, pois, apesar de não ter contribuído diretamente com esta dissertação, me ajudou nas disciplinas e proporcionou momentos de descontração dignos de um verdadeiro amigo.

Outro amigo merecedor da minha gratidão é o Flávio Sousa, que me ajudou bastante em algumas definições utilizadas no trabalho e, principalmente, foi deveras útil na minha atual compreensão sobre conceitos referentes à computação em nuvem. Outra atitude louvável deste grande ser humano foi a de participar da revisão do texto desta dissertação com o

único interesse de ajudar o próximo. Também gostaria de mencionar Diego Victor de Sousa, o qual compartilhou comigo o difícil desafio de trabalhar e pesquisar intercaladamente durante aproximadamente seis meses. Posso dizer que foi um dos períodos mais complicados do meu percurso no mestrado. Este grande amigo me apoiou bastante e dividiu comigo a desgastante e sacrificante jornada de trabalhar/estudar por três turnos durante os dias úteis, contemplando, ainda, boa parte dos fins de semana.

Vale mencionar também alguns dos amigos, que direta ou indiretamente, contribuíram com a realização deste trabalho: Paulo Rego, David Araújo, Henrique Viana, Lucas Lemos, Pedro Thiago Holanda, Ticiana Linhares, Francicleber Ferreira, Arlino Magalhães, Lívia Almada, Igo Brilhante, Carlos Filho, Leonardo Moreira, Gustavo Santos, Fabrício Lemos, Ângela Pinheiro, Luís Eufrásio Teixeira, Jeovane Reges Cordeiro, Wagner Al-Alam, Wellington Franco, Macedo Maia, Hélio Rodrigues, Fabiano Tavares e Vinícius Pires.

Deixo um espaço de agradecimento à Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico (FUNCAP) pelo suporte financeiro concedido durante boa parte do mestrado. Adicionalmente, o grupo de pesquisa ARiDa, da UFC, proporcionou grande parte da infraestrutura necessária para o desenvolvimento desta dissertação, bem como o compartilhamento de ideias com seus integrantes e, deste modo, merece meus agradecimentos. Ademais, a Universidade da Integração Internacional da Lusofonia Afro-Brasileira (UNILAB) foi de suma importância nos últimos seis meses anteriores à minha defesa de mestrado, oferecendo parte da infraestrutura necessária para a conclusão desta dissertação. Deste modo, deixo meu profundo agradecimento ao espaço a mim concedido na UNILAB para a conclusão desta dissertação.

Pertencentes à UNILAB, não poderia deixar de mencionar a PROAD (Pró-reitoria de Administração), a PROPLAN (Pró-reitoria de Planejamento) e a CTI (Coordenação de Tecnologia da Informação), que é vinculada à PROPLAN. Em especial, agradeço à servidora pública Adênia Guimarães, pró-reitora da PROAD (em 2012), pela liberação concedida a mim para utilização da infraestrutura da UNILAB no período noturno, com a finalidade de dar continuidade à minha pesquisa do mestrado. Não poderia deixar de expressar minha gratidão ao professor Fernando Afonso Ferreira, pró-reitor da PROPLAN (em 2012), ao Ladislav Trupl, coordenador da CTI (em 2012), e ao Carlos Eduardo Barbosa, gerente da Divisão de Suporte (em 2012), os quais foram muito solícitos e compreensivos sobre a dificuldade à qual eu me encontrava no momento pré-defesa e foram de uma cordialidade e generosidade louváveis, permitindo que eu tivesse a flexibilidade necessária no trabalho para defender esta dissertação a tempo. Sem o apoio dessas quatro pessoas não seria possível que minha defesa fosse realizada dentro do tempo acordado. Adicionalmente, deixo minha gratidão também aos demais integrantes da CTI: Diego Victor de Sousa, Débora Frota, Thiago Gomes e Marcos Nagaki. Todos eles contribuíram de alguma forma para que minha defesa ocorresse.

Por fim, agradeço também a todos que não mencionei, mas que contribuíram, direta ou indiretamente, para a concretização deste trabalho. Com certeza, sozinho eu não teria conseguido essa conquista tão importante em minha vida.

“Acredite no melhor, pense seu melhor, estude seu melhor, nunca fique satisfeito com menos que seu melhor, tente seu melhor, e no longo prazo as coisas se revelarão para o melhor"

(Henry Ford)

RESUMO

Computação em nuvem é uma tendência recente de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda e com pagamento baseado no uso. Um dos principais serviços disponibilizados por uma plataforma de computação em nuvem consiste no serviço de gerenciamento de dados, ou simplesmente, serviço de dados. Este serviço assume a responsabilidade pela instalação, configuração e manutenção dos sistemas de banco de dados, bem como pelo acesso eficiente aos dados armazenados. Este trabalho apresenta um *framework*, denominado GeDaNIC, para o gerenciamento de sistemas de banco de dados em nuvem. O *framework* proposto tem por objetivo fornecer a infraestrutura de *software* necessária para a disponibilização de serviços de dados em ambientes de computação em nuvem. Neste sentido, o mecanismo concebido busca solucionar alguns problemas ainda em aberto no contexto de sistemas de banco de dados em nuvem, tais como: despacho, escalonamento de consultas e provisionamento de recursos. A abordagem concebida estende os trabalhos anteriores adicionando importantes características, como: o suporte às cargas de trabalho imprevistas e a utilização de informações sobre as interações entre consultas. O suporte às cargas de trabalhos sazonais está relacionado a uma das principais propriedades da computação em nuvem: a elasticidade rápida. Já as interações entre consultas podem proporcionar impactos significativos no desempenho dos sistemas de banco de dados. Por este motivo, o GeDaNIC utiliza informações sobre essas interações com a finalidade de reduzir o tempo de execução das cargas de trabalho submetidas ao serviço de dados e, consequentemente, aumentar o lucro do provedor deste serviço. Para isso, três novas abordagens para modelar e mensurar as interações entre instâncias e tipos de consultas são propostas. Com o objetivo de demonstrar a eficiência do *framework* proposto uma avaliação experimental usando o *benchmark* TPC-H sobre o PostgreSQL foi realizada. Os resultados apontam que a solução concebida tem potencial para aumentar o lucro do provedor do serviço de dados em nuvem.

Palavras-chave: Computação em Nuvem. Banco de Dados em Nuvem. Interações entre Consultas.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AWS	*Amazon Web Services*
DB2	*Database 2*
DBA	*Database Administrator*
DBT-3	*Database Test 3*
DRR	*Data Retrieving Rate*
EC2	*Elastic Compute Cloud*
FIFO	*First in, First Out*
GBA	*Greedy with Bidimentional Array*
I/O	Entrada/Saída
IaaS	*Infrastructure as a Service*
IS	*Intercalation Strategy*
NRO	*Normalized Run Time Overhead*
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
QoS	Qualidade de Serviço
S3	*Simple Storage Service*
SBD	Sistema de Banco de Dados
SBDD	Sistema de Banco de Dados Distribuído
SGBD	Sistema de Gerenciamento de Banco de Dados
SLA	Acordo de Nível de Serviço
SLO	Objetivo de Nível de Serviço
SQL	*Structured Query Language*
TPC-H	*Transaction Processing Performance Council Benchmark*^{TM} *H*
VM	Máquina Virtual
VMC	Máquina Virtual Controladora
VME	Máquina Virtual Escalonadora

AWS DB2 DBA DBT-3 DRR EC2 FIFO GBA I/O IaaS IS NRO OLAP OLTP QoS S3 SBD SBDD SGBD SLA SLO SQL TPC-H VM VMC VME

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Na sociedade moderna, serviços básicos e essenciais são quase todos entregues de forma completamente transparente. Serviços de utilidade pública como água, gás, eletricidade e telefone tornaram-se fundamentais para nossa vida diária e são explorados através de um modelo de pagamento baseado no uso (OPENCLOUD, 2010). As infraestruturas existentes permitem entregar esses serviços em qualquer lugar e a qualquer hora, de forma que possamos simplesmente acender a luz, abrir a torneira ou usar o fogão. O uso desses serviços é, então, cobrado de acordo com diferentes políticas voltadas para o usuário final. Recentemente, a mesma ideia de utilidade tem sido aplicada no contexto da informática e uma mudança consistente neste sentido tem sido feita com a disseminação de *Cloud Computing* ou Computação em Nuvem (SOUSA et al., 2010).

A Computação em nuvem é uma tendência recente de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda, com pagamento baseado no uso. A nuvem computacional é um modelo de computação em que dados, arquivos e aplicações residem em servidores físicos ou virtuais, acessíveis por meio de uma rede em qualquer dispositivo compatível (fixo ou móvel), e que podem ser acessados a qualquer hora, de qualquer lugar, sem a necessidade de instalação ou configuração de programas específicos. Assim, a infraestrutura da computação em nuvem pode ser vista como um *pool* de recursos computacionais (virtualmente) infinito (e elástico) oferecido no modo *self-service*, por um terceiro, via um modelo "pague o quanto usa" (SOUSA et al., 2010).

O modelo de computação em nuvem foi desenvolvido com o objetivo de fornecer serviços de fácil acesso e de baixo custo, além de assegurar características como disponibilidade e escalabilidade. Este modelo visa fornecer, basicamente, três benefícios (SOUSA et al., 2010):

- **redução no custo de aquisição e composição da infraestrutura de TI:** a infraestrutura necessária para atender as necessidades das empresas pode ser composta sob demanda e com recursos heterogêneos e de menor custo.
- **flexibilidade:** o modelo de computação em nuvem permite a adição, remoção e alteração da configuração de recursos computacionais de maneira fácil e rápida. Assim, as aplicações disponibilizadas nos ambientes de computação em nuvem ganham escalabilidade (tanto em nível de recursos de *hardware* quanto em *software*), o que permite atender melhor as necessidades das empresas e dos usuários.
- **facilidade de acesso:** na computação em nuvem os recursos de TI são fornecidos como um serviço. Os usuários podem acessar esses serviços sem a necessidade de possuir conhecimentos sobre a tecnologia utilizada. Adicionalmente, os usuários e empresas passam a acessar os serviços sob demanda e independente de localização. Assim, os usuários podem utilizar os serviços disponibilizados em nuvem a partir de qualquer lugar, a qualquer momento e utilizando uma grande variedade de dispositivos (computadores,

notebooks, *tablets* e *smartphones*, por exemplo) e interfaces (*browsers*, APIs, aplicativos, etc.). Além disso, os usuários não precisam ter conhecimento acerca da localização física dos recursos de *hardware* envolvidos na execução de um determinado serviço.

Neste contexto, as aplicações voltadas ao gerenciamento de dados são candidatas potenciais para a implantação em nuvem. Isso ocorre porque, em geral: i) as instalações dos sistemas de banco de dados (SBDs) são complexas, ii) as bases de dados utilizadas envolvem um grande volume de dados e iii) as cargas de trabalho são sazonais, o que ocasiona elevados custos tanto em *hardware* quanto em *software*. Assim, para muitas empresas, especialmente para *start-ups* e médias empresas, o pagamento baseado no uso, juntamente com o suporte fornecido para a escalabilidade e a facilidade de acesso, o modelo de computação em nuvem torna-se bastante atraente (ABADI, 2009). Por estes motivos, os usuários estão movendo seus dados e aplicações para a nuvem com a finalidade de acessá-los de forma simples, barata, flexível e independente de localização.

Neste cenário, o serviço de gerenciamento de dados (ou simplesmente, serviço de dados) assume a responsabilidade pela instalação, configuração e manutenção do sistema de banco de dados, bem como pelo acesso eficiente aos dados armazenados. O provedor do serviço de dados é o responsável por fornecer o serviço (incluindo toda infraestrutura de *hardware* e *software*), contabilizar a sua utilização, assegurar a sua qualidade e cobrar pelo seu uso.

A utilização de um serviço de gerenciamento de dados em nuvem traz diversas vantagens para os usuários (CURINO et al., 2010):

- previsibilidade e custos mais baixos, proporcional à qualidade do serviço prestado e cargas de trabalho reais;
- complexidade técnica reduzida, graças às interfaces de acesso unificado e
- elasticidade e escalabilidade, proporcionando a percepção de recursos quase infinitos.

Por outro lado, o provedor do serviço de dados necessita:

- garantir a ilusão de recursos infinitos, sob cargas de trabalho dinâmicas;
- minimizar os custos operacionais associados a cada usuário e
- assegurar o cumprimento dos acordos de nível de serviço (*Sevice Level Agreements* - SLAs).

Em um serviço de gerenciamento de dados em nuvem, os sistemas de gerenciamento de banco de dados (SGBDs) são executados em máquinas virtuais e os dados estão distribuídos, particionados e/ou replicados, de forma a melhorar o desempenho, a escalabilidade e a disponibilidade do serviço. Para o usuário (ou cliente) que está hospedando seus bancos de dados em um serviço de gerenciamento de dados em nuvem, em alguns casos, pode ser descartada, dentre os seus funcionários, a figura do DBA (*Database Administrator*), profissional

responsável por manter o SGBD funcionando continuamente e com um desempenho sempre aceitável. Isto porque o SLA já especifica a latência e a vazão esperada, além das penalidades (multas) a serem aplicadas ao provedor do serviço de dados caso os níveis de qualidade acordados não sejam atendidos. Assim, o serviço de gerenciamento de dados em nuvem pode ser concebido de forma a atender, automaticamente, a qualidade do serviço de dados, sem a interferência do DBA. Além disso, o DBA pode ser contratado pela empresa fornecedora do serviço de dados em nuvem, em vez de fazer parte do quadro de funcionários do usuário.

Do ponto de vista do provedor do serviço de dados, há a necessidade de assegurar que o serviço de dados apresente um desempenho dentro dos limites acordados nos SLAs. Para isso, o provedor necessita monitorar o desempenho do serviço de dados e, caso o seu desempenho esteja inferior ao valor contratado (em termos de tempo de resposta ou vazão, por exemplo), ajustar a configuração dos SGBDs que compõem o serviço e/ou das máquinas virtuais que hospedam os SGBDs com a finalidade de melhorar o seu desempenho, assegurando o atendimento dos SLAs.

Adicionalmente, o serviço de dados deve manter seu desempenho dentro dos valores contratados (via SLA) mesmo na ocorrência de um aumento inesperado da demanda, uma vez que uma das características do ambiente de nuvem é a elasticidade rápida. Contudo, esta não é uma tarefa fácil já que o aumento da demanda de recursos pelas aplicações que utilizam o serviço de dados são, em geral, imprevisíveis. Além disso, a quantidade de SGBDs que compõem o serviço de gerenciamento de dados, o volume dos dados armazenados (tabelas, *tuplas*, etc.) e a quantidade de ajustes possíveis (índices, tamanho dos *buffers* de memória, etc.) são muito grandes, o que limita substancialmente a intervenção humana. Logo, espera-se que os ajustes necessários para que o serviço de dados mantenha o seu desempenho dentro dos limites acordados mesmo durante os picos de utilização sejam realizados automaticamente.

Neste trabalho, consideramos que um serviço de gerenciamento de dados em nuvem consiste de um sistema de banco de dados distribuído (SBDD) executado em máquinas virtuais (*Virtual Machines* - VMs) pré-configuradas e disponibilizadas por um provedor de infraestrutura como um serviço (*Infrastructure as a Service* - IaaS). Assim, cada máquina virtual hospeda uma cópia do SGBD e uma réplica total do banco de dados. Mas, o conjunto de máquinas virtuais atua como um único sistema de banco de dados lógico. Deste modo, com o objetivo de ilustrar um serviço de dados, definimos a arquitetura representada pela Figura 1.1.

Desta forma, podemos perceber que a construção e disponibilização de um serviço de gerenciamento de dados em nuvem envolve inúmeros desafios, muitos dos quais ainda se configuram como problemas em aberto. Recentemente, diversas pesquisas têm sido realizadas com o objetivo de fornecer suporte ao gerenciamento de dados em nuvem. Contudo, a maioria dessas iniciativas busca solucionar problemas específicos, tais como: despacho (PATON et al., 2009), escalonamento (COSTA; FURTADO, 2008) e planejamento de capacidade (SHIVAM et al., 2007).

Este trabalho apresenta um *framework*, denominado GeDaNIC (SIQUEIRA; MONTEIRO; MACHADO, 2011), para o gerenciamento de sistemas de banco de dados em nuvem. O *framework* proposto tem por objetivo fornecer a infraestrutura de *software* necessária para

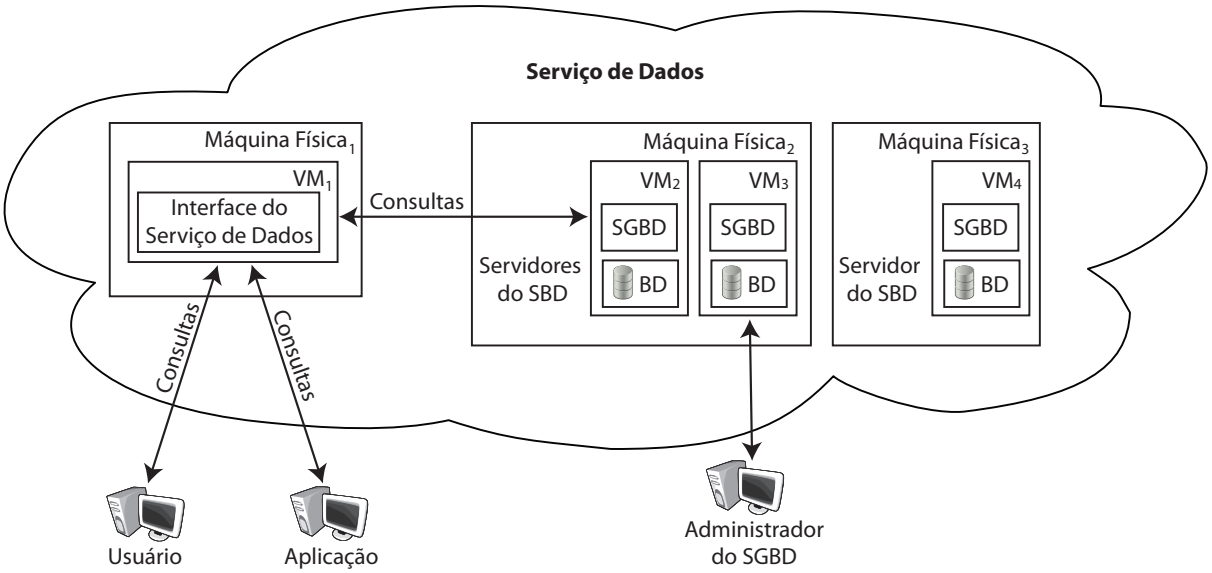


Figura 1.1: Arquitetura de um serviço de gerenciamento de dados em nuvem

a disponibilização de serviços de dados em ambientes de computação em nuvem. Neste sentido, o mecanismo concebido busca solucionar alguns problemas ainda em aberto no contexto de sistemas de banco de dados em nuvem. São eles: despacho, escalonamento de consultas e provisionamento de recursos. Estes problemas são brevemente descritos a seguir:

1. **despacho:** o objetivo do despacho é alocar cada consulta recebida pelo serviço de dados a uma das instâncias (máquinas virtuais que hospedam uma cópia do SGBD e uma réplica do banco de dados) que compõem o sistema de banco de dados em nuvem, buscando levar em consideração a sua disponibilidade e a sua capacidade para executar a consulta;

2. **escalonamento:** cada instância que compõe o sistema de banco de dados em nuvem possui um conjunto de consultas a serem executadas. Logo, torna-se necessário definir em que ordem essas consultas devem ser executadas;

3. **provisionamento de recursos:** dada uma carga de trabalho W submetida ao serviço de dados em nuvem é necessário verificar se, com a configuração atual do serviço (quantidade de VMs e de recursos físicos alocados a cada uma das VMs disponíveis, tais como memória, espaço em disco e CPUs), é possível executar a carga W, respeitando-se os parâmetros definidos no acordo de nível de serviço (SLA) ou se será necessário a adição/diminuição de recursos.

A abordagem concebida estende os trabalhos anteriores adicionando importantes características, como: o suporte às cargas de trabalho imprevistas e a utilização de informações sobre as interações entre consultas. O suporte às cargas de trabalhos sazonais está relacionado a uma das principais propriedades da computação em nuvem: a elasticidade rápida. Já as interações entre consultas podem proporcionar impactos significativos no desempenho dos sistemas de banco de dados. Por este motivo, o GeDaNIC utiliza informações sobre essas interações com a finalidade de reduzir o tempo de execução das cargas de trabalho submetidas ao serviço

de dados e, consequentemente, aumentar o lucro do provedor deste serviço. Para isso, três novas abordagens para modelar e mensurar as interações entre instâncias e tipos de consultas são propostas (SIQUEIRA et al., 2012).

Com o objetivo de demonstrar a eficiência do *framework* proposto uma avaliação experimental usando o *benchmark* TPC-H sobre o PostgreSQL foi realizada. Os resultados apontam que a solução concebida tem potencial para aumentar o lucro do provedor do serviço de dados em nuvem.

1.2 Objetivos

Este trabalho tem como objetivo principal a especificação e a implementação de um *framework* voltado para o gerenciamento de banco de dados em nuvem. O *framework* proposto tem por finalidade fornecer a infraestrutura de *software* necessária para a disponibilização de serviços de dados em ambientes de computação em nuvem. Adicionalmente, o *framework* deve apresentar ainda as seguintes características:

1. Fornecer suporte às cargas de trabalho imprevistas;
2. Explorar informações sobre as interações entre consultas.

Os objetivos específicos a serem alcançados com este trabalho incluem:

- Conceber novas abordagens para modelar e mensurar as interações entre instâncias e tipos de consultas;
- Avaliar a eficiência e a eficácia das abordagens concebidas para modelar e mensurar as interações entre consultas;
- Elaborar uma solução integrada para os problemas de despacho, escalonamento de consultas e provisionamento de recursos;
- Implementar o *framework* proposto com a finalidade de comprovar sua eficiência por meio de testes de desempenho;
- Conduzir uma avaliação comparativa do *framework* proposto em relação aos trabalhos anteriores.

1.3 Contribuições

Para atingir os objetivos descritos anteriormente, foi realizado, inicialmente, um estudo detalhado sobre o estado da arte referente ao gerenciamento de dados em nuvem. Em seguida, foram concebidas três novas abordagens para modelar e mensurar as interações entre consultas, as quais foram denominadas: *Intercalation Strategy* (IS), *Data Retrieving Rate*

(DRR) e *Greedy with Bidimentional Array* (GBA). Para avaliar a eficiência e a eficácia das abordagens propostas, um escalonador de consultas foi implementado. Esse escalonador utiliza as interações entre consultas em sua tomada de decisão, ou seja, para definir a ordem de execução das consultas. Mais precisamente, o escalonador pode ser configurado para utilizar qualquer uma das soluções propostas: IS, DRR e GBA. Testes de desempenho foram realizados para avaliar o comportamento do escalonador ao usar cada uma das três estratégias concebidas: IS, DRR e GBA.

Posteriormente, um *framework*, denominado GeDaNIC, voltado para o gerenciamento de banco de dados em nuvem foi concebido e implementado. O *framework* proposto tem por finalidade fornecer uma solução integrada para os problemas de despacho, escalonamento de consultas e provisionamento de recursos. Testes de desempenho foram realizados, utilizando-se o *benchmark* TPC-H sobre o PostgreSQL, com o objetivo de comprovar a eficiência do GeDaNIC. Por fim, uma avaliação comparativa do GeDaNIC em relação aos trabalhos anteriores foi realizada.

1.4 Estrutura da dissertação

Além desta introdução, a dissertação está organizada em mais seis capítulos, descritos a seguir:

- no Capítulo 2, são apresentados e discutidos os conceitos fundamentais para o entendimento deste trabalho, tais como: computação em nuvem, banco de dados em nuvem, SLA e interações entre consultas.
- o Capítulo 3 caracteriza e define os três problemas a serem tratados pela solução proposta neste trabalho.
- no Capitulo 4 são apresentadas e discutidas três novas abordagens para modelar e medir as interações entre consultas: *Intercalation Strategy* (IS), *Data Retrieving Rate* (DRR) e *Greedy with Bidimentional Array* (GBA). Adicionalmente, um escalonador orientado à interação entre consultas é proposto e os resultados dos testes de desempenho realizados com esse escalonador, para avaliar a eficiência das abordagens propostas, são analisados.
- o Capítulo 5 apresenta uma descrição detalhada do *framework* concebido para o gerenciamento de dados em nuvem. As soluções para os problemas de despacho, escalonamento e provisionamento de recursos são apresentadas e discutidas. Além disso, os resultados dos testes de desempenho realizados com a finalidade de avaliar o comportamento do *framework* GeDaNIC são analisados.
- no Capítulo 6, serão discutidas as principais estratégias, encontradas na literatura, para o gerenciamento de dados em nuvem.
- finalmente, o Capítulo 7 conclui este trabalho, avaliando os resultados obtidos, as dificuldades encontradas e apontando direções para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para facilitar o entendimento das abordagens propostas no decorrer deste trabalho, apresentaremos neste capítulo alguns conceitos fundamentais, tais como: computação em nuvem, banco de dados em nuvem, acordo de nível de serviço (SLA) e interação entre consultas.

2.1 Computação em Nuvem

A computação em nuvem está se tornando uma das palavras-chave da indústria de Tecnologia da Informação (TI). A nuvem é uma metáfora para a *Internet* ou infraestrutura de comunicação entre os diversos elementos que compõem um sistema computacional. Assim, a nuvem constitui uma abstração que oculta a complexidade dessa infraestrutura. Cada componente de uma nuvem computacional é disponibilizado como um serviço e, estes são normalmente alocados em *data centers*, utilizando *hardware* compartilhado para computação e armazenamento. Para utilizarem esses serviços, os usuários necessitam apenas ter em suas máquinas um sistema operacional, um navegador e acesso à *Internet*. Todos os recursos computacionais (incluindo *software* e *hardware*) estão disponíveis na *Internet*. Logo, os computadores utilizados pelos usuários não necessitam ter configurações de *hardware* avançadas, o que diminui o custo envolvido na aquisição e manutenção de equipamentos computacionais. Adicionalmente, qualquer componente de *hardware* pode ser utilizado para compor a infraestrutura da nuvem e realizar alguma tarefa que seja adequada ao seu poder de processamento ou armazenamento. Novos recursos de *hardware* podem ser adicionados à nuvem com a finalidade de aumentar a sua capacidade computacional (SOUSA et al., 2010).

A computação em nuvem é uma evolução dos serviços e produtos de tecnologia da informação sob demanda, também chamada de *Utility Computing*. O objetivo da *Utility Computing* é fornecer os componentes básicos necessários para compor uma solução de TI (tais como armazenamento, CPU e largura de banda de uma rede) como uma "mercadoria", por meio de provedores especializados e com um baixo custo por unidade utilizada. Usuários de serviços baseados em *Utility Computing* não precisam se preocupar com escalabilidade, pois a capacidade de armazenamento e processamento fornecida é teoricamente infinita (SOUSA et al., 2010).

A *Utility Computing* busca fornecer disponibilidade total. Isto é, os usuários podem ler e gravar dados a qualquer tempo, sem nunca serem bloqueados. Os tempos de resposta devem ser praticamente constantes e independentes do número de usuários simultâneos, do tamanho do banco de dados ou de qualquer parâmetro do sistema. Os usuários não precisam se preocupar com *backups*. Se os componentes falharem, é de responsabilidade do provedor substituí-los e tornar os dados disponíveis em tempo hábil por meio de réplicas (SOUSA et al., 2010).

Outra razão importante para a construção de novos serviços baseados em *Utility Computing* é que provedores de serviços que utilizam serviços de terceiros pagam apenas pelos recursos que recebem, ou seja, pagam pelo uso. Não são necessários investimentos iniciais em

TI e o custo cresce de forma linear e previsível com o uso. Dependendo do modelo do negócio, é possível que o provedor de serviços repasse o custo de armazenagem, computação e de rede para os usuários finais, já que é realizada a contabilização do uso (SOUSA et al., 2010).

Uma plataforma de computação em nuvem possui um conjunto de características essenciais. Essas características, em conjunto, definem exclusivamente a computação em nuvem e fazem a sua distinção em relação a outros paradigmas, tais como computação em grade, por exemplo. Assim, a elasticidade rápida de recursos, o amplo acesso e a medição de serviços são, consideradas por muitos autores, características básicas para compor uma solução de computação em nuvem (MELL; GRANCE, 2009). A seguir, serão discutidas as principais características de um ambiente de computação em nuvem.

- ***Self-service* sob Demanda**

  O usuário pode adquirir unilateralmente recursos computacionais, como, por exemplo, tempo de processamento no servidor ou capacidade de armazenamento, na medida em que estes recursos se façam necessários e sem precisar de interação humana com os provedores de cada serviço. Logo, o *hardware* e o *software* que compõem a nuvem podem ser automaticamente reconfigurados e orquestrados. Além disso, estas modificações são apresentadas de forma transparente para os usuários. Desta forma, os usuários, que possuem perfis diferentes, podem personalizar os seus ambientes computacionais, por exemplo, instalando *softwares* específicos ou configurando parâmetros do SGBD.

- **Amplo Acesso**

  Recursos são disponibilizados por meio da rede e acessados através de mecanismos padronizados que possibilitam o uso por plataformas *thin* ou *thin client*, tais como celulares, *laptops* e *PDAs*. A interface de acesso à nuvem não obriga os usuários a mudarem suas condições e ambientes de trabalho, como por exemplo, linguagens de programação e sistema operacional. Já os softwares clientes instalados localmente para o acesso à nuvem são leves, como um navegador de *Internet*.

- ***Pooling* de Recursos**

  Os recursos computacionais do provedor são organizados em um *pool* para servir múltiplos usuários por meio de um modelo multi-inquilino, onde diferentes recursos físicos e virtuais são dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários. Estes usuários não precisam ter conhecimento da localização física dos recursos computacionais, podendo somente especificar a localização em um nível mais alto de abstração, tais como o país, estado ou *data center*.

- **Elasticidade Rápida**

  Recursos podem ser adquiridos ou liberados de forma rápida, elástica e, em alguns casos, automática. Assim, os recursos podem ser facilmente adquiridos, quando ocorrer aumento da demanda, e liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento.

A virtualização é uma das técnicas utilizadas pelos ambientes de computação em nuvem para obter a característica de elasticidade rápida. A virtualização permite criar várias instâncias de máquinas virtuais, contendo os recursos solicitados pelos usuários (como memória, capacidade de armazenamento, capacidade de processamento, etc.), utilizando um único recurso real. Além disso, a virtualização é uma maneira de abstrair as características físicas da plataforma computacional, exibindo aos usuários *hardwares* virtuais e emulando um ou mais ambientes que podem ser independentes ou não.

- **Serviço Medido**

  As plataformas de computação em nuvem controlam e otimizam, de forma automática, o uso dos recursos computacionais. Para isso, torna-se necessário medir a utilização desses recursos. Assim, o uso dos recursos deve ser monitorado e controlado. O ajuste automático da utilização dos recursos que compõem a nuvem é realizado em um nível de abstração que seja adequado ao tipo de serviço fornecido. Por exemplo, pode-se medir e ajustar automaticamente a capacidade de armazenamento, de processamento, a largura de banda e a quantidade de contas de usuário ativas, dentre outros recursos. Adicionalmente, a medição do serviço deve ser acompanhada tanto pelo provedor quanto pelo usuário do serviço, o que proporciona transparência. Além disso, a medição do serviço é utilizada também na garantia da qualidade do serviço (*Quality of Service* - QoS). Para isso, em geral, os ambientes de computação em nuvem utilizam abordagens baseadas em SLAs. O SLA fornece informações sobre o nível mínimo de qualidade, que deve ser proporcionado pelo provedor do serviço, para determinados atributos, como, por exemplo, disponibilidade e desempenho. O SLA também contém informações sobre o faturamento (valores a serem pagos pelo usuário) e sobre as penalidades (a serem pagas pelo provedor do serviço) em caso de violação dos níveis mínimos de qualidade contratados (SOUSA et al., 2010).

2.2 Gerenciamento de Dados em Nuvem

Recentemente, diversas soluções com o objetivo de suportar o gerenciamento de dados em nuvem têm sido propostas (ABADI, 2009). Alguns serviços voltados para o gerenciamento de dados em nuvem estão começando a ser utilizados e têm o potencial de atrair clientes de diversos setores do mercado, desde pequenas empresas com o objetivo de reduzir o custo de infraestrutura, até grandes empresas que buscam soluções que possam suportar milhares de máquinas e serem capazes de absorver um aumento inesperado de demanda (ABADI, 2009). Contudo, a disponibilização desses serviços ainda envolve inúmeros desafios (SOUSA et al., 2010):

- **Sistemas de banco de dados são difíceis de escalar:** a maioria dos sistemas de banco de dados possui limites rígidos para além dos quais eles não são facilmente escaláveis. Assim, depois que um SBD atinge seus limites de escalabilidade, algumas ações se tornam inevitáveis, como por exemplo: o particionamento manual das tabelas, a migração

de dados e o balanceamento de carga. Porém, essas ações envolvem custos elevados e, frequentemente, implicam na paralisação dos sistemas em produção.

- **Sistemas de banco de dados são difíceis de configurar e manter**: os custos administrativos podem representar uma fração significativa do custo total de propriedade de um sistema de banco de dados. Além disso, é extremamente difícil para profissionais qualificados melhorarem o desempenho da maioria dos sistemas comerciais.

- **Diversidade de sistemas disponíveis dificultam a seleção:** o surgimento dos sistemas de banco de dados especializados para mercados específicos, como por exemplo, OLTP (*Online Transaction Processing*) ou OLAP (*Online Analytical Processing*), dificultam a escolha de um sistema, especialmente para os usuários cuja carga de trabalho não se enquadra perfeitamente em uma dessas categorias.

- **Provisionamento de carga inesperada ocasiona custos desnecessários:** cargas de sistemas de banco de dados são muitas vezes sazonais, levando ao fornecimento de recursos permanentes para a situação de pico. Isso geralmente resulta em excesso de recursos durante os períodos com pouca carga de trabalho e, conseqüentemente, custos desnecessários.

Da perspectiva do usuário, a principal necessidade é um serviço de gerenciamento de dados com uma interface muito simples que não necessite de ajuste ou administração. Trata-se de uma melhoria em relação a soluções tradicionais que requerem soluções de provisionamento, seleção de sistemas de banco de dados, instalação, configuração e administração. O usuário quer um bom desempenho, expresso em termos de latência e/ou vazão, que é independente de tamanhos da base de dados e de alterações da carga de trabalho. Assim, o serviço de dados deve fornecer escalabilidade e suportar cargas de trabalho inesperadas (SOUSA et al., 2010).

Alta disponibilidade é outro requisito fundamental para os serviços de gerenciamento de dados em nuvem. Adicionalmente, características avançadas de gerenciamento do banco de dados, tais como controle de concorrência, evolução de esquema e mineração de dados devem estar prontamente disponíveis e serem de fácil de utilização (SOUSA et al., 2010).

Adicionalmente, a quantidade de tarefas de administração deve ser minimizada. Para tanto, ferramentas sofisticadas que possibilitam analisar automaticamente a carga de trabalho, além de monitorar e gerenciar diversos sistemas de bancos de dados de forma centralizada, podem ser utilizadas. Para os provedores de serviços de dados que utilizam nuvens públicas, existem requisitos adicionais, tais como a definição do esquema de faturamento, segurança/privacidade dos dados e latência. Entretanto, estas questões não são específicas de bancos de dados e podem ser abordadas com técnicas em desenvolvimento pela comunidade de computação em nuvem (SOUSA et al., 2010).

A replicação de dados é, frequentemente, utilizada pelos serviços de gerenciamento de dados em nuvem com a finalidade de: i) aumentar a disponibilidade dos dados e ii) proporcionar ganhos de desempenho, por meio da redução dos tempos de resposta das consultas, e

auxiliando a garantia da qualidade do serviço (QoS). Contudo, a replicação dos dados deve ser transparente para os usuários. A replicação de dados é realizada por meio da criação de cópias dos dados em um mesmo centro de dados ou em centros de dados diferentes. Quanto à distribuição de dados, os serviços de gerenciamento de dados em nuvem podem apresentar controle global central ou distribuído (SOUSA et al., 2010).

Por outro lado, o provedor do serviço de dados tem por objetivo atender os acordos de nível de serviço, independentemente do volume de dados, das alterações na carga de trabalho ou da quantidade de transações simultâneas (SOUSA et al., 2010). Contudo, esta é uma tarefa bastante complexa, envolvendo o monitoramento contínuo do ambiente e o ajuste automático dos recursos de *software* e *hardware*.

2.3 Acordo de Nível de Serviço

A computação em nuvem é um paradigma que busca oferecer serviços sob demanda seguindo um modelo de pagamento baseado no uso. Nas plataformas de computação em nuvem, os usuários dos serviços possuem algumas garantias, que são definidas entre o provedor do serviço e o usuário. Essas garantias são descritas por meio de acordos de nível de serviço (SLAs - *Service Level Agreements*) (SOUSA et al., 2012).

Atualmente, a maioria dos provedores de serviços em nuvem fornece garantias apenas quanto à disponibilidade dos serviços. Logo, os SLAs comumente utilizados baseiam-se apenas na disponibilidade do serviço. Contudo, aspectos relacionados ao desempenho também são importantes para que a qualidade de serviço (QoS - *Quality of Service*) seja assegurada de forma mais abrangente. Portanto, é de fundamental importância que os provedores de serviço ofereçam aos seus clientes SLAs baseados em desempenho (SCHAD; DITTRICH; QUIANÉ-RUIZ, 2010)

Existem diversos modelos de SLA e QoS destinados à computação em nuvem. Alguns modelos são de propósito geral, não lidando com aspectos específicos do gerenciamento de dados. Como exemplos desses modelos, podemos citar: (COMELLAS; PRESA; FERNÁNDEZ, 2010), (MALKOWSKI et al., 2010), (SCHNJAKIN; ALNEMR; MEINEL, 2010), (MAZZUCCO, 2010), (FERRETTI et al., 2010) e (WU; GARG; BUYYA, 2011). Por outro lado, recentemente, alguns modelos de SLA e QoS voltados mais especificamente para serviços de gerenciamento de dados foram propostos. Como exemplos desses modelos, podemos citar: (LSCR, 2011), (YANG; SHANMUGASUNDARAM; YERNENI, 2009), (XIONG et al., 2011b), (CHI et al., 2011) e (SOUSA et al., 2012).

Nos ambientes de computação em nuvem, os SLAs possuem diferentes objetivos. Contudo, é possível identificar uma estrutura geral para eles. Assim, frequentemente, um SLA é composto por: a) informações sobre as partes (envolve informações sobre o provedor do serviço, sobre o cliente e sobre o faturamento, como, por exemplo, o cliente deve pagar um dólar para cada consulta executada), b) métricas ou parâmetros de qualidade, (por exemplo, tempo de resposta das consultas, vazão das transações, disponibilidade do serviço, etc.), c) algoritmos para calcular as métricas de qualidade utilizadas no SLA (como medir ou calcular

o tempo de resposta das consultas, por exemplo), d) objetivo de nível de serviço (toda consulta tem que possuir um tempo de resposta menor que 5 ms, por exemplo), em inglês SLO - *Service Level Objective*, e e) ações a serem tomadas em caso de violação do acordo (por exemplo, caso uma determinada consulta q_i apresente um tempo de resposta maior que 5 ms o cliente ganha um crédito igual ao valor pago ou que seria pago pela execução q_i) (SCHNJAKIN; ALNEMR; MEINEL, 2010).

Neste trabalho, utilizamos uma definição de SLA que foi adaptada de (SOUSA et al., 2012). Em (SOUSA et al., 2012), os autores propõem uma abordagem de QoS, denominada QoSDBC, voltada para serviços de gerenciamento de dados em nuvem. A seguir, apresentamos a definição de SLA proposta em (SOUSA et al., 2012), a qual foi concebida especificamente para serviços de gerenciamento de dados em nuvem.

Definição 2.3.1 (SLA para serviços de gerenciamento de dados em nuvem) *Um SLA específico para serviços de bancos de dados em nuvem deve ser composto por: a) informações das partes envolvidas, b) métricas do SLA, c) algoritmos para calcular as métricas do SLA, d) SLOs e e) penalidades.*

Informações sobre as partes envolvidas referem-se ao contrato entre o provedor e o cliente. As métricas do SLA estão relacionadas aos itens a serem monitorados (por exemplo, tempo de resposta das consultas, a vazão das transações, etc.) e o SLO contém os limites pré-definidos para a métrica (por exemplo, toda consulta deve possuir tempo de reposta menor do que 5 ms). Para cada métrica é definida uma forma de cálculo (por exemplo, o tempo de resposta de uma consulta q_i é igual ao instante de tempo em que q_i conclui sua execução subtraído do instante de tempo em que q_i iniciou sua execução) e penalidades em caso de não conformidade com SLOs (por exemplo, caso o tempo de resposta de uma determinada consulta q_i seja superior a 5 ms o cliente ganha um crédito igual ao valor pago ou que seria pago pela execução q_i) (SOUSA et al., 2012).

O modelo de SLA proposto em (SOUSA et al., 2012), denominado QoSDBC, utiliza as seguintes métricas:

- **tempo de resposta de uma consulta:** período de tempo entre o instante em que a consulta foi recebida pelo serviço de dados e o momento em que sua execução é concluída;
- **vazão:** quantidade de transações concluídas (ou consultas executadas) por unidade de tempo (ou seja, durante um período de tempo t);
- **disponibilidade:** fração entre a quantidade de consultas rejeitadas (não executadas) e a quantidade total de consultas recebidas pelo serviço de dados, ao longo de um período de tempo t;
- **consistência:** quantidade de dados inconsistentes que foram recuperados durante um intervalo de tempo t, de acordo com o tipo de consistência selecionado, que pode ser forte ou fraca.

No modelo proposto em (SOUSA et al., 2012), o SLA é orientado ao lucro. Neste sentido, busca-se motivar o provedor do serviço de dados a prestar serviços de alta qualidade e assim obter lucro. Adicionalmente, nos casos onde o provedor não é capaz de atender ao SLA, este é incentivado a melhorar a qualidade do serviço prestado até obter o lucro. Assim, o modelo tem por objetivo buscar o funcionamento adequado dos sistemas que utilizam o serviço de dados.

Para definir o lucro do provedor do serviço gerenciamento de dados, o modelo proposto em (SOUSA et al., 2012) considera três parâmetros: receita, custo e penalidades. Toda consulta q_i está associada a um determinado SLA SLA_i. Cada SLA SLA_i especifica a receita gerada pela execução de q_i, o SLO de q_i e a penalidade a ser paga pelo provedor do serviço de dados caso q_i não cumpra o SLO. Assim, a receita ou preço é o valor pago pelo cliente ao provedor do serviço de dados para que este execute uma determinada instância de consulta q_i. O custo operacional consiste nos gastos do provedor para a execução de q_i (dentro dos limites definidos no SLO), o que inclui, por exemplo, o custo com a infraestrutura. A penalidade ou multa é o valor pago pelo provedor ao cliente caso um SLA SLA_i não seja cumprido. Dessa forma, o lucro consiste na receita obtida pela execução de q_i menos a soma entre o custo e a penalidade, conforme mostra a fórmula a seguir (SOUSA et al., 2012):

$$Lucro = Receita - (Custo + Penalidades) \tag{2.1}$$

O lucro pode ser calculado tanto para uma instância de consulta q_i individualmente quanto para uma carga de trabalho W como um todo. A seguir, ilustraremos essas situações.

Inicialmente, suponha que uma determinada consulta q_i possua um SLA SLA_i, o qual define: a) a receita para q_i, b) o SLO acordado para q_i e c) a penalidade a ser aplicada em caso de violação do SLO. Assuma ainda os seguintes valores para esses parâmetros. A cada execução de q_i o cliente deve pagar ao provedor do serviço de dados o valor de U$ 1,0 (receita). A consulta q_i deve ser executada em no máximo 5 ms (SLO). Logo, neste exemplo, a métrica utilizada é o tempo de resposta da consulta. Caso a consulta q_i apresente um tempo de resposta maior que 5 ms o provedor paga ao cliente o valor de U$ 1,0 (penalidade). Logicamente, a penalidade deve ser definida utilizando as métricas definidas no SLO. Suponha que o custo do provedor para executar q_i seja de U$ 0,2 e que o tempo de resposta de q_i foi de 2 ms. Neste caso, o lucro do provedor seria:

$$Lucro = 1 - (0,2 - 0) = 0,8$$

Agora, considere que a consulta q_i tenha sido executada em 6 ms. Neste caso o lucro do provedor seria:

$$Lucro = 1 - (0,2 - 1) = -0,2$$

Vale ressaltar que outras métricas, como a disponibilidade, por exemplo, poderiam ser utilizadas para compor o SLO, e, consequentemente, a penalidade. No Google AppEngine[1],

[1] http://code.google.com/appengine/sla.html

Microsoft Azure[2] ou Amazon S3[3], se a disponibilidade ficar abaixo de 99,9% (SLO), os clientes recebem um crédito, o qual será calculado levando em consideração o tempo em que o serviço ficou indisponível e a receita (SOUSA et al., 2012).

Alguns modelos de SLA utilizam o tempo de resposta para compor as penalidades. Como exemplos desses modelos podemos citar: (CHI et al., 2011) e (SOUSA et al., 2012). Em (SOUSA et al., 2012), a penalidade é a razão entre a soma de todas as consultas violadas pelo total de consultas multiplicado pela receita do sistema, de acordo com a fórmula abaixo (SOUSA et al., 2012).

$$Penalidades = \frac{\sum consultas\_violadas}{\sum consulta} \times Receita \tag{2.2}$$

Neste caso, o lucro deve ser calculado para uma carga de trabalho completa. Para ilustrar, suponha uma carga de trabalho W contendo 100 consultas. Considere ainda que W possui um SLA SLA_W, o qual define: a) a receita para W, b) o SLO acordado para cada consulta q_i que pertence a W e c) a penalidade a ser aplicada em caso de violação do SLO. Assuma os seguintes valores para esses parâmetros. A cada execução da carga de trabalho W o cliente deve pagar ao provedor do serviço de dados o valor de U$ 100 (receita). Cada consulta q_i pertencente a W deve ser executada em no máximo 5 ms (SLO). Caso uma consulta q_i apresente um tempo de resposta maior que 5 ms esta viola o SLO. Suponha que o custo do provedor para executar W seja de U$ 20 e que 50 consultas apresentaram um tempo de resposta maior que 5 ms, violando o SLO. Neste caso, a penalidade e o lucro do provedor seriam calculados da seguinte maneira:

$$Penalidades = \tfrac{50}{100} \times 100 = 50$$

e

$$Lucro = 100 - (20 + 50) = 30$$

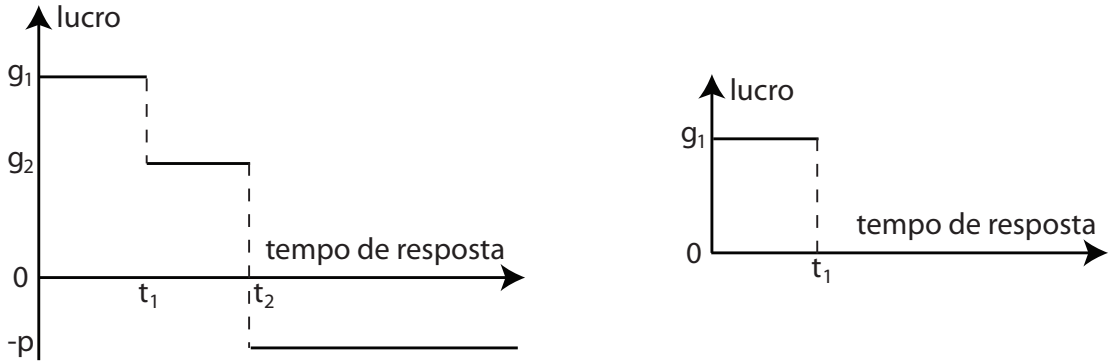
O trabalho apresentado em (CHI et al., 2011) difere da abordagem proposta por (SOUSA et al., 2012) em três aspectos fundamentais: a) (CHI et al., 2011) modela o SLA utilizando como métrica apenas o tempo de resposta das consultas, b) em (CHI et al., 2011) cada consulta q_i possui um SLO específico, assim uma consulta q_1 pode ter um SLO de 5 ms, enquanto uma consulta q_2 pode ter um SLO de 10 ms (já em (SOUSA et al., 2012) todas as consultas possuem o mesmo SLO) e c) em (CHI et al., 2011) os autores representam o SLA por meio de funções degrau.

Seja q_i uma consulta. (CHI et al., 2011) conceitua o lucro similarmente à (SOUSA et al., 2012), ou seja, utilizando os conceitos de receita, penalidade e custo operacional. Todavia, o SLA associado à consulta q_i é representado por uma função degrau que modela o lucro gerado pela execução de q_i. Assim, (CHI et al., 2011) também utiliza o conceito de lucro associado à consulta q_i, que consiste na receita acordada para execução de q_i subtraída da penalidade aplicada à consulta q_i. Ou seja, o custo operacional não é levado em consideração na definição

[2]http://www.microsoft.com/windowsazure/sla

[3]http://aws.amazon.com/s3-sla/

do lucro utilizado nas funções degrau. As figuras 2.1a e 2.1b ilustram a utilização das funções degrau para modelar o lucro associado à consulta q_i, como proposto em (CHI et al., 2011). Nestas figuras, assume-se que a consulta q_i começou sua execução no instante de tempo 0 e terminou no instante de tempo t. Nesta representação, como ilustrado na Figura 2.1a, quando $t \leq t_1$, há uma receita (ganho) de g_1 referente à q_i; caso contrário, se $t_1 \leq t \leq t_2$, há o ganho de g_2, e assim por diante. Quando nenhum dos tempos pré-definidos (t_1, t_2, etc.) é cumprido, a penalidade p é aplicada. Observe que a função degrau ilustrada na Figura 2.1b é um caso particular da função degrau exibida na Figura 2.1a.



(a) Modelo geral de lucro-penalidade (b) Modelo de lucro g/0

Figura 2.1: Representações de SLA por consulta (adaptado de (CHI et al., 2011))

A solução proposta neste trabalho modela a penalidade de maneira similar ao modelo ilustrado na Figura 2.1b, o qual também é utilizado em (SOUSA et al., 2012). Adicionalmente, utilizaremos uma definição de SLA semelhante à proposta em (SOUSA et al., 2012) (Definição 2.3.1). Porém, com as seguintes modificações:

- os valores para receita, SLO e penalidade serão calculados para cada instância de consulta;
- somente a métrica de tempo de resposta será utilizada;
- como a penalidade será calculada para cada instância de consulta, a fórmula 2.2, concebida para o cálculo das penalidades, não será utilizada.

Maiores detalhes sobre a modelagem do SLA utilizada na solução proposta nesse trabalho são encontrados na Seção 5.1.

2.4 Interação entre Consultas

Para melhorar o desempenho do gerenciamento de dados em nuvem, garantindo um melhor cumprimento dos SLAs acordados, os sistemas de banco de dados em nuvem podem explorar o potencial das consultas de influenciarem positivamente nos tempos de resposta umas das outras. Com este propósito, procuramos aliar a propriedade de interação entre consultas,

descrita a seguir, com o gerenciamento de dados em nuvem e, consequentemente, permitir que o provedor do serviço de dados incremente seu lucro.

Tipicamente uma carga de trabalho em sistemas de banco de dados consiste em um conjunto formado por diversas instâncias de consulta que executam concorrentemente e interagem entre si (AHMAD; ABOULNAGA; BABU, 2009). Cada instância de consulta pertence a um determinado tipo de consulta. Um tipo de consulta pode ser definido como um *template* (ou seja, uma consulta parametrizada) para consultas em SQL, o qual consiste de uma expressão SQL com parâmetros. Sempre que um *template* é instanciado, ou seja, um conjunto de valores é atribuído aos seus parâmetros, tem-se uma instância de consulta. Por exemplo, o *benchmark* TPC-H (TPC, 2012) define 22 *templates*. A partir de cada *template* do TPC-H, diversas instâncias de consulta podem ser geradas. Neste trabalho, cada *template* de consulta será considerado um tipo de consulta.

As figuras 2.2a e 2.2b ilustram as noções de tipo de consulta (*template*) e instância de consulta. A Figura 2.2a mostra um tipo de consulta Q_k contendo um único parâmetro, representado pelo símbolo "?". Diferentes valores para este parâmetro geram diferentes instâncias do mesmo tipo de consulta. Já a Figura 2.2b mostra uma instância de consulta q_i do tipo Q_k.

```
1 SELECT *
2 FROM lineitem AS l, orders AS o,
3   supplier AS s, nation AS n
4 WHERE l.l_orderkey = o.o_orderkey AND
5   l.l_suppkey = s.s_suppkey AND
6   s.s_nationkey = n.n_nationkey AND
7   n.n_name = ?;
```

(a) Tipo de consulta (*template*)

```
1 SELECT *
2 FROM lineitem AS l, orders AS o,
3   supplier AS s, nation AS n
4 WHERE l.l_orderkey = o.o_orderkey AND
5   l.l_suppkey = s.s_suppkey AND
6   s.s_nationkey = n.n_nationkey AND
7   n.n_name = 'USA';
```

(b) Instância de consulta

Figura 2.2: Exemplos de tipo e instância de consulta

Seja uma carga de trabalho W. Sejam $q_i \in W$ e $q_j \in W$ instâncias de consultas. As consultas q_i e q_j podem interagir entre si, influenciando no desempenho da carga W, tanto positivamente quanto negativamente. Por exemplo, a consulta q_i, quando executada, pode fazer com que um determinado conjunto de páginas de disco seja levado para a memória. Assim, caso q_j utilize esse mesmo conjunto de páginas de disco e q_j seja executada após q_i, a execução de q_j poderia se beneficiar dos dados já armazenados na memória. Neste caso, não seria necessário levar as páginas de disco para a memória, uma vez que estas já se encontram na memória. Assim, a consulta q_j poderia apresentar um tempo de resposta menor do que apresentaria caso fosse executada isoladamente ou, até mesmo, em outra ordem de execução. Logo, a ordem de execução das consultas q_i e q_j pode influenciar o desempenho da carga de trabalho W. Além disso, q_i e q_j poderiam interferir na utilização dos recursos de *hardware*, como processador ou memória (AHMAD; ABOULNAGA; BABU, 2009).

A Figura 2.3 ilustra o resultado de um experimento realizado por (AHMAD; ABOULNAGA; BABU, 2009). Nesse experimento, duas cargas de trabalho, W_1 e W_2, são compostas por um mesmo conjunto de 60 instâncias de consulta. A diferença entre W_1 e W_2 reside exclusivamente na ordem de execução das 60 instâncias de consulta. Observe que o tempo de resposta da carga de trabalho W_1 foi de 5,4 horas, enquanto o tempo de resposta da carga de

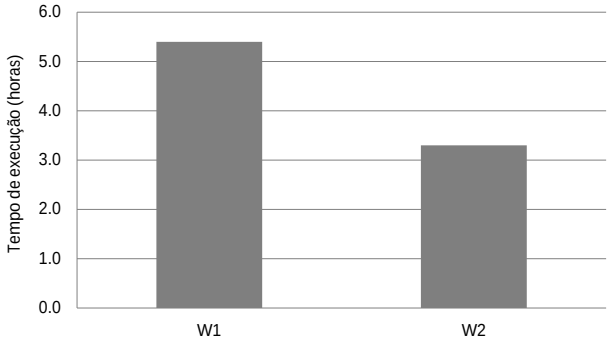


Figura 2.3: Exemplo de interação entre consultas (adaptado de (AHMAD; ABOULNAGA; BABU, 2009))

trabalho W_2 foi de 3,3 horas, resultando em uma diferença significativa de 2,1 horas. Portanto, a experimentação apresentada em (AHMAD; ABOULNAGA; BABU, 2009) ilustra um exemplo da influência que a interação entre consultas exerce sobre o desempenho de um SBD.

3 CARACTERIZAÇÃO DO PROBLEMA

Neste capítulo são descritos, em maiores detalhes, três dos principais problemas relacionados ao gerenciamento de dados em nuvem: despacho, escalonamento de consultas e provisionamento de recursos. A solução desses problemas torna-se de fundamental importância para que seja possível implementar e disponibilizar serviços de gerenciamento de dados em nuvem.

Como definido no Capítulo 1, um serviço de gerenciamento de dados em nuvem deve assumir a responsabilidade pela infraestrutura de *hardware* e *software* necessária para a disponibilização e utilização de sistemas de bancos de dados em um ambiente de computação em nuvem. Desta forma, o serviço de dados responsabiliza-se: a) pela instalação, configuração e manutenção do sistema de banco de dados, b) pelo armazenamento dos dados (que podem ser distribuídos, particionados e/ou replicados, com a finalidade de melhorar o desempenho e a disponibilidade do serviço), c) pelo acesso eficiente aos dados armazenados (assegurando o cumprimento dos SLAs e, consequentemente, a qualidade do serviço prestado), d) pela contabilização da utilização do serviço e e) pelo faturamento do serviço (ou seja, por cobrar pela utilização do serviço). Adicionalmente, um serviço de gerenciamento de dados em nuvem pode ser definido como um sistema de banco de dados distribuído (SBDD) executado em máquinas virtuais (*Virtual Machines* - VMs). Portanto, cada máquina virtual hospeda uma cópia do SGBD e uma réplica total do banco de dados. Contudo, o conjunto de máquinas virtuais atua como um único sistema de banco de dados lógico.

Além disso, para uma melhor definição do problema, algumas premissas são assumidas, tais como: a) os SBDs são relacionais, b) as aplicações que utilizam o serviço de dados apresentam características OLAP e c) o objetivo do serviço de dados é maximizar o lucro do provedor do serviço (e não minimizar os tempos de resposta das consultas).

3.1 Despacho

Em um sistema de bancos de dados em nuvem existem diversas máquinas virtuais, onde cada uma delas hospeda uma cópia do SGBD e uma réplica total do banco de dados. Logo, cada uma dessas máquinas virtuais possui a capacidade de executar as consultas requisitadas pelas aplicações clientes.

Deste modo, seja q_i uma consulta submetida ao serviço de dados em nuvem e, V o conjunto de máquinas virtuais com capacidade de executar q_i. O objetivo do despacho é alocar a consulta q_i para uma máquina virtual $v_k \in V$, a fim de que a consulta q_i seja executada em v_k.

Contudo, essa escolha deve levar em consideração a disponibilidade e a capacidade (ou seja, a quantidade de consultas já recebidas) das máquinas virtuais disponíveis no serviço de dados, além do SLO de q_i. Afinal, o provedor do serviço de dados tem por objetivo fazer com que a consulta q_i seja executada dentro dos limites definidos no seu SLO, envolvendo o menor custo operacional possível, com a finalidade de maximizar o lucro.

Segundo (CHI et al., 2011), o problema de despacho pode ser definido a partir do seguinte questionamento: quando uma nova consulta q_i chega ao serviço de dados e existe mais de uma VM, para qual VM deve-se despachar a consulta q_i a fim de maximizar o lucro do provedor? A Figura 3.1 ilustra o problema de despacho no gerenciamento de bancos de dados em nuvem.

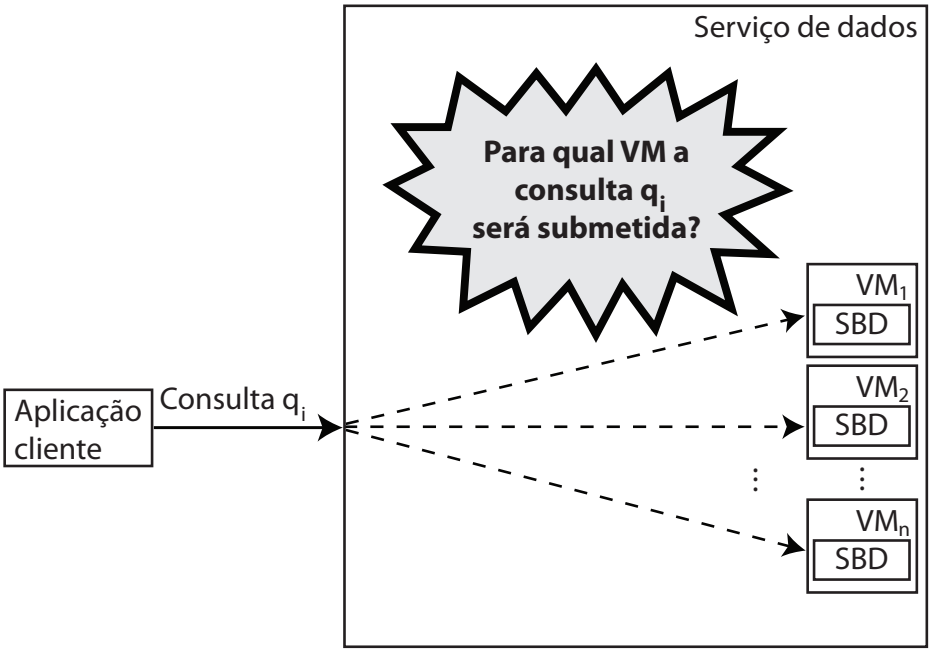


Figura 3.1: Despacho de consultas

3.2 Escalonamento

Seja q_i uma consulta submetida ao serviço de dados em nuvem, V o conjunto de máquinas virtuais com capacidade de executar q_i e, $v_k \in V$ uma VM. Cada VM v_k possui uma cópia (instância) do SGBD, uma réplica total do banco de dados e um conjunto de consultas a serem executadas. Esse conjunto de consultas constitui uma fila de escalonamento F. Logo, torna-se necessário definir em que ordem essas consultas devem ser executadas, ou seja, a posição de cada consulta na fila de escalonamento, sendo este o objetivo do problema de escalonamento de consultas. Portanto, seja F a fila de escalonamento de uma VM $v_k \in V$ e, q_i uma consulta pertencente à fila F. O problema de escalonamento de consultas consiste em definir qual consulta $q_i \in F$ será a próxima a executar.

Contudo, cada consulta q_i possui um SLA (SLA_i), o qual define uma receita, um SLO e uma penalidade. Assim, por exemplo, duas consultas q_1 e q_2 podem ter receitas, SLOs e penalidades diferentes. Logo, esses parâmetros devem ser considerados durante o escalonamento das consultas, ou seja, no processo de definição do posicionamento das consultas na fila F, com a finalidade de maximizar o lucro do provedor do serviço de dados.

Segundo (CHI et al., 2011), o problema do escalonamento de consultas em bancos de dados em nuvem pode ser descrito por meio da seguinte pergunta: quando há uma fila de

escalonamento contendo diversas consultas com diferentes perfis de lucro (receitas, SLOs e penalidades), qual consulta deveria ser escolhida para executar primeiro? Ou ainda, em que ordem as consultas presentes na fila de escalonamento devem ser executadas a fim de maximizar o lucro do provedor do serviço de dados? A Figura 3.2 ilustra o problema de escalonamento.

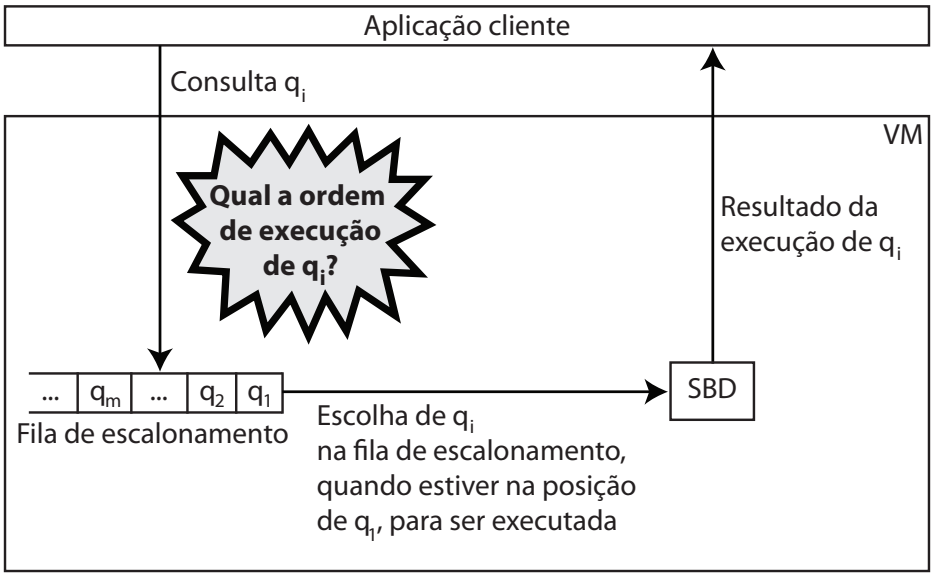


Figura 3.2: Escalonamento de consultas

3.3 Provisionamento de Recursos ou Planejamento de Capacidade

Dada uma carga de trabalho W submetida ao serviço de dados em nuvem é necessário verificar, inicialmente, se com a configuração atual do serviço (quantidade de VMs e de recursos físicos alocados a cada uma das VMs disponíveis, tais como memória, espaço em disco e CPU) é possível executar a carga W respeitando-se os parâmetros definidos no acordo de nível de serviço (SLA). Caso seja verificado que a configuração atual não é capaz de assegurar os níveis de qualidade contratados, novas VMs devem ser adicionadas (instanciadas). Por outro lado, caso observe-se que a configuração atual está assegurando os níveis de qualidade contratados, deve ser verificado sé é possível reduzir a quantidade de recursos utilizados (ou seja, VMs) sem comprometer o atendimento dos SLAs contratados. Caso seja possível, deve-se selecionar quantas e quais VMs deverão ser suspensas.

Segundo (CHI et al., 2011), o problema do provisionamento de recursos ou planejamento de capacidade pode ser descrito por meio das seguintes perguntas: dada a carga de trabalho corrente W e a situação atual do lucro proporcionado pela execução de W, torna-se necessário instanciar uma nova VM? Existe subutilização de recursos (VMs)? Ou seja, é possível atender o SLA associado a W com uma quantidade menor de VMs?

Assim, uma carga de trabalho W pode demandar uma quantidade de recursos de *hardware* (ou de maneira simplificada, VMs) maior ou menor que a quantidade de recursos (VMs) atualmente alocados para a execução de W. Logo, para atender adequadamente os SLAs

contratados e ao mesmo tempo maximizar o lucro do provedor do serviço de dados, pode ser necessária a instanciação/suspensão de VMs. A Figura 3.3 ilustra o problema do provisionamento de recursos.

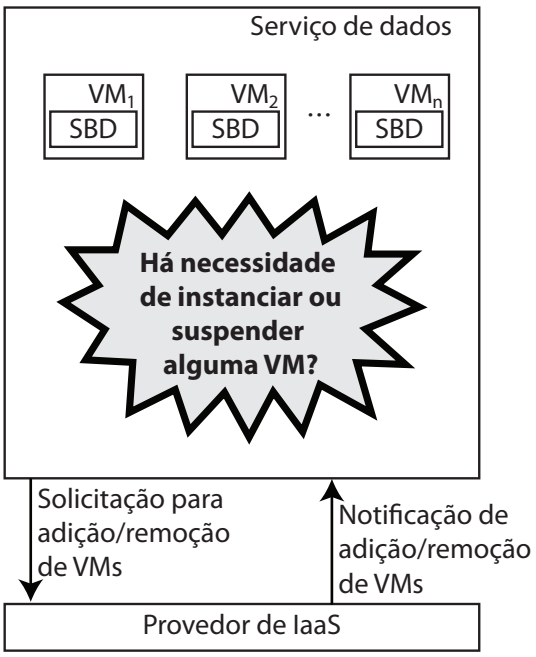


Figura 3.3: Provisionamento de recursos

4 ABORDAGENS PARA INTERAÇÃO ENTRE CONSULTAS

Atualmente, a maioria dos bancos de dados é armazenada em discos rígidos. Adicionalmente, sabe-se que a taxa de acesso a dados armazenados em disco é algumas ordens de magnitude maior que em memória principal, especialmente quando o acesso a dados ocorre de maneira aleatória (randômica).

Assim, para executar uma consulta q_i, o gerenciador de *buffer* (um dos módulos que compõem um SGBD) precisa transferir páginas de dados do disco para a memória principal (mais especificamente, para uma área da memória denominada *buffer pool*). Logicamente, essas páginas podem ser utilizadas por uma consulta q_j que execute concorrentemente com q_i ou após a execução de q_i. Este cenário ilustra o conceito de interação entre consultas. Neste exemplo, a interação ocorre entre as consultas q_i e q_j. Desta forma, executando-se q_i antes de q_j (ou vice-versa), q_j pode se beneficiar, apresentando um tempo de resposta menor, caso parte das páginas necessárias para a sua execução já se encontrem no *buffer pool*, uma vez que isso diminuiria a quantidade de operações de I/O necessárias para o processamento de q_j. Com base nesta observação, propomos três abordagens distintas para modelar e medir a interação entre instâncias e tipos de consulta. Neste capítulo, discutiremos essas abordagens em detalhes.

As abordagens propostas utilizam um conceito denominado fator de interação, o qual consiste em um número entre 0 e 1. O fator de interação quantifica a interação entre duas instâncias de consulta ou entre dois tipos de consulta. Valores próximos a 1 indicam uma forte interação enquanto valores próximos a 0 indicam uma fraca interação.

As três abordagens propostas para modelar e medir a interação entre consultas (Seção 4.2) foram denominadas: *Intercalation Strategy* (IS), *Data Retrieving Rate* (DRR) e *Greedy with Bidimentional Array* (GBA). As três abordagens não requerem nenhuma informação acerca de aspectos internos do SGBD, o que torna as abordagens não intrusivas, ou seja, portáveis entre os SGBDs. A abordagem IS baseia-se somente nos textos SQL e nos planos de execução das consultas. Nenhum pré-processamento é necessário. A abordagem DRR baseia-se na taxa de recuperação de dados do disco rígido. Para que esta taxa seja calculada necessita-se de uma etapa de pré-processamento. A abordagem GBA utiliza uma matriz bidimensional que armazena para cada par de tipos de consulta Q_{k_i} e Q_{k_j} a estimativa do ganho que seria proporcionado pela execução de um par qualquer de instâncias de consulta q_i e q_j, onde k_i e k_j são os identificadores dos tipos Q_{k_i} e Q_{k_j}, respectivamente, $q_i \in Q_{k_i}$ e $q_j \in Q_{k_j}$. Para a definição dos valores de cada célula da matriz bidimensional utilizada na abordagem GBA, ocorre uma etapa de pré-processamento com um custo computacional maior do que aquele necessário para a etapa de pré-processamento da abordagem DRR.

Para demonstrar o benefício de se explorar as interações entre consultas, um novo escalonador, voltado para cargas de trabalho contínuas (*online*), foi concebido, implementado e avaliado. Neste escalonador, a decisão sobre a ordem de execução das consultas baseia-se no fator de interação entre elas, ou seja, nas interações entre as consultas. Uma avaliação experimental do escalonador proposto foi realizada utilizando-se o *benchmark* TPC-H sobre o SGBD PostgreSQL. Os resultados demonstram que as abordagens propostas têm potencial para

aprimorar não só os algoritmos de escalonamento, mas muitos dos algoritmos de sintonia de bancos de dados.

Este capítulo é organizado como se segue: a Seção 4.1 discorre sobre o conceito de relacionamento entre tabelas, o qual é utilizado nas abordagens IS e DRR. A Seção 4.2 descreve as três abordagens propostas para modelar e medir as interações entre consultas (IS, DRR e GBA). A Seção 4.3 apresenta o escalonador concebido com a finalidade de avaliar as abordagens propostas e discute os resultados dos experimentos realizados.

4.1 Relacionamento entre Tabelas

As abordagens IS e DRR computam o fator de interação entre duas instâncias de consulta q_i e q_j utilizando a definição de relacionamento entre tabelas. Existem dois tipos de relacionamentos entre tabelas: possível interseção (Definição 4.1.1) e disjunção (Definição 4.1.2). As definições desses relacionamentos consideram:

1. P_{q_i} e P_{q_j}, planos de execução de duas instâncias de consulta q_i e q_j, respectivamente;
2. $T_{P_{q_i}}$ e $T_{P_{q_j}}$, conjuntos de tabelas referenciadas em P_{q_i} e P_{q_j}, respectivamente;
3. $OP(t_i, P_{q_i})$, o conjunto de operações de P_{q_i}, executadas sobre a tabela t_i, onde $t_i \in T_{P_{q_i}}$.

Definição 4.1.1 (Relacionamento de possível interseção) *Duas tabelas $t_i \in T_{P_{q_i}}$ e $t_j \in T_{P_{q_j}}$ apresentam um relacionamento de possível interseção, se e somente se: (i) $t_i = t_j$ (ou seja, representam a mesma tabela do banco de dados), e; (ii) $\exists o_i \in OP(t_i, P_{q_i})$ e $\exists o_j \in OP(t_j, P_{q_j})$, tal que uma das seguintes condições ocorre: a) o_i ou o_j é uma operação de table scan; b) o_i e o_j são operações de index scan definidas sobre chaves de busca (colunas) distintas; c) o_i e o_j são operações de index scan definidas sobre a mesma chave de busca (coluna) e existe interseção entre os filtros (condições de seleção) utilizados em o_i e o_j.*

Definição 4.1.2 (Relacionamento de disjunção) *Duas tabelas $t_i \in T_{P_{q_i}}$ e $t_j \in T_{P_{q_j}}$ têm um relacionamento de disjunção, se e somente se: (i) $t_i \neq t_j$, ou; (ii) nenhuma das condições do item (ii) da Definição 4.1.1 ocorre.*

Considerando $t_i \in T_{P_{q_i}}$ e $t_j \in T_{P_{q_j}}$ duas tabelas, o relacionamento de possível interseção entre t_i e t_j diz respeito à possibilidade de existirem operações sobre t_i e t_j em P_{q_i} e P_{q_j}, respectivamente, que acessem dados em comum. Por sua vez, o relacionamento de disjunção implica que operações sobre t_i e t_j em P_{q_i} e P_{q_j}, respectivamente, não acessam dados em comum.

4.2 Abordagens Propostas para Modelar e Medir a Interação entre Consultas

Esta seção apresenta em detalhes as três abordagens concebidas para modelar e medir a interação entre instâncias e tipos de consulta (IS, DRR e GBA). A interação entre

duas instâncias de consulta q_i e q_j é quantificada por meio do fator de interação (Definição 4.2.1). Assim, os algoritmos concebidos para implementar as abordagens propostas recebem como entrada duas instâncias de consulta q_i e q_j (ou os tipos de consulta dessas instâncias, denominados Q_{k_i} e Q_{k_j}, como no caso da abordagem GBA) e retornam como saída o fator de interação entre q_i e q_j.

Definição 4.2.1 (Fator de interação) *Um número entre 0 e 1 que quantifica a interação entre duas instâncias de consulta q_i e q_j (ou entre dois tipos de consultas Q_{k_i} e Q_{k_j}). Valores próximos a 1 indicam forte interação enquanto valores próximos a 0 indicam fraca interação.*

4.2.1 *Intercalation Strategy* (IS)

Sejam q_i e q_j duas instâncias de consulta executadas nessa ordem, e t_k a única tabela em comum entre q_i e q_j. A abordagem IS assume que:

1. operações de *table scan* sobre t_k acessam mais páginas de dados pertencentes à tabela t_k do que operações de *index scan*;
2. as páginas de dados de t_k são armazenadas sequencialmente em disco;
3. a partir da premissa anterior, a abordagem IS considera também que operações de *table scan* sobre t_k acessam mais páginas do disco rígido por segundo (apresentam maior vazão) do que operações de *index scan* sobre t_k.

A ideia que leva a esta terceira suposição é que operações de *table scan* fariam buscas sequenciais sobre t_k no disco rígido. Já operações de *index scan* fariam buscas de páginas aleatórias (não sequenciais) em disco, uma vez que além das páginas de dados, as páginas relativas aos índices também seriam acessadas. Deste modo, a abordagem IS considera que q_i e q_j possuem um fator de interação maior quando ocorre operação de *table scan* sobre t_k no plano de execução de q_i e ocorre somente operações de *index scan* sobre t_k no plano de execução de q_j do que na situação inversa (q_i somente com *index scan* e q_j com *table scan*).

Para computar o fator de interação entre duas instâncias de consulta q_i e q_j, a abordagem *Intercalation Strategy* (IS) baseia-se apenas nas instruções SQL e nos planos de execução de q_i e q_j. Nenhuma etapa de pré-processamento é necessária. O Algoritmo 1 descreve os passos executados pela abordagem IS para calcular o fator de interação entre duas instâncias de consulta q_i e q_j.

O Algoritmo 1 recebe como entrada duas instâncias de consulta q_i e q_j, onde se supõe que q_i começa sua execução antes de q_j. A variável global *sgbd* representa um *driver* utilizado para acessar a metabase do SGBD. A saída desse algoritmo é o fator de interação entre q_i e q_j, denotado por $f(q_i, q_j)$. Vale salientar que $f(q_i, q_j)$ pode ser diferente de $f(q_j, q_i)$.

Sejam: a) P_{q_i} e P_{q_j}, planos de execução de q_i e q_j, respectivamente; b) $T_{P_{q_i}}$ e $T_{P_{q_j}}$, conjuntos de tabelas referenciadas em P_{q_i} e P_{q_j}, respectivamente, e; c) $OP(t_i, P_{q_i})$, o conjunto de operações de P_{q_i}, executadas sobre a tabela t_i, onde $t_i \in T_{P_{q_i}}$. Assim, $o_i \in OP(t_i, P_{q_i})$, representa

Algoritmo 1: Abordagem IS

Entrada: instâncias de consulta q_i e q_j
Saída: fator de interação
Dados: sgbd

```
1  início
2      se eNulo(q_i) ou eNulo(q_j) então
3          retorna 0;
4      fim se
5      T_{q_i} ← sgbd.interpretarConsulta(q_i);
6      T_{q_j} ← sgbd.interpretarConsulta(q_j);
7      tamanhoTotal ← Σ_{t_i∈T_{q_i}} tamanho(t_i) + Σ_{t_j∈T_{q_j}} tamanho(t_j) − Σ_{t_k∈T_{q_i}∩T_{q_j}} tamanho(t_k);
8      α ← 0;
9      para cada t_i ∈ T_{q_i} faça
10         para cada t_j ∈ T_{q_j} faça
11             relacionamento ← t_i.obterRelacionamento(t_j);
12             se ePossivelIntersecao(relacionamento) então
13                 se temTableScan(OP(t_i, P_{q_i})) e temTableScan(OP(t_j, P_{q_j})) então
14                     pesoInteracao ← P_1;
15                 senão se temTableScan(OP(t_i, P_{q_i})) e não temTableScan(OP(t_j, P_{q_j})) então
16                     pesoInteracao ← P_2;
17                 senão se não temTableScan(OP(t_i, P_{q_i})) e temTableScan(OP(t_j, P_{q_j})) então
18                     pesoInteracao ← P_3;
19                 senão
20                     pesoInteracao ← P_4;
21                 fim se
22                 pesoTamanho ← tamanho(t_i) / tamanhoTotal;
23                 α ← α + pesoInteracao × pesoTamanho;
24             fim se
25         fim para cada
26     fim para cada
27     retorna α;
28 fim
```

uma operação (*table scan*, *index scan* ou *index seek*) definida sobre a tabela t_i presente no plano de execução P_{q_i}.

A variável $tamanho(t_k)$ representa o tamanho da tabela t_k (em *kilobytes*), *tamanhoTotal* armazena a soma de $tamanho(t_k)$ para cada $t_i \in T_{q_i}$ e $t_j \in T_{q_j}$, considerando tabelas repetidas somente uma única vez. A variável *relacionamento* representa o relacionamento (interação possível ou disjunção) entre duas tabelas.

Analisando o Algoritmo 1, podemos observar na linha 2 que se q_i ou q_j são nulos, o valor 0 é retornado (linha 3). Assim, são necessárias pelo menos duas consultas para que haja interação, ou seja, para que se possa medir o fator de interação. A linha 2 realiza essa verificação. Seja T_{q_i} o conjunto das tabelas presentes em P_{q_i} (plano de execução de q_i) e, T_{q_j} o conjunto das tabelas existentes em P_{q_j} (plano de execução de q_j). Nas linhas 5 e 6, as variáveis

T_{q_i} e T_{q_j} são criadas, utilizando-se para isso o método *interpretaConsulta*. Cada tabela $t_i \in T_{q_i}$ é representada por uma estrutura de dados, cujo tipo é denominado EDTabela. Essa estrutura de dados armazena as seguintes informações: tamanho da tabela (por exemplo, em KB), nome da tabela e o conjunto $OP(t_i, P_{q_i})$ de operações (*table scan* ou *index scan*) definidas sobre t_i no plano de execução P_{q_i}, incluindo os predicados (filtros) utilizados por essas operações. O mesmo vale para as tabelas em T_{q_j}.

Seja $tamanho(t_k)$ o tamanho de uma tabela qualquer t_k (em KB, por exemplo), representada pela estrutura de dados do tipo EDTabela, a variável $tamanhoTotal$ armazena a soma de $tamanho(t_k)$ para cada $t_i \in T_{q_i}$ e $t_j \in T_{q_j}$ (na linha 7), desconsiderando a ocorrência de repetições de tabelas. Estas informações são usadas para definir o peso de cada tabela no valor do fator de interação (linha 22).

Das linhas 8 até 27, a variável α é calculada. Esse processo começa com a inicialização dessa variável na linha 8. Em seguida, para cada par de tabelas t_i e t_j, tal que $t_i \in T_{q_i}$ e $t_j \in T_{q_j}$, é verificado o quanto esse par influencia na interação entre q_i e q_j (entre as linhas 9 e 27, inclusive). O primeiro passo dessa verificação consiste em identificar o relacionamento entre t_i e t_j (linha 11), que pode ser possível interseção ou disjunção. Caso o relacionamento entre t_i e t_j seja de disjunção, a variável α não será modificada. Do contrário, a variável α será incrementada baseada em dois pesos:

- peso da interação: representado pela variável *pesoInteracao* e calculado entre as linhas 13 e 21, de acordo com a Tabela 4.1. Dependendo da existência ou não de operações de *table scan* definidas em $OP(t_i, P_{q_i})$ e $OP(t_j, P_{q_j})$, a variável *pesoInteracao* poderá assumir quatro valores distintos: P_1, P_2, P_3 ou P_4. Estes valores pertencem ao intervalo $[0,1]$. Por exemplo, seja $t_i = t_j$, se $\exists o_i \in OP(t_i, P_{q_i})$ e $\exists o_j \in OP(t_j, P_{q_j})$, tal que o_i representa uma operação de *table scan* sobre a tabela t_i no plano de execução P_{q_i} e o_j representa uma operação de *table scan* sobre a tabela t_j no plano P_{q_j}, então, o valor atribuído à variável *pesoInteracao* será P_1;
- peso do tamanho da tabela: refere-se à influência do tamanho de uma determinada tabela t_k, sob o aspecto de espaço em disco ocupado, quando comparada com o valor da variável $tamanhoTotal$, que representa o tamanho total de todas as tabelas em T_{q_i} e T_{q_j}. Vale salientar que este peso pertence ao intervalo $[0,1]$.

Tabela 4.1: Utilização dos pesos de interação na abordagem IS

Peso	*Table scan*		Valor utilizado nos experimentos
	t_i	t_j	
P_1	Sim	Sim	0,5
P_2	Sim	Não	1
P_3	Não	Sim	0,25
P_4	Não	Não	0,5

O Algoritmo 1 se encerra, na linha 27, quando o valor da variável α, que representa o fator de interação entre q_i e q_j, é retornado. Vale ressaltar que $0 \leq \alpha \leq 1$ pelos seguintes motivos:

1. α é inicializada com o valor 0;
2. cada tabela aparece, no máximo, uma única vez em T_{q_i} e uma única vez em T_{q_j};
3. α pode ser incrementada somente quando o par de tabelas analisado é constituído de tabelas iguais;
4. os motivos 2 e 3 implicam que cada tabela pode possibilitar somente um incremento à α;
5. o peso de interação pertence ao intervalo $[0,1]$;
6. $\sum_{t_k \in T_{q_i} \cup T_{q_j}} \frac{tamanho(t_k)}{tamanhoTotal} = 1.$

Durante os experimentos, definimos os valores 0,5, 1, 0,25 e 0,5 para os pesos P_1, P_2, P_3 e P_4, respectivamente, conforme ilustrado na Tabela 4.1. Não foram realizados experimentos para avaliarem a melhor distribuição desses pesos. Entretanto, os valores destes pesos foram definidos de forma a respeitarem as seguintes restrições:

1. $P_3 < P_1 < P_2$;
2. $P_3 < P_4 < P_2$.

Sejam q_i e q_j duas instâncias de consulta executadas nessa ordem, e t_k a única tabela em comum entre q_i e q_j. A ideia dessas restrições é a de que o maior peso seja atribuído para o caso onde se supõe que a instância de consulta q_i acesse mais páginas de dados da tabela t_k (por *table scan*) do que q_j e, que a instância de consulta q_j recupere páginas de dados de t_k (somente por *index scan*) com uma vazão menor do que q_i. Já o menor peso seria atribuído para o caso inverso, em que q_i recuperaria menos páginas de dados da tabela t_k (somente por *index scan*) e, q_j recuperaria páginas de dados de t_k (por *table scan*) com uma vazão (páginas por segundo) maior do que q_i. Já os pesos P_1 e P_4 são considerados maiores que P_3 e menores que P_2 porque possuem parte dos benefícios do melhor caso, que é representado pelo peso P_2. Deste modo, quando o peso P_1 é aplicado no cálculo do fator de interação entre q_i e q_j, assume-se que a instância de consulta q_i acessa t_k por *table scan*, recuperando mais páginas de dados do que se acessasse esta tabela somente por *index scan*. Já no caso do peso P_4, a instância de consulta q_j acessa t_k somente por *index scan*, logo se supõe que a interação entre q_i e q_j seria maior do que se q_j acessasse t_k por *table scan*, à qual se supõe maior vazão.

4.2.2 *Data Retrieving Rate* (DRR)

Para computar o fator de interação entre duas instâncias de consulta q_i e q_j (instâncias dos tipos de consulta Q_{k_i} e Q_{k_j}, respectivamente), a abordagem *Data Retrieving Rate* (DRR) baseia-se nas instruções SQL e nos planos de execução de q_i e q_j, além da taxa de recuperação (acesso) de dados (taxa DRR) do disco para os tipos de consulta Q_{k_i} e Q_{k_j}, denominadas $DRR(Q_{k_i})$ e $DRR(Q_{k_j})$, respectivamente. Assim, para calcular as taxas DRR, uma etapa de pré-processamento se torna necessária.

Esta etapa de pré-processamento consiste em executar, para cada tipo de consulta Q_k, um conjunto A_k de n instâncias de consulta, m vezes. Após cada execução de uma instância de consulta $q_i \in A_k$, utilizamos uma ferramenta do sistema operacional Linux, denominada IOSTAT, que retorna a quantidade de dados transferidos do disco para a memória principal durante a execução de q_i, em KB. Denominamos esse valor de $DRR(q_i)$. Após as m execuções das n instâncias de consultas em A_k, calcula-se a média dos valores obtidos nas $m \times n$ execuções. Esse valor é considerado a taxa DRR para o tipo de consulta Q_k, ou seja, $DRR(Q_k)$. Em seguida, um vetor R contendo os valores das taxas DRR para cada tipo de consulta é construído. Nos experimentos realizados, fixamos $m = 5$ e avaliamos dois valores para n: $n = 1$ e $n = 5$.

De fato, a taxa DRR representa a quantidade de dados transferidos do disco para a memória principal por unidade de tempo (KB/s, em nossos experimentos). Seja t_k uma tabela e, q_i e q_j instâncias de consulta que acessam somente a tabela t_k. Suponha que q_i executa anteriormente à q_j. A abordagem DRR apresenta uma ideia similar à abordagem IS, pois ela considera que

1. operações de *table scan* sobre t_k acessam mais páginas de dados pertencentes à tabela t_k do que operações de *index scan*;
2. as páginas de dados de t_k são armazenadas sequencialmente em disco.

Entretanto, diferentemente da abordagem IS, a DRR busca melhorar o desempenho de consultas com baixa taxa DRR. Isto é feito atribuindo-se maior fator de interação em casos onde q_i acesse t_k por *table scan* (acessando todas as páginas de dados referentes à t_k) e q_j possua uma taxa DRR baixa em comparação com a taxa DRR dos demais tipos de consulta. Deste modo, supõe-se que quanto menor a taxa DRR de q_j, maior será a interação entre q_i e q_j quando as páginas de dados da tabela t_k estiverem em memória.

O Algoritmo 2 descreve os passos executados pela abordagem DRR para calcular o fator de interação entre duas instâncias de consulta q_i e q_j. Este algoritmo possui como entrada: i) duas instâncias de consulta q_i e q_j, onde se supõe que q_i inicia sua execução antes de q_j; ii) o tipo de consulta de q_j, definido como Q_{k_j} e iii) um vetor R com o valor da taxa DRR anteriormente calculado, para cada tipo de consulta. A variável global *sgbd* representa um *driver* que obtém informações da base de dados. A saída do algoritmo é o fator de interação entre q_i e q_j, denotado por $f(q_i, q_j)$. Seja R o vetor contendo as taxas DRR para cada tipo de consulta Q_k, $max(R)$ é o maior valor em R. Ademais, $DRR(Q_k)$ representa a taxa DRR de Q_k.

A linha 2 do Algoritmo 2 verifica se q_i ou q_j são nulos. Em caso positivo, o valor 0 é retornado (linha 3). Assim, são necessárias pelo menos duas consultas para que haja interação, ou seja, para que se possa medir o fator de interação. Seja T_{q_i} o conjunto de tabelas presentes em P_{q_i} (plano de execução de q_i) e, T_{q_j} o conjunto de tabelas existentes em P_{q_j} (plano de execução de q_j), nas linhas 5 e 6, as variáveis T_{q_i} e T_{q_j} são criadas a partir do método $interpretaConsulta$. Cada tabela $t_i \in T_{q_i}$ é representada por uma estrutura de dados, cujo tipo é denominado EDTabela (anteriormente definido). O mesmo vale para T_{q_j}.

Seja $tamanho(t_k)$ o tamanho de uma tabela qualquer t_k (em KB, por exemplo), representada pela estrutura de dados EDTabela, a variável $tamanhoTotal$ armazena a soma de

Algoritmo 2: Abordagem DRR

Entrada: vetor R de taxas DRR, instância de consulta q_i, instância de consulta q_j com seu respectivo tipo Q_{k_j}
Saída: fator de interação
Dados: sgbd

1 **início**
2 **se** `eNulo(`q_i`)` ***ou*** `eNulo(`q_j`)` **então**
3 **retorna** 0;
4 **fim se**
5 $T_{q_i} \leftarrow$ sgbd.`interpretaConsulta(`q_i`)`;
6 $T_{q_j} \leftarrow$ sgbd.`interpretaConsulta(`q_j`)`;
7 tamanhoTotal $\leftarrow \sum_{t_i \in T_{q_i}} tamanho(t_i) + \sum_{t_j \in T_{q_j}} tamanho(t_j) - \sum_{t_k \in T_{q_i} \cap T_{q_j}} tamanho(t_k)$;
8 $\alpha \leftarrow 0$;
9 **para cada** t_i ***em*** T_{q_i} **faça**
10 **para cada** t_j ***em*** T_{q_j} **faça**
11 relacao $\leftarrow t_i$.`obterRelacionamento(`t_j`)`;
12 **se** `ePossivelIntersecao(`relacao`)` **então**
13 **se** `temTableScan(`$OP(t_i, P_{q_i})$`)` **então**
14 pesoInteracao $\leftarrow P_1 \times \left(1 - \frac{DRR(Q_{k_j})}{max(R)}\right)$;
15 **senão**
16 pesoInteracao $\leftarrow P_2 \times \left(1 - \frac{DRR(Q_{k_j})}{max(R)}\right)$;
17 **fim se**
18 pesoTamanho $\leftarrow \frac{tamanho(t_i)}{\text{tamanhoTotal}}$;
19 $\alpha \leftarrow \alpha +$ pesoInteracao $\times$ pesoTamanho;
20 **fim se**
21 **fim para cada**
22 **fim para cada**
23 **retorna** α;
24 **fim**

$tamanho(t_k)$ para cada $t_i \in T_{q_i}$ e $t_j \in T_{q_j}$ (na linha 7), desconsiderando a ocorrência de repetições de tabelas. Estas informações são usadas para definir o peso de cada tabela no valor do fator de interação (linha 18).

O próximo passo do Algoritmo 2, entre as linhas 8 e 22, consiste em calcular o valor da variável α. Este processo começa com a inicialização da variável α na linha 8. Posteriormente, é verificado o relacionamento (possível interseção ou disjunção) entre cada par de tabelas t_i e t_j obtidos pelos laços aninhados das linhas 9 até 22. Nos casos classificados como possível interseção, o valor de α é modificado com base: nos pesos P_1 e P_2; nas operações $OP(t_i, P_{q_i})$ definidas para a tabela t_i no plano de execução P_{q_i} (linha 13); na taxa DRR de Q_{k_j} ($DRR(Q_{k_j})$), linhas 14 e 16; no tamanho da tabela t_i (linha 18) e no tamanho total das tabelas (linha 18). Os valores para as constantes P_1 e P_2 pertencem ao intervalo $[0,1]$ e, nos experimentos,

foram definidos como 1 e 0,5, respectivamente.

O fator de interação calculado pela abordagem DRR é o mesmo valor retornado pela variável α no fim do Algoritmo 2. Assim, $0 \leq \alpha \leq 1$ pelos seguintes motivos:

1. α é inicializado com o valor 0;
2. cada tabela aparece, no máximo, uma única vez em T_{q_i} e uma única vez em T_{q_j};
3. α pode ser incrementada somente quando o par de tabelas analisado é constituído de tabelas iguais;
4. os motivos 2 e 3 implicam que cada tabela pode possibilitar somente um incremento à α;
5. $0 \leq P_1 \leq 1$, $0 \leq P_2 \leq 1$ e $0 \leq \frac{DRR(Q_k)}{max(R)} \leq 1$;
6. o motivo 5 implica que $0 \leq P_l \times \left(1 - \frac{DRR(Q_k)}{max(R)}\right) \leq 1$, tal que $l \in \{1,2\}$ e $k \in \{k_i, k_j\}$. Consequentemente, o peso de interação pertence ao intervalo $[0,1]$;
7. $\sum_{t_k \in T_{q_i} \cup T_{q_j}} \frac{tamanho(t_k)}{tamanhoTotal} = 1.$

Vale salientar que a abordagem DRR tem o mesmo conceito de relacionamento entre tabelas que a abordagem IS. Além disso, as abordagens DRR e IS são baseadas na heurística em que consultas acessando tabelas por *table scan* carregam mais dados para a memória, permitindo, assim, que mais páginas de dados sejam reutilizadas. Por esta razão, durante os experimentos, P_1 foi definida como 1 e P_2 para 0,5. No entanto, na abordagem DRR, o valor do fator de interação α incorpora a taxa de recuperação de dados do disco. Isto ocorre porque abordagem DRR implementa a seguinte heurística: instâncias de consulta com DRRs menores devem ser executadas após instâncias que carregam mais dados em memória (através da operação de *table scan*). O parâmetro DRR é utilizado na sua forma normalizada que, basicamente, consiste de $\frac{DRR(Q_{k_j})}{max(R)}$.

4.2.3 *Greedy with Bidimentional Array* (GBA)

Para computar o fator de interação entre duas instâncias de consulta q_i e q_j (instâncias dos tipos de consulta Q_{k_i} e Q_{k_j}, respectivamente), a abordagem *Greedy with Bidimentional Array* (GBA) baseia-se em uma matriz bidimensional obtida por meio de uma etapa de pré-processamento. O Algoritmo 3 descreve essa etapa de pré-processamento.

Cada linha k_i dessa matriz representa um tipo de consulta Q_{k_i} e cada coluna k_j representa um tipo de consulta Q_{k_j}. Assim, a célula de localização $<k_i, k_j>$, com linha k_i e coluna k_j, armazena uma estimativa do ganho proporcionado por executar uma instância de consulta do tipo Q_{k_j} depois de uma instância de consulta de tipo Q_{k_i}. Este ganho, definido por G_{k_i,k_j}, pode ser calculado da seguinte maneira: $tempoExecucao(Q_{k_j}) - tempoExecucao(Q_{k_i}, Q_{k_j})$,

onde $tempoExecucao(Q_{k_j})$ é uma estimativa do tempo de execução de uma instância de Q_{k_j} executando com a memória livre e $tempoExecucao(Q_{k_i}, Q_{k_j})$ é uma estimativa do tempo de execução para uma instância de Q_{k_j} que começa sua execução imediatamente após a finalização da execução de uma instância de Q_{k_i}.

Algoritmo 3: Pré-processamento da GBA

```
   Entrada: conjunto D de instâncias de consulta, qtdTipos, qtdInstancias, iteracoes
   Saída: matriz G de ganhos por tipo de consulta
   Dados: sgbd
1  início
2      T ← criarVetor(qtdTipos);
3      G ← criarMatriz(qtdTipos);
4      para k_i ← 1 até qtdTipos faça
5          tempo_i ← 0;
6          para k_j ← 1 até qtdTipos faça
7              tempo_j ← 0;
8              para l_i ← 1 até qtdInstancias faça
9                  q_i ← D_{k_i,l_i};
10                 para l_j ← 1 até qtdInstancias faça
11                     q_j ← D_{k_j,l_j};
12                     para k ← 1 até iteracoes faça
13                         liberarMemoria();
14                         tempo_i ← tempo_i + sgbd.executarConsulta(q_i);
15                         tempo_j ← tempo_j + sgbd.executarConsulta(q_j);
16                     fim para
17                 fim para
18             fim para
19             G_{k_i,k_j} ← tempo_j / (qtdInstancias^2 × iteracoes);
20         fim para
21         T_{k_i} ← tempo_i / (qtdTipos × qtdInstancias^2 × iteracoes);
22     fim para
23     para k_i ← 1 até qtdTipos faça
24         para k_j ← 1 até qtdTipos faça
25             G_{k_i,k_j} ← T_{k_j} − G_{k_i,k_j};
26         fim para
27     fim para
28     retorna G;
29 fim
```

O Algoritmo 3 tem como entrada: i) D, um conjunto de instâncias de consulta usado para calcular G_{k_i,k_j}; ii) *qtdTipos*, o número de tipos de consultas; iii) *qtdInstancias*, o número de instâncias de consulta para cada tipo de consulta e iv) *iteracoes*, total de iterações (execuções) para cada par de instâncias de consulta. Adicionalmente, a variável *sgbd* representa um *driver* utilizado para obter informações do banco de dados (metadados).

Inicialmente, o Algoritmo 3 cria o vetor T (linha 2) e a matriz de ganhos G (linha 3). O vetor T é utilizado para armazenar o tempo de execução estimado para as instâncias de cada tipo de consulta Q_{k_i}, quando executadas com memória livre. Em seguida, este algoritmo irá inicializar as variáveis T e G (linhas 4 a 22). Para ilustrar como o Algoritmo 3 funciona, considere que Q_{k_i} e Q_{k_j} são os tipos de consulta que estão sendo analisados e que as instân-

cias de Q_{k_i} são executadas antes das instâncias de Q_{k_j}. Entre as linhas 8 e 18, cada par de instâncias $<q_i, q_j>$ é executado *iteracoes* vezes, da seguinte forma: primeiro, executa-se q_i com memória livre (linha 14) e, em seguida, executa-se q_j (linha 15), com o *cache* contendo as páginas de dados transferidas para a memória durante a execução de q_i. Então, a variável $tempo_i$ acumula o tempo de execução das instâncias de Q_{k_i}, onde cada instância $q_i \in D$ é executada $qtdTipos \times qtdInstancias \times iteracoes$ vezes. Já a variável $tempo_j$ acumula o tempo de execução das instâncias de Q_{k_j} quando executadas após as instâncias de Q_{k_i}, onde cada instância $q_j \in D$ é executada $qtdInstancias \times iteracoes$ vezes, sempre após a execução de uma instância de Q_{k_i}. Na linha 19 é calculada a média dos tempos de execução das instâncias de Q_{k_j} quando executadas após uma instância de Q_{k_i}. Esse valor é armazenado na célula $<k_i, k_j>$ da matriz G. Na linha 21 é calculada a média dos tempos de execução das instâncias de Q_{k_i} com memória livre. Esse valor é utilizado para inicializar a posição k_i do vetor T.

Note que nesse ponto do algoritmo cada célula $<k_i, k_j>$ da matriz G está armazenando o $tempoExecucao(Q_{k_i}, Q_{k_j})$. Contudo, cada célula $<k_i, k_j>$ da matriz G deveria armazenar o ganho G_{k_i,k_j}, onde $G_{k_i,k_j} = tempoExecucao(Q_{k_j}) - tempoExecucao(Q_{k_i}, Q_{k_j})$. As linhas de 23 a 27 fazem o ajuste necessário para que os valores corretos sejam armazenados na matriz G. Finalmente, esta matriz é retornada (linha 28).

Algoritmo 4: Abordagem GBA

Entrada: identificador k_i do tipo de consulta Q_{k_i}, identificador k_j do tipo de consulta Q_{k_j}, matriz G de ganhos por tipo de consulta
Saída: fator de interação

```
1 início
2     α ← 0;
3     se não existeTipo(k_i) e não existeTipo(k_j) então
4         α ← G_{k_i,k_j} / max(G);
5     fim se
6     retorna α;
7 fim
```

Uma vez concluída a etapa de pré-processamento, tendo sido gerada a matriz de ganhos G, a execução da abordagem GBA (Algoritmo 4) pode ser iniciada. O Algoritmo 4 possui como entrada: i) os identificadores k_i e k_j dos tipos de consulta Q_{k_i} e Q_{k_j}, respectivamente; e ii) G, que é a matriz bidimensional com os ganhos em desempenho (gerada na saída da etapa de pré-processamento, pelo Algoritmo 3).

De acordo com os passos do Algoritmo 4, da mesma forma que nas abordagens anteriores, α é inicializada com o valor 0, como mostrado na linha 2. Seja $max(G)$ o valor máximo armazenado na matriz G. Posteriormente, na condição descrita na linha 3, é verificado se k_i e k_j são identificadores de tipos de consulta existentes. Neste caso, na linha 4, o valor da variável α é obtido por $\frac{G_{k_i,k_j}}{max(G)}$, onde k_i é a linha da matriz G e k_j, a coluna. O valor de α é obtido por $\frac{G_{k_i,k_j}}{max(G)}$ para que esteja dentro do intervalo $[0,1]$, respeitando, assim, a definição de fator de interação (Definição 4.2.1). Por fim, o fator de interação é retornado (linha 6).

Vale destacar que para computar o fator de interação entre duas instâncias de con-

sulta q_i e q_j basta executar o Algoritmo 4 passando como parâmetros os identificadores dos seus respectivos tipos de consulta na matriz G de ganhos, ou seja, k_i e k_j.

4.3 ISO: Um Escalonador Baseado nas Interações entre Consultas

Com a finalidade de avaliar a utilização das abordagens propostas para modelar e medir as interações entre consultas (IS, DRR e GBA), desenvolvemos um escalonador baseado nas interações entre consultas, denominado ISO (*An Intelligent Scheduler for Multiple-query Execution Ordering*), voltado para a tomada de decisão em tempo real, mas que facilmente pode ser adaptado a cargas de trabalho em lote. Este escalonador mantém uma fila de escalonamento e utiliza o fator de interação entre consultas (Definição 4.2.1) para definir a ordem de execução das instâncias de consulta. Vale salientar que o ISO não soluciona o problema de negação de serviço (*starvation*), uma vez que este não é o foco deste trabalho. Além disso, o objetivo do ISO é otimizar o tempo de resposta de uma determinada carga de trabalho, escolhendo dinamicamente, para cada instância de consulta recebida, a posição na fila de escalonamento que provê o maior ganho em interação (Definição 4.3.1). O ganho em interação $gi(q_i, q_j, q_{j+1})$ é calculado pela Fórmula 4.1.

Definição 4.3.1 (Ganho em interação) *Seja q_i uma consulta a executar em um dado SGBD. O ganho em interação de q_i representa o ganho estimado, em fator de interação, trazido pela execução de q_i após uma consulta q_j e antes de uma consulta q_{j+1}.*

A Fórmula 4.1 supõe que $f(q_i, q_j)$ representa o cálculo do fator de interação, onde q_i e q_j são instâncias de consultas. Além disso, supõe-se que q_i executa antes de q_j. Vale destacar que o cálculo do fator de interação pode ser realizado pelos algoritmos 1, 2 e 4.

$$gi(q_i, q_j, q_{j+1}) = \begin{cases} f(q_i, q_{j+1}), & \text{se } q_j \text{ é nula} \\ f(q_j, q_i), & \text{se } q_{j+1} \text{ é nula} \\ (f(q_j, q_i) + f(q_i, q_{j+1})) - f(q_j, q_{j+1}), & \text{caso contrário} \end{cases} \tag{4.1}$$

As etapas implementadas pelo ISO são ilustradas no Algoritmo 5. O Algoritmo 5 requer dois parâmetros de entrada: a fila de escalonamento F e a instância de consulta q_i a ser escalonada. A fila de escalonamento F é ilustrada na Figura 4.1. Seja n a quantidade de consultas em F. Deste modo, a posição 0 de F (primeira posição) é reservada para a última consulta já submetida à execução. A posição 1 (segunda posição), por sua vez, armazena a próxima consulta a ser executada, a posição $n-1$ se refere à última consulta em F a ser executada e a posição n consiste na primeira posição livre em F. Por exemplo, suponha $n = 10$. Assim, a fila F tem 9 consultas ($n-1$ consultas) para executar e a primeira posição livre em F é a posição 10 (posição n). Além disso, novas consultas podem ser inseridas entre as posições 1 e n, inclusive. Ademais, sempre que uma consulta q_i for inserida em uma determinada posição da fila F, por conseguinte, cada consulta localizada após q_i, na fila F, terá sua posição em F incrementada

Algoritmo 5: Escalonamento baseado na interação entre consultas

Entrada: fila de escalonamento F, instância de consulta q_i a ser escalonada

```
1  início
2      Δ_interacao ← gi(q_i, q_{|F|-1}, nulo);
3      j ← |F|;
4      para l ← 1 até |F| − 1 faça
5          se Δ_interacao < gi(q_i, q_{l-1}, q_l) então
6              Δ_interacao ← gi(q_i, q_{l-1}, q_l);
7              j ← l;
8          fim se
9      fim para
10     F.inserir(q_i, j);
11 fim
```

em uma unidade. No decorrer do trabalho, assumiremos que $|F| = n$, ou seja, que terão $|F|$ consultas na fila F.

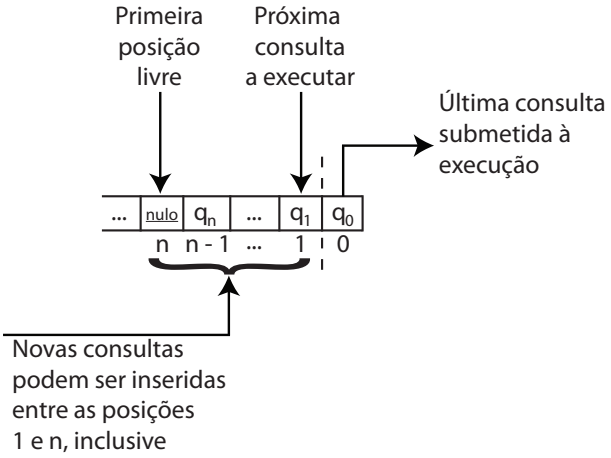


Figura 4.1: Fila de escalonamento

Seja F a fila de escalonamento, q_i a consulta a ser escalonada, j a posição em F para q_i ser inserida e, $q_l \in F$ a consulta na posição l da fila F. A linha 2 do Algoritmo 5 realiza a inicialização da variável $\Delta_{interacao}$, que é responsável por armazenar o valor do maior ganho em interação. Em seguida, na linha 3, a posição j é inicializada com a posição $|F|$ da fila F. Das linhas 4 a 9, q_i é testada em cada posição l da fila F. A instância de consulta q_i ocupará a posição de F que fornecer o maior ganho em interação (linha 6), que é calculada de acordo com a Fórmula 4.1. Essa posição fica armazenada na variável j. Finalmente, na linha 10, q_i é inserida na posição de F com maior ganho em interação.

4.4 Experimentos

A fim de avaliar as três abordagens propostas para modelar e medir as interações entre consultas, bem como o escalonador ISO, escolhemos um cenário de teste em que as con-

sultas são executadas de forma sequencial. A escolha para tal cenário foi motivada pela discussão sobre a necessidade de previsibilidade no paradigma atual para o problema de otimização (FLORESCU; KOSSMANN, 2009). A ideia é garantir que as consultas sejam executadas num período de tempo aceitável, não o mais rapidamente possível. De acordo com esta abordagem, otimizando para o percentil 99 é mais importante do que a média. Assim, por exemplo, no caso em que cada consulta possui um tempo máximo de resposta, uma solução que é otimizada para o percentil 99 é mais adequada. Um exemplo disso é mostrado em (CHI et al., 2011), que descreve uma solução baseada em acordos chamados SLAs (Acordos de Nível de Serviço), com uma métrica específica: tempo de resposta das consultas. Neste caso, cada consulta tem que ser executada em um período de tempo definido para gerar lucro para o provedor do serviço responsável pela execução das consultas, caso contrário, penalidades podem ser aplicadas ao provedor.

Quando cada consulta tem um tempo máximo para ser executada, o uso de uma solução concorrente pode ser uma maneira ineficiente para agendar consultas devido à complexidade na previsão de seus tempos de resposta se comparado com abordagens sequenciais. Muitos fatores, tais como, algoritmo de escalonamento de processos do sistema operacional e o impacto da utilização de recursos durante a execução das consultas interferem na eficácia de uma solução concorrente. De fato, as soluções concorrentes podem diminuir o tempo de resposta da carga de trabalho. Contudo, é mais difícil, para as soluções concorrentes do que para as soluções sequenciais, controlar o cumprimento do prazo de execução estipulado para cada consulta ou garantir o lucro em um ambiente que utiliza o tempo de resposta das consultas como métrica de SLA, por exemplo. Portanto, para estes contextos, uma solução sequencial se mostra mais viável do que uma concorrente.

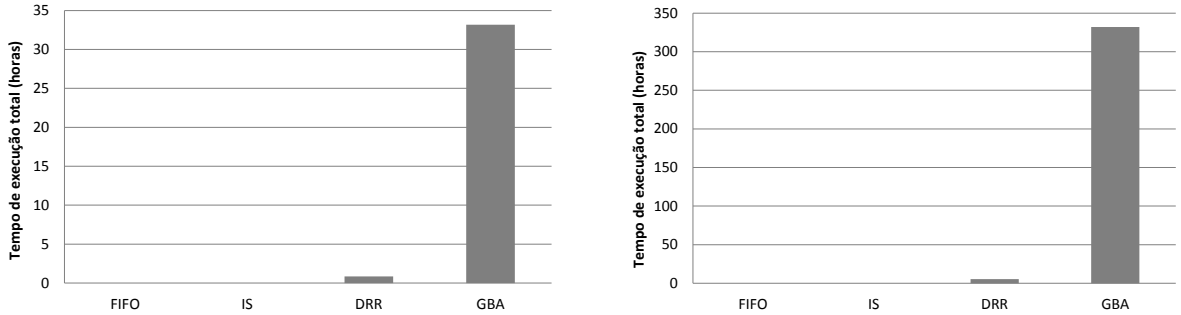
4.4.1 Configuração do Ambiente de Testes

Uma máquina Intel Core 2 Duo de 2 GHz com 3 GB de RAM e 500 GB de HD, usando Ubuntu 11.10 de 64 bits como sistema operacional, foi utilizada para executar os experimentos. Os testes foram executados implementando o benchmark TPC-H com fator de escala de 2 GB, utilizando o projeto DBGEN/DBT-3, em um SGBD PostgreSQL 9.1.3.

Os experimentos foram realizados utilizando duas cargas de trabalho diferentes, em função do número de consultas a serem escalonadas. Na primeira carga de trabalho, uma instância de consulta foi criada para cada tipo de consulta TPC-H, totalizando 22 consultas. Estas instâncias foram geradas atribuindo diferentes valores para os parâmetros do *template*, usando QGEN/DBT-3. Depois disso, 10 diferentes ordens de execução foram geradas aleatoriamente, cada uma com 22 instâncias de consulta. Para esta carga, o vetor de taxas DRR usado pela abordagem DRR foi criado utilizando-se $n = 1$. Na carga segunda carga de trabalho, cinco instâncias de consulta foram criadas para cada tipo de consulta TPC-H, totalizando 110 instâncias. Para esta carga, o vetor de taxas DRR usado pela abordagem DRR foi criado utilizando-se $n = 5$. Além disso, 20 ordens de execução diferentes foram produzidas aleatoriamente. Vale notar que as diferentes ordens geradas simulam as ordens nas quais as consultas estão chegando para serem executadas. Para cada ordem de execução definida, a abordagem FIFO foi utilizada para enviar as consultas ao escalonador ISO.

4.4.2 Resultado dos Testes

Nesta seção, apresentamos os resultados dos experimentos que realizamos. Para isso, foram utilizadas quatro métricas: tempo de execução para o pré-processamento, tempo de execução para realizar o escalonamento de consultas; eficácia e eficiência. Uma vez que as abordagens DRR e GBA requerem uma etapa de pré-processamento, utilizamos uma métrica para o tempo de execução do pré-processamento. A Figura 4.2a ilustra o resultado considerando esta métrica para a primeira carga de trabalho (22 instâncias de consulta e 10 ordens de chegada aleatórias). Por sua vez, a Figura 4.2b traz os resultados para a segunda carga de trabalho (110 consultas e 20 ordens de chegada aleatórias).

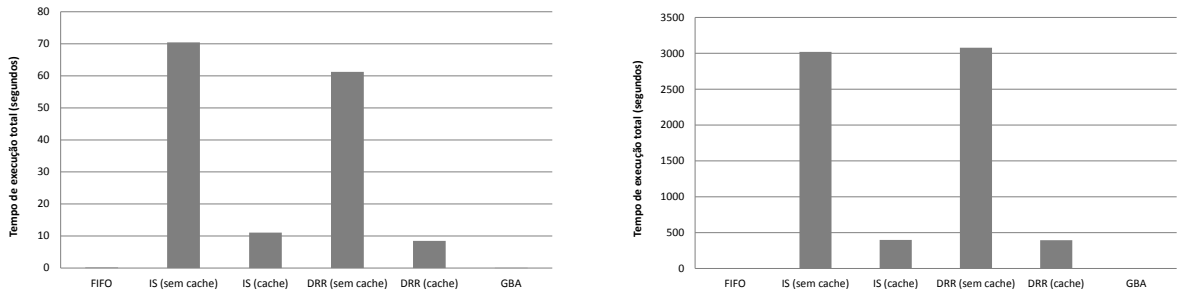


(a) Primeira carga de trabalho (b) Segunda carga de trabalho

Figura 4.2: Análise comparativa entre IS, DRR, GBA e FIFO, considerando o tempo gasto na fase de pré-processamento

Seja W uma carga de trabalho e, $q_i \in W$ e $q_j \in W$ duas instâncias de consulta. A segunda métrica tem o objetivo de especificar quanto tempo cada abordagem leva para escalonar as consultas das cargas de trabalho analisadas. A Figura 4.3a apresenta os resultados para a primeira carga de trabalho e a Figura 4.3b, para a segunda carga de trabalho. Desta forma, as abordagens FIFO e GBA escalonaram as consultas com um tempo desprezível porque estas abordagens não realizam consultas no SGBD. Por outro lado, as abordagens IS e DRR, para calcular o fator de interação entre q_i e q_j, buscam no SGBD o plano de execução dessas instâncias e, desta forma, consomem um tempo relevante para escalonar estas instâncias de consulta. Assim, com o intuito de otimizar o tempo decorrido pelo escalonador ISO para definir a ordem de execução de instâncias de consulta, desenvolvemos uma estratégia com cache para as abordagens IS e DRR. Nesta estratégia, as informações sobre o plano de execução de q_i ou q_j são obtidas do SGBD e, posteriormente, armazenadas em memória. Assim, o SGBD é acessado somente uma vez para cada consulta $q_i \in W$ durante a etapa de escalonamento. Deste modo, toda vez que estas informações forem necessárias para o cálculo do fator de interação entre consultas, durante o escalonamento, elas serão reaproveitadas. Consequentemente, o banco será acessado menos vezes do que na estratégia sem cache, na qual as informações sobre o plano de execução das consultas não são armazenadas em memória para posterior reaproveitamento.

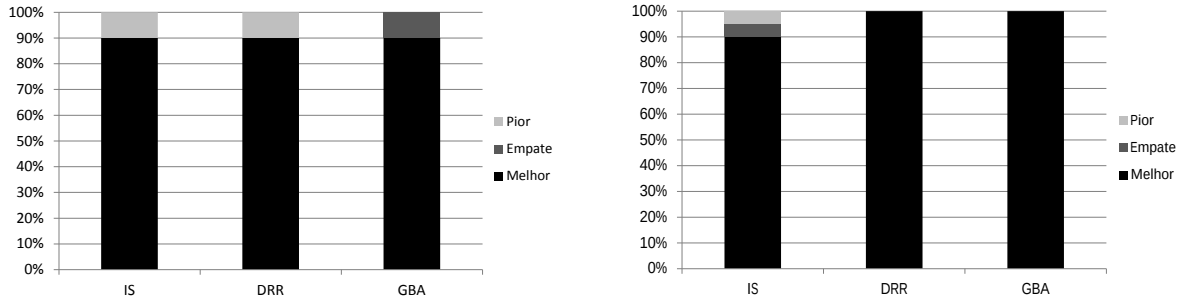
A métrica de eficácia foi utilizada para comparar o número de vezes que cada abordagem de escalonamento apresentou um resultado melhor do que a abordagem FIFO (as consultas são executadas de acordo com a ordem de chegada aleatória), a qual foi utilizada como *baseline*. Consideramos "Empate" quando a diferença absoluta entre os tempos de execução é



(a) Primeira carga de trabalho (b) Segunda carga de trabalho

Figura 4.3: Análise comparativa entre IS, DRR, GBA e FIFO, considerando o tempo gasto na fase de escalonamento

inferior ou igual a um minuto, na primeira carga de trabalho, e inferior ou igual a cinco minutos, na segunda carga de trabalho. Deste modo, consideramos cerca de 3 segundos de margem de erro para cada instância de consulta, nos dois casos. A Figura 4.4a ilustra a eficácia das três abordagens propostas para a primeira carga de trabalho. IS e DRR apresentaram melhor desempenho em 90% dos casos, e GBA foi melhor do que FIFO em todos os casos. Apesar das abordagens IS e DRR apresentarem a mesma eficácia, quando analisado o tempo de execução total dessas duas abordagens, DRR levou vantagem sobre IS (conforme será explicitado na análise da métrica de eficiência). Quanto à eficácia para a segunda carga de trabalho (Figura 4.4b), apenas a abordagem IS não foi melhor que FIFO em todos os casos. As abordagens DRR e GBA tiveram a mesma eficácia para a segunda carga de trabalho, entretanto a melhoria do tempo de execução total obtida pela abordagem GBA, considerando as duas cargas de trabalho e utilizando FIFO como *baseline*, foi muito mais significativa, em termos percentuais, do que a melhoria obtida pela abordagem DRR. Constatou-se 40% de redução do tempo da GBA na primeira carga de trabalho contra 33% da DRR na segunda carga de trabalho.



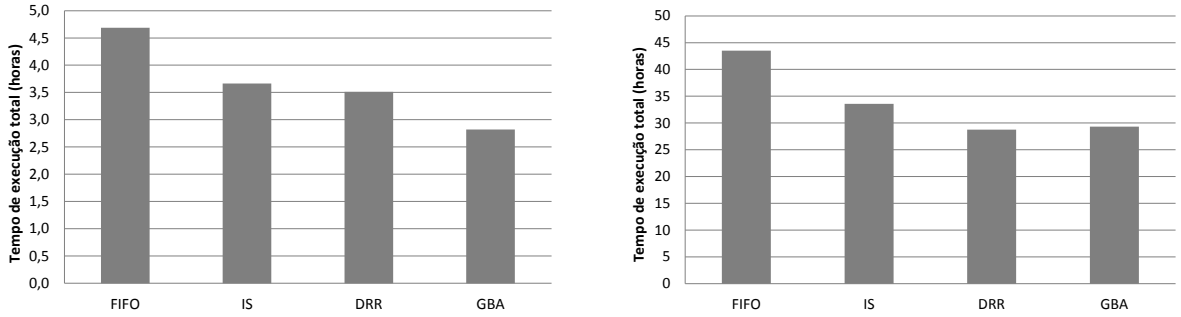
(a) Primeira carga de trabalho (b) Segunda carga de trabalho

Figura 4.4: Análise comparativa entre IS, DRR, GBA e FIFO, considerando a métrica de eficácia

A métrica de eficiência foi aplicada para medir o tempo total consumido para executar consultas usando os escalonamentos produzidos pelas três abordagens propostas (IS, DRR e GBA) e pela estratégia FIFO. Como a Figura 4.5a (resultados para a primeira carga de trabalho) ilustra, IS foi executada em 3,7 horas, poupando uma hora em relação ao tempo de execução da abordagem FIFO (22% de redução). DRR teve um melhor tempo de resposta, uma vez que o escalonamento produzido pela DRR foi executado em 3,5 horas (25% de redução). No en-

tanto, a estratégia mais eficiente foi GBA com redução de 40%, economizando 1,9 horas, sendo, portanto, a única solução com o tempo de execução menor que 3 horas.

A Figura 4.5b traz os resultados sobre a métrica de eficiência em relação à execução da segunda carga de trabalho (110 consultas e 20 ordens de chegada aleatórias). IS foi executada em 33,6 horas, economizando 9,9 horas com relação ao tempo de execução da abordagem FIFO (redução de 23%). DRR teve o melhor tempo de resposta, com duração de 28,8 horas (redução de 33%). No entanto, GBA também apresentou um resultado relevante, pois economizou 14,7 horas, representando 33% de redução, comparando-se com o tempo de execução obtido por FIFO.



(a) Primeira carga de trabalho (b) Segunda carga de trabalho

Figura 4.5: Análise comparativa entre IS, DRR, GBA e FIFO, considerando a métrica de eficiência

Os resultados apresentados na Figura 4.6 destacam os benefícios de se considerar a interação entre consultas. Para este experimento, nós obtivemos o tempo de execução das seis consultas TPC-H com maior tempo de execução quando executadas por meio dos escalonamentos produzidos pelas três abordagens propostas (IS, DRR e GBA) e FIFO. Vale notar que para cada uma dessas consultas, o tempo de execução, utilizando as abordagens IS, DRR e GBA, é menor do que quando essas consultas são executadas pela abordagem FIFO. Observe que em FIFO interação entre consultas não é considerada.

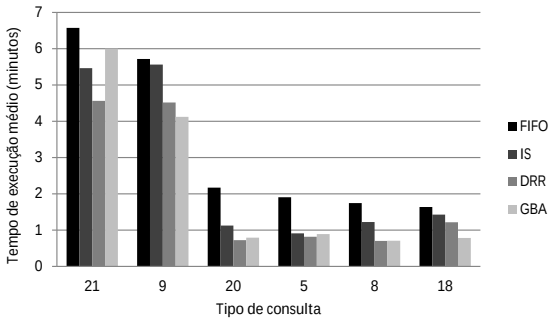


Figura 4.6: Avaliação de desempenho por tipo de consulta

5 GEDANIC: UM *FRAMEWORK* PARA O GERENCIAMENTO DE BANCOS DE DADOS EM NUVEM BASEADO NAS INTERAÇÕES ENTRE CONSULTAS

No Capitulo 3, foram apresentados e discutidos três dos principais problemas relacionados ao gerenciamento de dados em nuvem: despacho, escalonamento de consultas e planejamento de capacidade ou provisionamento de recursos. Buscando solucionar esses problemas de maneira satisfatória, propomos, neste capítulo, um *framework*, denominado GeDaNIC (*Framework* para o Gerenciamento de Banco de Dados em Nuvem Baseado nas Interações entre Consultas). Vale salientar que o problema de negação de serviço (*starvation*) não é abordado pelo GeDaNIC porque não é o foco deste trabalho. Adicionalmente, O *framework* proposto possui as seguintes características:

- **não intrusivo:** o GeDaNIC não requer alterações no código fonte do SGBD utilizado. Isso possibilita que o *framework* proposto seja independente de SGBD, ou seja, possa ser utilizado com qualquer SGBD;
- **utiliza um modelo de custo genérico:** no GeDaNIC, a forma de calcular o custo de execução de uma determinada consulta ou carga de trabalho baseia-se no custo econômico (medido, por exemplo, em dólar) referente aos recursos utilizados para sua execução e não mais no custo computacional (número de I/Os, etc.). Este fato decorre da utilização dos conceitos de serviço medido e "pagamento pelo uso";
- **baseia-se em SLAs:** para garantir a qualidade de serviço, os sistemas de banco de dados (SBDs) em nuvem, em geral, utilizam o conceito de SLA (definido na Seção 2.3). O SLA fornece informações sobre o nível de qualidade esperado para o serviço de dados, o qual pode ser especificado em termos de: disponibilidade, tempo de resposta, vazão, dentre outros. Além disso, o SLA especifica também penalidades em caso de violação do nível de qualidade contratado (SOUSA et al., 2010). Assim, o *framework* proposto baseia-se na utilização do conceito de SLA (conforme será discutido na Seção 5.1);
- **orientado ao lucro:** do ponto de vista do provedor do serviço de dados, o objetivo principal de uma solução de gerenciamento de dados em nuvem consiste em maximizar o lucro total do serviço, ou seja, minimizar o seu custo. Deste modo, o GeDaNIC busca minimizar os recursos necessários para executar as cargas de trabalho, respeitando o nível de qualidade especificado no SLA;
- **monitoramento e ajuste dinâmico do ambiente:** o GeDaNIC monitora o SBD em nuvem e verifica se o SLA será atendido satisfatoriamente. Caso o *framework* proposto verifique que os SLAs não serão atendidos adequadamente, este irá ajustar o SBD em nuvem automaticamente (criando novas instâncias/cópias do SBD, por exemplo), com a finalidade cumprir os SLAs, evitando multas que venham a diminuir o lucro;
- **suporta cargas inesperadas:** as cargas de trabalho dos SBDs em nuvem são muitas vezes sazonais. Assim, o GeDaNIC busca manter seu desempenho dentro dos valores contratados nos SLAs, mesmo na ocorrência de um aumento inesperado da demanda, uma vez que uma das características da nuvem é a elasticidade rápida;

- **baseia-se na interação entre consultas:** o GeDaNIC modela e mede a interação entre consultas (definida na Seção 2.4) e utiliza essa propriedade na resolução dos problemas de despacho, escalonamento e planejamento de capacidade. Assim, o *framework* proposto procura reduzir o tempo de execução das cargas de trabalho submetidas ao serviço de dados e, consequentemente, aumentar o lucro do provedor deste serviço;

- **suporta a replicação de dados:** o GeDaNIC permite que réplicas completas do SBD sejam instanciadas dinamicamente e utilizadas para processar a carga de trabalho. Esta estratégia é fundamental para aumentar a disponibilidade dos dados e o desempenho do SBD em nuvem.

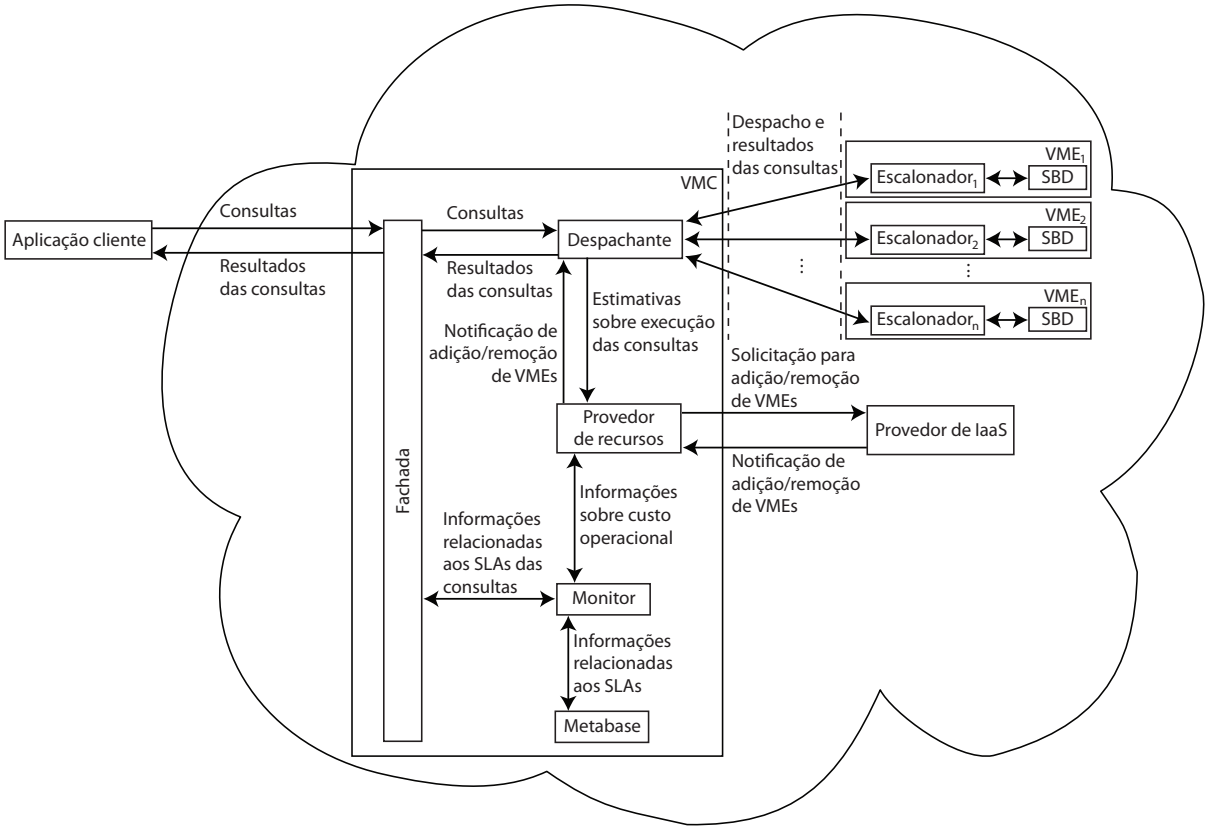


Figura 5.1: Arquitetura do GeDaNIC

A Figura 5.1 ilustra a arquitetura do *framework* GeDaNIC. Esta arquitetura é composta por:

- **VM controladora (VMC):** essa VM hospeda os módulos responsáveis pelo tratamento dos problemas de despacho (módulo despachante) e provisionamento de recursos (módulo provedor de recursos), bem como pela interface com o usuário (módulo fachada) e pela coleta de estatísticas (módulo monitor). Além disso, as informações relativas aos SLAs são armazenadas em um componente denominado metabase. Vale ressaltar que o GeDaNIC utiliza apenas uma instância da VMC;

- **VM escalonadora (VME):** cada VME hospeda uma instância do módulo escalonador e uma réplica completa do SBD. O módulo escalonador é responsável por definir a ordem de execução das consultas a ele direcionadas (despachadas) e por, efetivamente, enviar essas consultas para serem executadas na réplica do SBD hospedada localmente na VME. Vale destacar que podem existir diversas VMEs, ou seja, diversas instâncias da VM escalonadora.

O *framework* GeDaNIC, conforme se pode observar na Figura 5.1 é composto por cinco módulos distintos. A seguir descreveremos cada um desses módulos:

- **fachada:** as aplicações cliente enviam requisições ao módulo fachada solicitando a execução de determinadas consultas. Além disso, este módulo recebe do módulo monitor os dados relacionados ao cumprimento dos SLAs (tipo de consulta, receita, SLO e penalidade) das consultas a serem executadas. Cada consulta (requisição) recebida pelo módulo fachada é redirecionada ao módulo despachante;

- **despachante:** o módulo despachante tem por objetivo decidir qual instância do módulo escalonador executará uma determinada consulta. Além disso, o despachante envia informações relativas à execução de consultas, as quais ocorrem nas diferentes instâncias do módulo escalonador, ao módulo provedor de recursos;

- **escalonador:** cada instância do módulo escalonador gera as estimativas utilizadas pelo módulo despachante, bem como define a ordem de execução das consultas a ela direcionadas. Além disso, cada instância do módulo escalonador submete as consultas sob sua responsabilidade à réplica local do SBD, a fim de que esta as execute. Outra responsabilidade desse módulo é a de enviar o resultado da execução das consultas ao módulo despachante, o qual retorna esse resultado para o módulo fachada;

- **provedor de recursos:** o módulo provedor de recursos decide, baseado em informações fornecidas pelo módulo despachante, sobre a necessidade de instanciação/suspensão de VMEs;

- **monitor:** o módulo monitor armazena na metabase informações relativas ao cumprimento dos SLAs acordados com a aplicação cliente e sobre os custos operacionais originados pela utilização de VMs.

Complementando a arquitetura do GeDaNIC, existe um componente denominado metabase, o qual se localiza na VMC. O objetivo da metabase consiste em armazenar as informações coletadas pelo módulo monitor. Adicionalmente, dois componentes externos à arquitetura proposta atuam para o funcionamento do *framework* GeDaNIC. São eles: a aplicação cliente e o provedor de IaaS. O primeiro envia a solicitação de execução de consultas ao serviço de dados na nuvem, já o segundo refere-se ao provedor de infraestrutura, que fornece as VMs utilizadas pelo GeDaNIC.

O fluxo de execução do GeDaNIC inicia-se com a submissão pela aplicação cliente de uma dada consulta q_i, em SQL, para o módulo fachada, que, por sua vez, envia q_i para o

módulo despachante *md*. Em seguida, o módulo *md* envia a consulta q_i, juntamente com seu SLA, para cada instância do módulo escalonador. Isto ocorre a fim de se obter estimativas que auxiliem o *md* na tomada de decisão sobre qual instância do módulo escalonador será responsável pela execução q_i. Escolhida esta instância me_k, q_i é despachada a ela. A função principal da instância me_k consiste em definir o momento em que q_i será executada na réplica local do SBD vinculada à me_k. Quando q_i termina sua execução, seu resultado é enviado por me_k ao módulo despachante, que o redireciona para o módulo fachada, o qual envia à aplicação cliente o resultado da execução da consulta q_i.

Durante a execução de q_i, ocorrem outros passos, também descritos na Figura 5.1. Um deles está relacionado ao envio de estimativas, do módulo despachante para o módulo provedor de recursos, sobre a execução das consultas. Esta informação é utilizada na tomada de decisão sobre a solicitação de instanciação/suspensão de VMEs para o provedor de IaaS. Outra etapa da arquitetura do GeDaNIC é o armazenamento de informações referentes ao cumprimento dos SLAs na metabase pela instância do módulo monitor. Estas informações dizem respeito ao ganho financeiro com as execuções das consultas submetidas pela aplicação cliente, bem como, aos gastos originados pela utilização das VMEs.

O restante do capítulo é iniciado pela Seção 5.1, que descreve o modelo de SLA utilizado pelo GeDaNIC. Em seguida, a solução para o problema de despacho é detalhada na Seção 5.2, o modo como o *framework* proposto lida com o escalonamento de consultas é discorrido na Seção 5.3 e, na Seção 5.4, há o detalhamento da solução utilizada para o problema de planejamento de capacidade. Por fim, a Seção 5.5 apresenta uma análise comparativa entre o GeDaNIC e as abordagens Roundrobin/FIFO e SLA-Tree.

5.1 SLA

O SLA é um acordo entre duas partes: o cliente e o provedor de um determinado serviço (COMELLAS; PRESA; FERNÁNDEZ, 2010), cuja finalidade é garantir a qualidade do serviço contratado. O *framework* GeDaNIC utiliza o conceito de SLA para garantir a qualidade do serviço de dados. O SLA utilizado no GeDaNIC tem como métrica o tempo de resposta das consultas (tempo decorrido desde a chegada de uma dada consulta no serviço de dados até o início do envio ao cliente do resultado obtido por sua execução). Porém, diferentemente de (SOUSA et al., 2012), o SLO (por exemplo, tempo de reposta menor do que 10 minutos), a receita e a penalidade são definidos para cada tipo de consulta Q_k. Assim, cada instância q_i de Q_k possui os mesmos valores de SLO, receita e penalidade definidos para o tipo Q_k. Já em (SOUSA et al., 2012) a definição dos valores de SLO, receita e penalidade é a mesma para todas as instâncias de consulta, independentemente do seu tipo. A escolha dessa nova definição para o SLA foi motivada pelo fato de a maioria das aplicações atualmente disponibilizadas na nuvem apresentarem características OLAP (Online Analytical Processing), o que envolve cargas de trabalho cujas consultas apresentam tempos de resposta de ordens de grandeza diferentes (por exemplo, enquanto algumas consultas demoram 16 segundos, outras demoram 15 minutos).

Para exemplificar a utilização do SLA no *framework* proposto, considere uma carga de trabalho W. Seja R_W a receita referente à W, C_W o custo operacional (custo com utilização

de VMs) para a execução de W e, P_W as penalidades impostas ao provedor do serviço de dados após finalizar a execução de W. O lucro obtido pela execução de W, representado por L_W, é definido pela seguinte fórmula:

$$L_W = R_W - (C_W + P_W) \tag{5.1}$$

A receita R_W consiste na soma das receitas obtidas com todas as consultas $q_i \in W$, independentemente do cumprimento dos SLAs. Seja a receita por consulta r_{q_i} o valor monetário acordado entre o cliente e o provedor do serviço de dados para que uma determinada consulta $q_i \in W$ seja executada, R_W é obtido pela seguinte fórmula:

$$R_W = \sum_{q_i \in W} r_{q_i} \tag{5.2}$$

Assim como em (SILVA et al., 2012), o tempo foi discretizado em unidades faturáveis (por exemplo, no Amazon EC2[1] a receita é calculada a cada hora). Porém, o modelo de custo operacional foi simplificado, assumindo-se que cada instância de VM terá um gasto fixo a cada unidade faturável, em função do seu tipo. Por exemplo, cada instância do tipo *Small* (tipo de VM do Amazon EC2[2]) incidirá em um custo operacional fixo (por exemplo, de R$ 2,00 por hora).

Seja H o conjunto de possíveis tipos de VMs, E_W o tempo de execução da carga de trabalho W (usando as n VMEs existentes e a VMC), E'_W o tempo E_W convertido para a unidade faturável utilizada, $n_W(h,t)$ a quantidade de VMs instanciadas do tipo h em uma unidade faturável t de tempo e, $c(h)$ o custo de se utilizar uma VM do tipo $h \in H$ por unidade faturável de tempo. Dessa forma, suponha que a unidade faturável seja de uma hora e $E_W = 45$ min. Neste caso, $E'_W = 1$ hora, pois o pagamento é contabilizado a cada hora. Então, o custo operacional C_W é calculado pela seguinte fórmula:

$$C_W = \sum_{h \in H} \sum_{t=1}^{E'_W} n_W(h,t) \times c(h) \tag{5.3}$$

O não cumprimento dos SLAs associados às instâncias de consulta de W leva ao pagamento de penalidades P_W, por parte do provedor do serviço de dados, para o cliente. Seja p_{q_i} a penalidade associada a uma dada instância de consulta $q_i \in W$, P_W é definida pela seguinte fórmula:

$$P_W = \sum_{q_i \in W} p_{q_i} \tag{5.4}$$

Seja tr_{q_i} o tempo de resposta de uma dada instância de consulta q_i, ou seja, o tempo decorrido desde sua chegada ao módulo fachada até o início do envio dos resultados de sua

[1] http://aws.amazon.com/ec2/pricing/

[2] http://aws.amazon.com/ec2/instance-types/

execução ao cliente. Seja o objetivo de nível de serviço SLO_{q_i} associado à q_i, o tempo máximo aceitável para que não haja uma penalidade relacionada à q_i e, r_{q_i} a receita associada à consulta q_i. Assim, a fórmula a seguir descreve o modo como p_{q_i} é calculada. Vale salientar que outros modelos de penalidade podem ser utilizados.

$$p_{q_i} = \begin{cases} r_{q_i}, & \text{se } tr_{q_i} > SLO_{q_i} \\ 0, & \text{caso contrário} \end{cases} \tag{5.5}$$

Derivado do conceito de penalidade, utilizamos também o conceito de penalidade estimada. Seja q_i uma instância de consulta, esta penalidade estimada se refere ao valor monetário que será pago pelo provedor do serviço de dados ao cliente, caso a consulta q_i não seja executada no tempo de resposta estimado tre_{q_i}. Seja r_{q_i} a receita acordada para consulta q_i e, SLO_{q_i} o objetivo de nível de serviço associado à instância de consulta q_i. Assim, a penalidade estimada $pe_{q_i}(tre_{q_i})$, em função do tempo de resposta estimado tre_{q_i} de q_i, é definida pela seguinte fórmula:

$$pe_{q_i}(tre_{q_i}) = \begin{cases} r_{q_i}, & \text{se } tre_{q_i} > SLO_{q_i} \\ 0, & \text{caso contrário} \end{cases} \tag{5.6}$$

O tempo de resposta estimado de uma dada consulta q_i pode ser calculado em função do instante de tempo estimado tie de início da execução de q_i. Assim, consideremos $tre_{q_i}(tie)$ esse tempo de resposta. Seja tee_{q_i} o tempo de execução estimado da consulta q_i (tempo decorrido desde o início da execução de q_i até o seu término) e, tc_{q_i} o tempo de chegada de q_i no serviço de dados. Deste modo, o tempo de resposta estimado da consulta q_i em função do tempo estimado tie para início de sua execução é calculado pela fórmula a seguir:

$$tre_{q_i}(tie) = (tie + tee_{q_i}) - tc_{q_i} \tag{5.7}$$

Um conceito adicional, derivado das definições de (SOUSA et al., 2012), utilizado na solução proposta é o de saldo, o qual consiste na receita subtraída das penalidades. O saldo é definido em dois níveis: por consulta (Equação 5.8) e por carga de trabalho (Equação 5.9). Assim, considere que s_{q_i} representa o saldo referente à consulta q_i e S_W, o saldo da carga de trabalho W. Suponha que r_{q_i} e p_{q_i} são a receita e a penalidade associadas à q_i, respectivamente, e, R_W e P_W são a receita e a penalidade associadas à W, respectivamente. Então, temos que:

$$s_{q_i} = r_{q_i} - p_{q_i} \tag{5.8}$$

e

$$S_W = R_W - P_W \tag{5.9}$$

Baseado no conceito de saldo, será utilizado também o saldo estimado, que consiste na receita r_{q_i} subtraída da penalidade estimada $pe_{q_i}(tre_{q_i})$ (Fórmula 5.6). O saldo estimado $se_{q_i}(tie)$ da execução de q_i, se q_i iniciar sua execução no instante de tempo tie, pode

ser calculado pela Fórmula 5.10. Esta fórmula considera a receita r_{q_i} e a penalidade estimada $pe_{q_i}(tre_{q_i}(tie))$ (Fórmula 5.6). Neste caso, tie é o tempo inicial estimado de execução de q_i e $tre_{q_i}(tie)$ (Fórmula 5.7), o tempo de resposta estimado, caso q_i inicie sua execução no instante de tempo tie.

$$se_{q_i}(tie) = r_{q_i} - pe_{q_i}(tre_{q_i}(tie)) \tag{5.10}$$

Para que o lucro do provedor do serviço de dados em nuvem seja calculado, algumas informações referentes ao SLA são armazenadas na metabase, por tipo de consulta. A seguir, descreveremos essas informações:

- k, o identificador do tipo de consulta Q_k;
- texto SQL de Q_k;
- SLO_{Q_k}, trata-se do tempo de resposta t_k máximo para quaisquer instâncias de consulta do tipo Q_k;
- r_{Q_k}, representa o valor monetário acordado entre o cliente e o provedor do serviço de dados para executar uma instância de consulta q_i do tipo Q_k.

Quando uma instância q_i do tipo Q_k é recebida pelo módulo fachada, os valores previamente definidos para r_{Q_k} e SLO_{Q_k} são associados à consulta q_i (ou seja, $r_{q_i} = r_{Q_k}$ e $SLO_{q_i} = SLO_{Q_k}$). Após a execução de q_i, o tempo de resposta tr_{q_i} de q_i é obtido. Todas essas informações (r_{q_i}, SLO_{q_i} e tr_{q_i}) são armazenadas na metabase. Assim, as fórmulas 5.5 e 5.8 podem ser aplicadas para calcular a penalidade e o saldo de q_i, respectivamente.

5.2 Despacho

O problema de despacho consiste em alocar (atribuir) cada consulta q_i, pertencente a uma determinada carga de trabalho W, a uma das instâncias do módulo escalonador disponíveis, de forma a proporcionar o maior lucro possível ao provedor do serviço de dados. A Figura 5.2 ilustra os módulos envolvidos na solução do problema de despacho. Inicialmente, a aplicação cliente envia ao módulo fachada uma requisição solicitando a execução de uma determinada consulta q_i. O módulo fachada redireciona a requisição recebida para o módulo despachante. Em seguida, o módulo despachante envia uma requisição, contendo a consulta q_i, para cada instância me_k do módulo escalonador, solicitando uma estimativa (a ser armazenada na variável e) do benefício que a execução da consulta q_i na instância me_k traria para o serviço de dados. Esse benefício é mensurado por meio da utilização de duas métricas: ganho em interação (Definição 4.3.1) e ganho em saldo (Definição 5.2.1). Essas duas métricas são calculadas por um procedimento denominado escalonamento hipotético. Após todas as instâncias do módulo escalonador enviarem essas estimativas ao módulo despachante, este seleciona a instância do módulo escalonador que proporcionar o maior ganho em saldo. Seja gs_{max} o maior valor

retornado para o ganho em saldo. Caso duas ou mais instâncias retornem o mesmo valor gs_{max} para o ganho em saldo, o despachante irá selecionar aquela com maior valor para a estimativa do ganho em interação. Por fim, o módulo despachante irá enviar a consulta q_i para a instância do módulo escalonador selecionada.

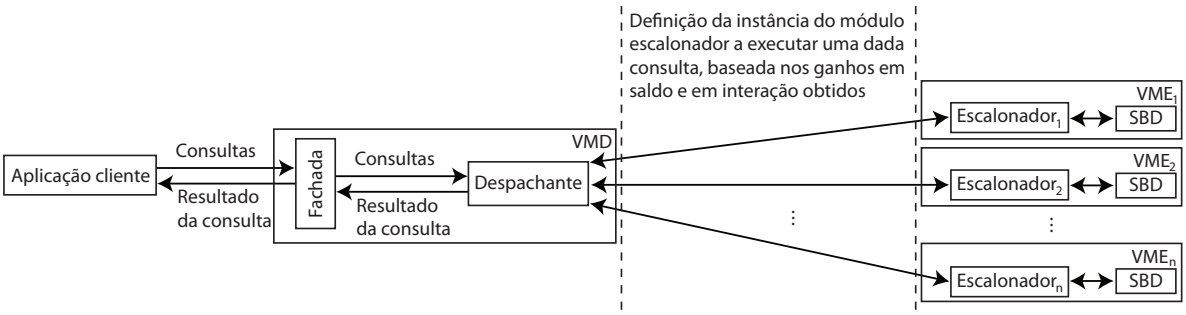


Figura 5.2: Arquitetura da solução proposta para o problema de despacho

Definição 5.2.1 (Ganho em saldo) *Seja q_i uma consulta, me_k uma instância do módulo escalonador, F o conjunto das consultas na fila de escalonamento de me_k, SE_F a estimativa do saldo proporcionado pela execução das consultas pertencentes à fila F e $SE_{F\cup\{q_i\}}$ a estimativa do saldo proporcionado pela execução de q_i em me_k, o ganho em saldo (valor monetário) de se executar a consulta q_i na instância me_k, representado por $gs(q_i,F)$, é calculado de acordo com a Fórmula 5.11.*

$$gs(q_i,F) = SE_{F\cup\{q_i\}} - SE_F \tag{5.11}$$

A fila de escalonamento mencionada na definição de ganho em saldo (Definição 5.2.1) possui o mesmo funcionamento daquela descrita na Seção 4.3 e ilustrada na Figura 4.1. Seja F a fila de escalonamento e, n a quantidade de consultas em F. Deste modo, a posição 0 de F (primeira posição) é reservada para a última consulta submetida à execução. A posição 1 (segunda posição), por sua vez, armazena a próxima consulta a ser executada, a posição $n-1$ se refere à última consulta em F a ser executada e a posição n consiste na primeira posição livre em F. Por exemplo, suponha $n = 10$. Assim, a fila F tem 9 consultas ($n-1$ consultas) para executar e a primeira posição livre em F é a posição 10 (posição n). Além disso, novas consultas podem ser inseridas entre as posições 1 e n, inclusive. Ademais, sempre que uma consulta q_i for inserida em uma determinada posição da fila F, por conseguinte, cada consulta localizada após q_i, na fila F, terá sua posição em F incrementada em uma unidade. No decorrer do trabalho, assumiremos que $|F| = n$, ou seja, que terão $|F|$ consultas na fila F.

Seja me_k uma instância do módulo escalonador, F o conjunto das consultas na fila de escalonamento de me_k e, $q_j \in F$ uma consulta inserida na fila F, tal que $1 \leq j \leq |F|-1$. A definição de ganho em saldo (Definição 5.2.1) utiliza a estimativa SE_F do saldo proporcionado pela execução das consultas pertencentes à fila F. Esta estimativa é calculada pela Fórmula 5.12, que consiste no somatório do saldo estimado (Fórmula 5.10) de cada consulta q_j, considerando o tempo inicial estimado tie_{q_j} pré-definido de q_j. Outra estimativa utilizada na definição do

ganho em saldo (Definição 5.2.1), representada por $SE_{F\cup\{q_i\}}$, refere-se ao saldo proporcionado pela execução de q_i em me_k (Fórmula 5.13). Esta estimativa consiste no saldo estimado máximo obtido pela inserção de q_i em cada uma das posições possíveis da fila F (entre as posições 1 e $|F|$, inclusive). Assim, a estimativa $SE_{F\cup\{q_i\}}(j)$ para o cálculo do saldo estimado de se inserir q_i na posição j da fila F é realizado pela Fórmula 5.14 e consiste na soma dos seguintes termos:

1. somatório do saldo estimado de cada consulta q_l, tal que $1 \leq l \leq j-1$, considerando o instante de tempo inicial estimado tie_{q_l} pré-definido de q_l;

2. saldo estimado da consulta q_i, considerando o instante de tempo inicial estimado tie_{q_j} pré-definido de q_j. Este saldo é calculado baseado no tempo tie_{q_j} porque a estimativa de saldo $SE_{F\cup\{q_i\}}(j)$ se refere à inserção de q_i na posição j da fila F. Logo, o tempo inicial estimado de q_i seria o mesmo tempo inicial estimado pré-definido para q_j;

3. somatório do saldo estimado de cada consulta q_l, tal que $j \leq l \leq |F|-1$, considerando que cada consulta q_l seria adiada em função do tempo de execução estimado tee_{q_i} de q_i, ou seja, o instante de tempo inicial estimado tie_{q_l} de q_l seria adiado em função do tempo tee_{q_i}.

$$SE_F = \sum_{j=1}^{|F|-1} se_{q_j}(tie_{q_j}) \tag{5.12}$$

$$SE_{F\cup\{q_i\}} = max\{SE_{F\cup\{q_i\}}(j) | 1 \leq j \leq |F|\} \tag{5.13}$$

e

$$SE_{F\cup\{q_i\}}(j) = \sum_{l=1}^{j-1} se_{q_l}(tie_{q_l}) + se_{q_i}(tie_{q_j}) + \sum_{l=j}^{|F|-1} se_{q_l}(tie_{q_l} + tee_{q_i}) \tag{5.14}$$

Para uma melhor compreensão da solução de despacho utilizada pelo GeDaNIC, o diagrama de sequência na Figura 5.3 ilustra o funcionamento do processo de despacho no *framework* proposto. A seguir, descreveremos em detalhes todos os passos e algoritmos envolvidos na solução de despacho proposta.

O processo de despacho tem início quando a aplicação cliente requisita ao módulo fachada uma requisição solicitando a execução de uma determinada consulta q_i. Ao receber essa requisição, o módulo fachada executa o Algoritmo 6 (algoritmo principal do módulo fachada), o qual recebe como entrada a consulta q_i e utiliza como variável global a referência *md* ao módulo despachante. Então, o módulo fachada cria uma estrutura de dados q_i' para representar a consulta q_i (linhas 2 a 6 do Algoritmo 6). Esta estrutura de dados, cujo tipo foi denominado EDConsultaSLA (Figura 5.4), contém a própria consulta q_i, além de algumas informações adicionais: o instante de tempo em que a consulta q_i foi recebida pelo módulo fachada, ou seja, o tempo de chegada de q_i no serviço de dados (linha 2), o identificador de q_i (linha 3), a receita e o

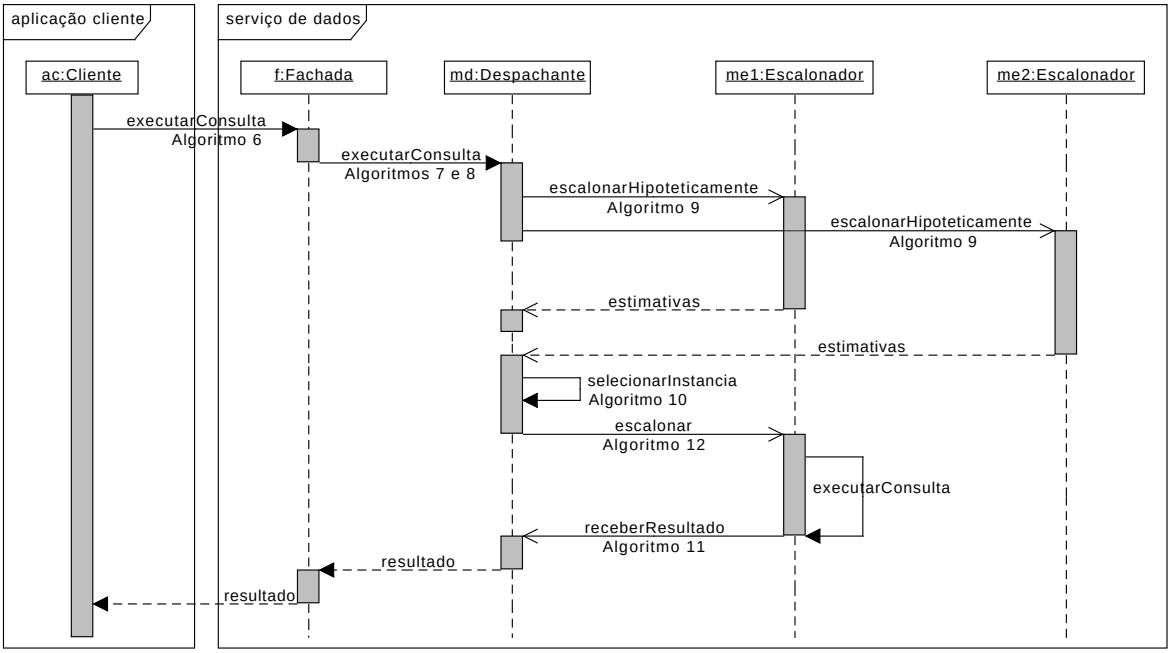


Figura 5.3: Diagrama de sequência para o despacho de uma consulta

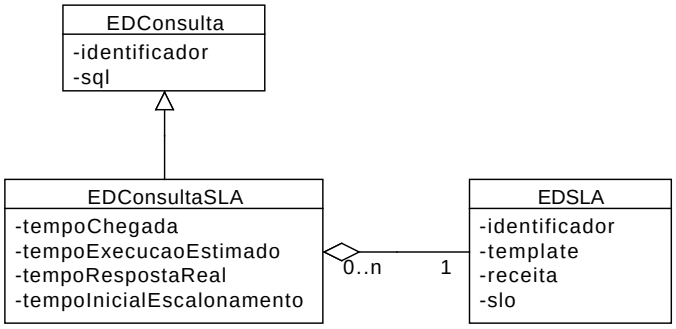


Figura 5.4: Estrutura de dados representativa da consulta acoplada ao SLA

SLO (tempo de resposta máximo) de q_i (linha 4) acordados entre o cliente e o provedor do serviço de dados, e o tempo de execução estimado de q_i, o qual deve ser obtido previamente (linha 5). Nos experimentos realizados, foi definido um SLO para cada tipo (template) de consulta Q_k. Assim, todas as instâncias de consulta de um mesmo tipo possuem o mesmo SLO. Já o tempo de execução estimado de uma instância de consulta q_i foi obtido executando-se q_i com o cache vazio (memória livre). Posteriormente, o módulo fachada redireciona a requisição recebida, agora contendo a estrutura de dados que representa a consulta q_i (ou seja, q_i'), para o módulo despachante (linha 7 do Algoritmo 6, chamada ao Algoritmo 7 do módulo despachante). Posteriormente, o tempo de resposta $tr_{q_i'}$ é calculado (linha 8) e q_i' é enviada ao módulo monitor para que o mesmo atualize as estatísticas referentes à q_i' (linha 10). A execução do Algoritmo 7 irá retornar o resultado da execução da consulta q_i', o qual será armazenado na variável *resultado*. Por fim, o Algoritmo 6 retorna a variável *resultado* para a aplicação cliente (linha 11).

Algoritmo 6: Algoritmo principal do módulo fachada

Entrada: instância de consulta q_i
Saída: resultado da consulta
Dados: referência *md* ao módulo despachante

```
1  início
2      tc_qi ← obterTempoCorrente();
3      id_qi ← gerarIdConsulta();
4      sla_qi ← obterSLA(q_i);
5      tee_qi ← obterTempoEstimado(q_i);
6      q'_i ← criarEDConsultaSLA(id_qi, q_i, sla_qi, tc_qi, tee_qi);
7      resultado ← md.Algoritmo7(q'_i);
8      tr_qi ← obterTempoCorrente() − tc_qi;
9      q'_i.atribuirTempoResposta(tr_qi);
10     enviarEstatisticasMonitor(q'_i);
11     retorna resultado;
12 fim
```

O módulo despachante, ao receber uma requisição do módulo fachada solicitando a execução da consulta q_i, executa o Algoritmo 7 (algoritmo de recebimento de solicitações de execução de consulta pelo módulo despachante), o qual recebe como entrada a estrutura de dados q'_i, a qual representa a consulta q_i. Além disso, a variável global D, que representa a fila de despacho, é utilizada pelo Algoritmo 7. A fila de despacho é utilizada para armazenar as consultas a serem despachadas pelo módulo despachante. Inicialmente, o Algoritmo 7 insere a consulta q'_i na fila de despacho D (linha 2) e o despachante notifica a si mesmo informando que uma nova consulta foi adicionada na fila de despacho (linha 3). O Algoritmo 8 (algoritmo principal para o despacho de consultas) é responsável por receber essa notificação e despachar a consulta q'_i. Enquanto q'_i não concluir sua execução, o módulo despachante aguarda por uma notificação, informando que o resultado gerado pela execução de q'_i esteja disponível (entre as linhas 4 e 6). Por fim, o resultado obtido pela execução de q'_i é retornado para o módulo fachada (linha 7).

Algoritmo 7: Algoritmo de recebimento de solicitação de consulta pelo módulo despachante

Entrada: estrutura de dados q'_i
Saída: resultado da consulta
Dados: fila de despacho D

```
1  início
2      D.inserir(q'_i);
3      notificarDespachante(q'_i);
4      enquanto não possuiResultado(q'_i.obterId()) faça
5          esperarNotificacaoResultado();
6      fim enqto
7      retorna obterResultado(q'_i.obterId());
8  fim
```

Algoritmo 8: Algoritmo principal para o despacho de consultas

Dados: lista V das instâncias do módulo escalonador disponíveis, referência mpr ao módulo provedor de recursos

```
1  início
2      enquanto verdadeiro faça
3          q'_i ← obterProximaConsulta();
4          para cada me_k ∈ V faça
5              e ← me_k.Algoritmo9(q'_i);
6              me_k.atualizarEstimativas(e);
7          fim para cada
8          me_k ← Algoritmo10(q'_i);
9          me_k.Algoritmo12(q'_i);
10         mpr.Algoritmo15(q'_i, me_k);
11     fim enqto
12 fim
```

Ao ser notificado de que uma nova consulta foi inserida na fila de despacho, o despachante executa o Algoritmo 8 (algoritmo principal para o despacho de consultas), o qual utiliza como variáveis globais: a lista das instâncias do módulo escalonador disponíveis, representada por V, e a referência mpr ao módulo provedor de recursos. Este algoritmo executa indefinidamente, por meio do laço de repetição entre as linhas 2 e 11. Dentro desse laço, o primeiro passo consiste na obtenção da próxima consulta a ser despachada (linha 3), a qual é armazenada na variável q'_i. Em seguida, o laço entre as linhas 4 e 7 é executado. Neste laço é executado o escalonamento hipotético de q'_i em cada instância $me_k \in V$ do módulo escalonador (linha 5, chamada ao Algoritmo 9). Ou seja, o módulo despachante envia uma requisição, contendo a consulta q'_i, para cada instância $me_k \in V$ do módulo escalonador, solicitando uma estimativa do benefício que a execução da consulta q'_i na instância me_k traria para o serviço de dados (linha 5). Como mencionado anteriormente, esse benefício é mensurado por meio da utilização de duas métricas: ganho em interação (Definição 4.3.1) e ganho em saldo (Definição 5.2.1). Além dessas duas métricas, a variável e vai armazenar o tempo de execução estimado associado a uma dada instância me_k do módulo escalonador, que consiste no instante de tempo estimado para a conclusão da execução da última consulta na fila de escalonamento de me_k executar subtraído do instante de tempo corrente. Esta informação é utilizada para identificar se a instância me_k está sobrecarregada em comparação com as outras instâncias do módulo escalonador. As estimativas de ganho em saldo, ganho em interação e o tempo de execução estimado supracitados serão utilizados pelo Algoritmo 10 para selecionar a instância do módulo escalonador para a qual será despachada a consulta q'_i. Após uma instância me_k responder à solicitação enviada pelo despachante, as estimativas de ganhos em saldo e interação que seriam proporcionados ao serviço de dados caso a consulta q'_i fosse despachada para me_k, bem como o tempo de execução estimado associado à me_k, são atualizados (linha 6). Na linha 8, é selecionada a instância me_k do módulo escalonador com maior ganho em interação, dentre aquelas com maior ganho em saldo (chamada ao Algoritmo 10). Esta instância será responsável pela execução de q'_i. Em seguida, o módulo despachante irá enviar a consulta q'_i para a instância do módulo escalonador selecionada (linha 9, chamada ao Algoritmo 12, algoritmo principal do módulo escalonador).

Por fim, na linha 10 é executada uma chamada ao Algoritmo 15 com a finalidade de verificar a necessidade de se criar novas instâncias do módulo escalonador.

O Algoritmo 9 (algoritmo para escalonamento hipotético realizado por uma instância do módulo escalonador) descreve o escalonamento hipotético de uma consulta q'_i em uma determinada instância do módulo escalonador me_k. Assim, esse algoritmo executa nas instâncias do módulo escalonador. O Algoritmo 9, possui como entrada a consulta q'_i. Já a saída desse algoritmo, representada pela variável e, é composta pelo tempo de execução estimado para que todas as consultas na fila de escalonamento de me_k sejam executadas, bem como por uma estimativa do benefício que a execução da consulta q'_i na instância me_k traria para o serviço de dados (mais precisamente, esse benefício é mensurado por meio das estimativas para o ganho em saldo e para o ganho em interação). Essa variável consiste em uma estrutura de dados cujo tipo, denominado EDEstimativas, é constituído por: i) id_{me_k}, identificador da instância do módulo escalonador; ii) ganho em saldo proporcionado por q'_i; iii) ganho em interação trazido por q'_i; e, iv) $t_{execucao}$, tempo de execução estimado para que todas as consultas na fila de escalonamento de me_k sejam executadas. Além disso, o Algoritmo 9 utiliza a variável global F, a qual representa a fila de escalonamento de me_k, e o identificador id_{me_k} de me_k. Inicialmente, é executada uma chamada ao Algoritmo 13 (linha 2). O Algoritmo 13 tem por finalidade estimar o ganho em saldo e o ganho em interação que seria proporcionado pela execução da consulta q'_i na instância me_k e será detalhado na Seção 5.3. Uma estrutura de dados contendo essas estimativas é armazenada na variável g. Em seguida, o tempo estimado para a execução de todas as consultas em F é armazenado na variável $t_{execucao}$ (linha 3). Esta informação é usada para auxiliar na escolha da instância do módulo escalonador responsável pela execução de q'_i, no Algoritmo 10. Por fim, uma estrutura de dados do tipo EDEstimativas, contendo as estimativas geradas pelo escalonamento hipotético, é criada (linha 4) e retornada (linha 5).

Algoritmo 9: Algoritmo para escalonamento hipotético realizado por uma instância do módulo escalonador

Entrada: estrutura de dados q'_i
Saída: estrutura de dados e
Dados: fila de escalonamento F, identificador id_{me_k} da instância me_k

```
1 início
2     g ← Algoritmo13(q'_i);
3     t_execucao ← q_{|F|-1}.obterTempoFinal() - obterTempoCorrente();
4     e ← novasEstimativas(id_{me_k}, g, t_execucao);
5     retorna e;
6 fim
```

O algoritmo de seleção da instância do módulo escalonador, chamado pelo Algoritmo 8, é descrito pelo Algoritmo 10 (algoritmo para selecionar a instância do módulo escalonador para executar uma determinada consulta q'_i), que tem como dado de entrada a consulta q'_i a ser despachada. Já o dado de saída do Algoritmo 10 é a instância do módulo escalonador $me_{selecionada}$ escolhida para receber a consulta q'_i. Além disso, as seguintes variáveis globais são utilizadas por este algoritmo: a lista V de instâncias do módulo escalonador e o fator de interação global f_{global} (Definição 5.2.2). Neste trabalho, não definimos como calcular o fator

de interação global. Assim, fixamos nos experimentos realizados o fator de interação global com o valor de 0,5, ou seja, $f_{global} = 0,5$.

Definição 5.2.2 (Fator de interação global) *Um número entre 0 e 1 que quantifica o nível de interação entre consultas, sob um aspecto global, ou seja, que represente a interação entre consultas no serviço de dados como um todo. Valores próximos a 1 indicam forte interação e valores próximos a 0, fraca interação.*

Algoritmo 10: Algoritmo para selecionar a instância do módulo escalonador para executar uma determinada consulta q'_i

Entrada: estrutura de dados q'_i
Saída: instância $me_{selecionada}$ do módulo escalonador
Dados: lista V das instâncias do módulo escalonador disponíveis, fator de interação global f_{global}

1 **início**
2 $\quad gs_{max} \leftarrow max\{gsme_k \mid me_k \in V\}$;
3 $\quad V_{saldo} \leftarrow \{me_k \in V \mid gsme_k = gs_{max}\}$;
4 $\quad tee_{min} \leftarrow min\{teeme_k \mid me_k \in V_{saldo}\}$;
5 $\quad tee_{max} \leftarrow (1 + f_{global}) \times tee_{min}$;
6 $\quad gi_{max} \leftarrow max\{gime_k \mid me_k \in V_{saldo}$ e $teeme_k \leq tee_{max}\}$;
7 $\quad V_{interacao} \leftarrow \{me_k \in V_{saldo} \mid gime_k = gi_{max}$ e $teeme_k \leq tee_{max}\}$;
8 $\quad me_{selecionada} \leftarrow$ `obterElemento(`$V_{interacao}$`);`
9 $\quad$ **retorna** $me_{selecionada}$;
10 **fim**

Seja me_k uma instância do módulo escalonador, F a fila de escalonamento de me_k, gs_{me_k} o ganho em saldo de se executar a consulta q'_i em me_k, gi_{me_k} o ganho em interação de se executar a consulta q'_i em me_k e, tee_{me_k} o intervalo tempo estimado em que todas as consultas em F executarão. Além disso, gs_{me_k}, gi_{me_k} e tee_{me_k} compõem as estimativas de me_k obtidas no algoritmo principal de despacho (Algoritmo 8). Deste modo, inicialmente, o Algoritmo 10 obtém o conjunto V_{saldo} de instâncias do módulo escalonador com maior ganho em saldo, considerando todas as instâncias do módulo escalonador do conjunto V de instâncias disponíveis (linhas 2 e 3). Em seguida, é obtido o intervalo de tempo mínimo estimado tee_{min}, considerando o tempo tee_{me_k} de cada $me_k \in V_{saldo}$ (linha 4). O tempo tee_{minimo} é utilizado no cálculo do limite superior de tempo de execução (tee_{max}) aceitável para uma dada instância $me_k \in V_{saldo}$ (linha 5), o qual também é calculado em função do fator de interação global f_{global}. Posteriormente, é calculado o ganho em interação máximo gi_{max} entre as instâncias $me_k \in V_{saldo}$, dentre aquelas que possuem o tempo de execução tee_{me_k} menor ou igual ao tempo limite tee_{max} (linha 6). Assim, o conjunto $V_{interacao}$ de instâncias $me_k \in V_{saldo}$ do módulo escalonador com ganho em interação igual a gi_{max} é definido, respeitando-se a restrição de que cada instância me_k possua o tempo de execução tee_{me_k} menor ou igual ao tempo limite tee_{max} (linha 7). A restrição $tee_{m_k} \leq tee_{max}$ (linhas 6 e 7) foi utilizada a fim de evitar sobrecarga de consultas em uma determinada instância me_k. Por fim, a instância $me_{selecionada}$ do módulo escalonador retornada (linha 9) será qualquer uma pertencente ao conjunto $V_{interacao}$ (linha 8).

Suponha que a consulta q'_i tenha sido despachada para a instância do módulo escalonador me_k. Neste caso, no momento em que q'_i concluir sua execução, a instância do módulo escalonador me_k envia o resultado da execução de q'_i para o módulo despachante. Este resultado é enviado através da chamada ao Algoritmo 11 executado no módulo despachante. Este algoritmo possui como parâmetros de entrada a consulta q'_i e a variável *resultado*, que representa o resultado obtido pela execução da consulta q'_i. Ademais, o Algoritmo 11 utiliza as variáveis globais V (lista de instâncias do módulo escalonador), R (mapeamento chave-valor, onde a chave é o identificador da consulta e o valor, o resultado obtido por sua execução) e *mpr* (referência ao módulo provedor de recursos). No Algoritmo 11, o primeiro passo consiste em armazenar o resultado da execução da consulta q'_i no mapeamento R (linha 2). Este mapeamento é utilizado no Algoritmo 7 para enviar o resultado da execução de q'_i do módulo despachante para o módulo fachada. Em seguida, na linha 3, o módulo despachante notifica a si próprio para que possa dar prosseguimento à execução dos passos do Algoritmo 7 (que envia os resultados do módulo despachante para o módulo fachada). Por fim, o módulo provedor de recursos *mpr*, na linha 4, executa o Algoritmo 16 para verificar a necessidade de suspensão de VMEs.

Algoritmo 11: Algoritmo para recebimento do resultado da execução de uma consulta q'_i executada em uma instância me_k do módulo escalonador pelo módulo despachante

Entrada: estrutura de dados q'_i, variável resultado
Dados: lista V das instâncias do módulo escalonador disponíveis, mapeamento chave-valor R, referência *mpr* ao módulo provedor de recursos

```
1 início
2 |  R.adicionarResultado(q'_i.obterId(), resultado);
3 |  notificarResultado();
4 |  mpr.Algoritmo16(V);
5 fim
```

5.3 Escalonamento de Consultas

Seja me_k uma instância do módulo escalonador, F a fila de escalonamento de me_k e, q'_i uma instância de consulta despachada à me_k, que deve ser inserida na fila F. O problema do escalonamento de consultas consiste na tomada de decisão sobre a ordem em que as consultas pertencentes a uma determinada fila de escalonamento F devem ser executas a fim de maximizar o lucro do provedor do serviço de dados. Neste trabalho, esta ordem é definida no momento de inserção de q'_i em F. Assim, o problema pode ser redefinido da seguinte forma: dada uma fila de escalonamento F e uma instância de consulta q'_i, em que posição da fila F a consulta q'_i deve ser inserida a fim de maximizar o lucro do provedor do serviço de dados? Assim, uma instância do módulo escalonador me_k tem como uma de suas funções a definição da ordem de execução das consultas a ela submetidas, ou seja, das consultas armazenadas em sua fila de escalonamento. Esta ordem de execução é sequencial, orientada ao lucro e baseada na interação entre consultas.

A Figura 5.5 ilustra o fluxo de execução do escalonamento de consultas em uma instância do módulo escalonador me_k. Seja me_k uma instância do módulo escalonador, F a

fila de escalonamento de me_k, n a quantidade de consultas na fila F e, q'_i uma instância de consulta despachada à me_k. O fluxo de execução do processo de escalonamento é iniciado com a submissão de q'_i para me_k. O próximo passo consiste em inserir q'_i na fila de escalonamento F, entre as posições 1 e n, inclusive. Quando q'_i localiza-se na posição 1 e me_k é notificada de que pode executar uma nova consulta, então q'_i é deslocada para a posição 0 de F e, posteriormente, executada na réplica do SGBD utilizada por me_k. Terminada a execução de q'_i, seu resultado é enviado à me_k, que o redireciona para o módulo despachante.

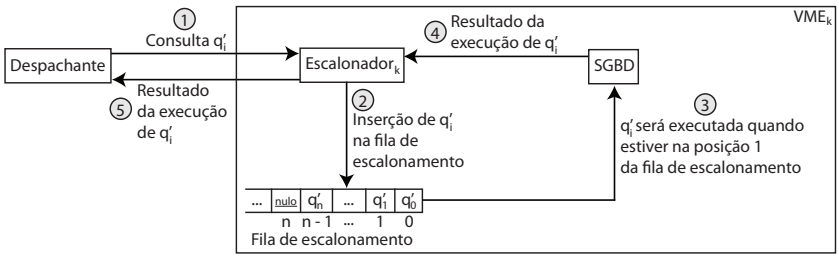


Figura 5.5: Execução de uma dada consulta q'_i em uma instância do módulo escalonador

Quando q'_i é recebida por me_k, sua posição em F é definida por meio do Algoritmo 12 (algoritmo principal do módulo escalonador), o qual possui como dado de entrada q'_i e, como variável global a fila F de escalonamento de me_k. Assim, seja q'_i uma consulta despachada à me_k e, q'_j a consulta na posição j da fila de escalonamento F, tal que $1 \leq j \leq |F|-1$. O Algoritmo 12 inicia com a obtenção da estrutura de dados g do tipo EDGanhos (linha 2), que é composta por: i) estimativa de ganho em saldo; ii) estimativa de ganho em interação e; iii) posição j da fila F selecionada para inserir q'_i, tal que $1 \leq j \leq |F|$. Os valores da estrutura de dados g para a consulta q'_i são definidos no Algoritmo 13. Em seguida, seja j a posição selecionada em F para a consulta q'_i ser inserida (linha 3), o tempo inicial estimado $tie_{q'_i}$ da consulta q'_i é definido com o mesmo valor do tempo final estimado da q'_{j-1} (linha 4). O próximo passo desse algoritmo consiste no adiamento das consultas q'_l, tal que $j \leq l \leq |F|-1$, em função do tempo estimado $tee_{q'_i}$ de execução de q'_i (linha 5). Por fim, q'_i é inserida na posição j de F (linha 6).

Algoritmo 12: Algoritmo principal do módulo escalonador

Entrada: estrutura de dados q'_i
Dados: fila de escalonamento F

```
1 início
2     g ← Algoritmo13(q'_i);
3     j ← g.obterPosicaoSelecionada();
4     q'_i.definirTempoInicial(obterTempoFinal(q'_{j-1}));
5     adiarConsultas(j, tee_{q'_i});
6     F.adicionar(q'_i, j);
7 fim
```

A definição da posição j de F para a consulta q'_i ser inserida, bem como das estimativas de ganho em saldo e em interação citadas no Algoritmo 12 é realizada no Algoritmo 13 (algoritmo para determinar a melhor posição na fila de escalonamento para uma dada consulta

q'_i, bem como as estimativas dos ganhos em saldo e interação proporcionados para o escalonamento de q'_i na posição selecionada). Este algoritmo possui como dado de entrada a consulta q'_i e como variável global a fila de escalonamento F. O dado de saída desse algoritmo é a estrutura de dados g do tipo EDGanhos. Inicialmente, o Algoritmo 13 supõe que q'_i será inserida na posição $|F|$ (primeira posição livre) da fila F. Assim, os ganhos em saldo e em interação serão calculados pelas fórmulas 5.15 e 4.1, respectivamente (linhas 2 e 3). Além disso, a posição j será inicializada com a posição $|F|$ da fila F (linha 4). Seja gs_{max} a estimativa de ganho em saldo de se inserir q'_i na posição de F que gere o maior ganho em saldo. O próximo passo do Algoritmo 13 consiste em percorrer cada posição l, tal que $1 \leq l \leq |F|-1$, e identificar qual posição gera o maior ganho em interação, dentre aquelas que possuem o ganho em saldo igual ao valor gs_{max} (linhas 5 a 14). Caso mais de uma posição seja encontrada, pode-se escolher qualquer uma delas para ser a posição j, armazenada em g. Por fim, a estrutura de dados g é retornada (linha 15).

O Algoritmo 13 calcula o ganho em saldo através da Fórmula 5.15, que se utiliza da estimativa SE_F do saldo proporcionado pela execução das consultas pertencentes à fila F (Fórmula 5.12) e, da estimativa $SE_{F\cup\{q'_i\}}(j)$ do saldo proporcionado pela execução das consultas pertencentes à fila F, quando q_i for inserida na posição j da fila F (Fórmula 5.14).

$$gs(q'_i, F, j) = SE_F - SE_{F\cup\{q'_i\}}(j) \tag{5.15}$$

Algoritmo 13: Algoritmo para determinar a melhor posição na fila de escalonamento para uma dada consulta q'_i, bem como as estimativas dos ganhos em saldo e interação proporcionados para o escalonamento de q'_i na posição selecionada

Entrada: estrutura de dados q'_i
Saída: estrutura de dados g
Dados: fila de escalonamento F

```
1  início
2      gs_max ← gs(q'_i, |F|);
3      gi_max ← gi(q'_i, q'_{|F|-1}, nulo);
4      j ← |F|;
5      para l ← |F| − 1 até 1 faça
6          se gs_max < gs(q'_i, F, l) então
7              gs_max ← gs(q'_i, F, l);
8              gi_max ← gi(q'_i, q'_{l-1}, q'_l);
9              j ← l;
10         senão se gs_max = gs(q'_i, F, l) e gi_max < gi(q'_i, q'_{l-1}, q'_l) então
11             gi_max ← gi(q'_i, q'_{l-1}, q'_l);
12             j ← l;
13         fim se
14     fim para
15     retorna novoGanho(gs_max, gi_max, j);
16 fim
```

Seja me_k uma instância do módulo escalonador, q'_i a próxima consulta a ser executada em me_k e, F o conjunto das consultas pertencentes à fila de escalonamento de me_k. A execução de q'_i por me_k é realizada pelo Algoritmo 14 (algoritmo para executar as consultas na fila de escalonamento). Este algoritmo utiliza a variável global *md*, que representa a referência ao módulo despachante. O Algoritmo 14 executa indefinidamente por meio do laço de repetição entre as linhas 2 e 6, inclusive. Inicialmente, neste laço, a consulta q'_i (próxima a executar) é obtida, uma vez que está localizada na posição 1 da fila F (linha 3). Em seguida, q'_i é executada e seu resultado armazenado na variável *resultado* (linha 4). Por fim, o Algoritmo 11 (algoritmo para recebimento do resultado da execução de uma consulta q'_i executada em uma instância me_k do módulo escalonador pelo módulo despachante) é chamado (linha 5).

Algoritmo 14: Algoritmo para executar as consultas na fila de escalonamento

```
   Dados: referência md ao módulo despachante
1  início
2      enquanto verdadeiro faça
3          q'_i ← obterProximaConsulta();
4          resultado ← executarConsulta(q'_i);
5          md.Algoritmo11(q'_i.obterId(), resultado);
6      fim enqto
7  fim
```

5.4 Planejamento de Capacidade

Em um serviço de gerenciamento de dados em nuvem, um dos motivos para o não cumprimento das especificações do SLA (ou seja, dos SLOs das consultas) pode ser a quantidade insuficiente de instâncias do módulo escalonador. O não cumprimento dos SLAs implica em penalidades e, consequentemente, reduz o lucro do provedor do serviço. Por outro lado, manter uma quantidade de instâncias do módulo escalonador acima da quantidade necessária ocasiona pagamentos desnecessários ao provedor de IaaS, o que reduz o lucro do provedor. Neste contexto, o problema de planejamento de capacidade diz respeito à decisão sobre a quantidade de instâncias do módulo escalonador utilizadas pelo serviço de dados em nuvem. Portanto, uma solução que gerencie automaticamente, e de forma eficiente, a quantidade de instâncias do módulo escalonador que compõem o serviço de dados em nuvem se torna imprescindível.

A Figura 5.6 ilustra a arquitetura utilizada pelo GeDaNIC para tratar o problema de planejamento de capacidade. Nesta arquitetura, o módulo despachante envia estimativas sobre a execução das consultas para o módulo provedor de recursos, de forma a auxiliar na decisão sobre a instanciação/suspensão de VMEs. O módulo provedor de recursos solicita ao provedor de IaaS que instâncias do módulo escalonador sejam criadas/suspensas, quando verificar que o lucro poderia ser incrementado. Por sua vez, o provedor de IaaS notifica o módulo provedor de recursos quando uma dada VME vme_k foi instanciada/suspensa. Em seguida, o módulo provedor de recursos notifica o módulo despachante que a instância do módulo escalonador associada à vme_k precisa ser registrada na (ou removida da) lista de instâncias do módulo escalonador, a qual é armazenada pelo módulo despachante.

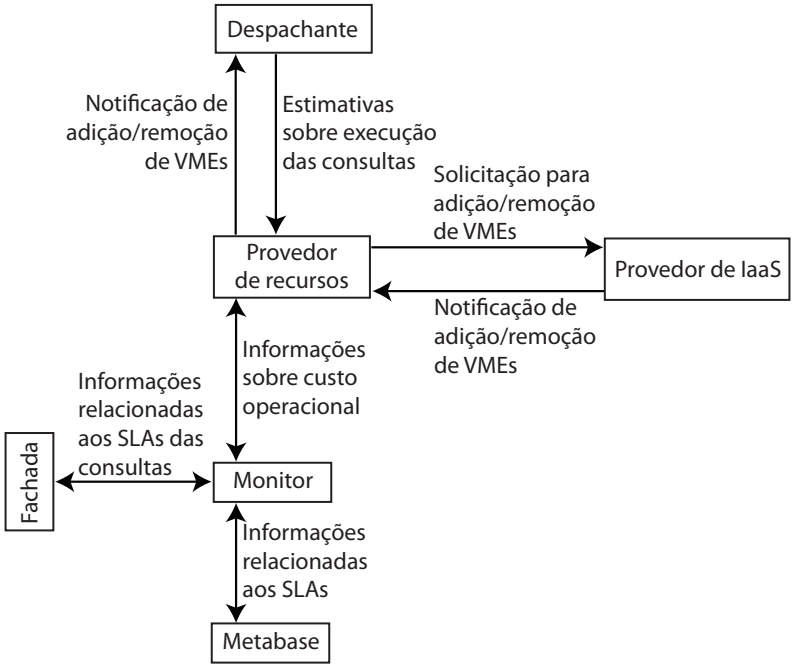


Figura 5.6: Arquitetura da solução para o planejamento de capacidade

Além disso, o provedor de IaaS envia o valor monetário gasto com as VMEs por unidade de tempo faturável para o módulo provedor de recursos. Este redireciona essa informação para o módulo monitor, o qual armazena o custo operacional na metabase. Adicionalmente, o módulo fachada envia ao módulo monitor as consultas com seus respectivos tempos de resposta, a fim de armazenar as estatísticas sobre o cumprimento dos SLAs. Já o módulo monitor, considerando q'_i uma instância de consulta do tipo Q_k, informa ao módulo fachada a receita e o SLO associados à instância q'_i, obtidos da metabase.

Seja f_{global} o fator de interação global (Definição 5.2.2), q'_i uma instância de consulta, me_k uma instância do módulo escalonador, F a fila de escalonamento de me_k e, j a posição em que q'_i seria inserida na fila F, tal que $1 \leq j \leq |F| - 1$, caso fosse despachada para me_k.

O Algoritmo 15 (chamado pelo Algoritmo 8, algoritmo principal para o despacho de consultas) é responsável por verificar a necessidade de se instanciar uma nova VME. O Algoritmo 15 tem como dados de entrada uma dada consulta q'_i e a instância me_k do módulo escalonador para a qual q'_i foi despachada.

Inicialmente, o Algoritmo 15 (linha 2) calcula o ganho em saldo que seria proporcionado pela inserção da consulta q'_i na fila F, representado por $gsf(q'_i, F)$. O cálculo do ganho em saldo baseia-se na utilização do fator de interação global, denominado f_{global}. Assim, o ganho em saldo é calculado por meio da Fórmula 5.16. Nesta fórmula, SEF_F representa o saldo estimado proporcionado pelas consultas pertencentes à fila F, enquanto $SEF_{F \cup \{q'_i\}}$ representa o saldo estimado proporcionado pela fila F caso a consulta q'_i fosse inserida em F.

$$gsf(q'_i, F) = SEF_{F \cup \{q'_i\}} - SEF_F \tag{5.16}$$

Algoritmo 15: Algoritmo para avaliar a instanciação de uma nova VME

Entrada: estrutura de dados q'_i, referência me_k a uma dada instância do módulo escalonador

Dados: referência me' a uma instância hipotética do módulo escalonador, fila de escalonamento F' de me', fator de interação global f_{global}, acúmulo de saldo estimado Δ_{saldo_+}, custo operacional c por unidade faturável

```
1  início
2      gsf_{me_k} ← gsf(q'_i, F);
3      gsf_{me'} ← gsf(q'_i, F');
4      se gsf_{me'} > gsf_{me_k} então
5          Δ_{saldo_+} ← Δ_{saldo_+} + gsf_{me'} − gsf_{me_k};
6          me'.escalonar(q'_i);
7          se Δ_{saldo_+} > c então
8              solicitarCriacaoVME();
9              F' ← ∅;
10             Δ_{saldo_+} ← 0;
11         fim se
12     fim se
13 fim
```

O valor da estimativa SEF_F é calculado pela Fórmula 5.17, na qual $tie_{q'_j}$ representa a estimativa para o tempo de início da execução da consulta que está na posição j da fila F. Assim, $sef_{q'_j}$ indica o saldo estimado que seria proporcionado pela execução da consulta q'_j, a qual ocupa a posição j da fila F, caso essa inicie sua execução no tempo $tie_{q'_j}$. Logo, SEF_F recebe o somatório dos saldos proporcionados pelas consultas atualmente inseridas na fila F.

$$SEF_F = \sum_{j=1}^{|F|-1} sef_{q'_j}(tie_{q'_j}) \tag{5.17}$$

O valor de $sef_{q'_j}$ é calculado por meio da Fórmula 5.18, em que $r_{q'_i}$ representa a receita de q'_i. Já $trf_{q'_i}(tie)$ representa o tempo de resposta estimado para a consulta q'_i caso esta inicie sua execução no instante de tempo tie. O termo $pe_{q'_i}$ indica a penalidade estimada para o provedor do serviço de dados, caso a consulta q'_i apresente um tempo de resposta igual a $trf_{q'_i}(tie)$. Assim, $sef_{q'_i}(tie)$ recebe a receita de q'_i menos a penalidade estimada referente à q'_i caso esta inicie sua execução no tempo tie.

$$sef_{q'_i}(tie) = r_{q'_i} - pe_{q'_i}(trf_{q'_i}(tie)) \tag{5.18}$$

O valor de $trf_{q'_i}(tie)$ é calculado pela Fórmula 5.19 e consiste no tempo de resposta estimado para a consulta q'_i, caso esta inicie sua execução no instante de tempo tie. O termo $trf_{q'_i}(tie)$ é calculado levando em consideração a redução no tempo de resposta de q'_i proporcionada pela interação entre consultas. Essa redução (influência) é estimada por meio da utilização do fator de interação global f_{global}. Na Fórmula 5.19, tie representa o tempo estimado para o início da execução de q'_i, $tee_{q'_i}$ o tempo de execução estimado para q'_i e $tc_{q'_i}$ o tempo de chegada da consulta q'_i no serviço de dados.

$$trf_{q'_i}(tie) = (1 - f_{global}) \times [(tie + tee_{q'_i}) - tc_{q'_i}] \tag{5.19}$$

O valor da estimativa $SEF_{F \cup \{q'_i\}}$, saldo proporcionado pela execução de q'_i em me_k, é calculado por meio da Fórmula 5.20. Para se estimar o valor de $SEF_{F \cup \{q'_i\}}$ é necessário estimar o saldo que seria proporcionado pela execução de q'_i em cada uma das posições da fila F (posições de 1 a $|F|$, inclusive). Em seguida, $SEF_{F \cup \{q'_i\}}$ recebe a maior dessas estimativas. Assim, $SEF_{F \cup \{q'_i\}}(j)$ representa o saldo que seria proporcionado pela execução de q'_i na posição j da fila F (da instância me_k).

$$SEF_{F \cup \{q'_i\}} = max\{SEF_{F \cup \{q'_i\}}(j) | 1 \leq j \leq |F|\} \tag{5.20}$$

A estimativa $SEF_{F \cup \{q'_i\}}(j)$ para o saldo estimado de se inserir q'_i na posição j da fila F é calculada pela Fórmula 5.21, que consiste na soma dos seguintes termos:

1. somatório do saldo estimado de cada consulta q'_l, tal que $1 \leq l \leq j-1$, considerando o instante de tempo inicial estimado $tie_{q'_l}$ pré-definido de q'_l e o fator f_{global};

2. saldo estimado da consulta q'_i, considerando o instante de tempo inicial estimado $tie_{q'_j}$ pré-definido de q'_j e o fator f_{global}. Este saldo é calculado baseado no tempo $tie_{q'_j}$ porque a estimativa de saldo $SEF_{F \cup \{q'_i\}}(j)$ se refere à inserção de q'_i na posição j da fila F. Logo, o tempo inicial estimado de q'_i seria o mesmo tempo inicial estimado pré-definido para a instância de consulta q'_j;

3. somatório do saldo estimado de cada instância de consulta q'_l, tal que $j \leq l \leq |F|-1$, considerando que cada consulta q'_l teria seu instante de tempo inicial estimado $tie_{q'_l}$ adiado em função do tempo de resposta estimado $tee_{q'_i}$ de q'_i. Além disso, o fator de interação global f_{global} é utilizado para representar a redução de tempo de resposta originada pela interação entre consultas.

$$SEF_{F \cup \{q'_i\}}(j) = \sum_{l=1}^{j-1} sef_{q'_l}(tie_{q'_l}) + sef_{q'_i}(tie_{q'_j}) + \sum_{l=j}^{|F|-1} sef_{q'_l}(tie_{q'_l} + tee_{q'_i}) \tag{5.21}$$

Na linha 3, o Algoritmo 15 estima o ganho em saldo que seria proporcionado pela inserção da consulta q'_i na fila F' de uma instância hipotética do módulo escalonador me'. Nas próximas linhas do Algoritmo 15, as variáveis globais me', f_{global}, F', Δ_{saldo_+} e c são utilizadas. A variável me' representa uma instância hipotética do módulo escalonador; f_{global}, o fator de interação global; F', a fila de escalonamento da instância me'; Δ_{saldo_+}, o acúmulo do saldo estimado que o serviço de dados receberia a mais, caso uma nova instância do módulo escalonador fosse criada e; c, o custo operacional por unidade faturável do tipo de VM a ser instanciada.

Em seguida, o Algoritmo 15 verifica se $gsf(q'_i, F') > gsf(q'_i, F)$ (linha 4). Em caso afirmativo, a diferença entre o ganho em saldo de se executar q'_i em me' e o ganho em saldo de se executar q'_i em me_k é acumulada na variável Δ_{saldo_+} (linha 5). Além disso, na linha 6, a consulta q'_i é escalonada em me_k da mesma forma que no Algoritmo 12 (algoritmo principal do

módulo escalonador). Consequentemente, a fila F' e a variável Δ_{saldo_+} são atualizadas a cada execução do Algoritmo 15.

Por fim, se a variável Δ_{saldo_+} for maior que o custo (obtido pela variável c) de se criar uma nova instância do módulo escalonador (linha 7), o módulo provedor de recursos solicitará que seja criada outra instância do módulo escalonador (linha 8). Neste caso, a fila F' e a variável Δ_{saldo_+} serão reinicializadas (linhas 9 e 10, respectivamente) e uma nova instância hipotética me' será criada, a fim de serem utilizadas nas próximas execuções do Algoritmo 15.

O Algoritmo 16 avalia a possibilidade de suspender uma VME. Para isso, utiliza o conceito de folga estimada de uma fila F, o qual é representado por FE_F e calculado por meio da Fórmula 5.22. Desta forma, FE_F consiste no somatório das folgas estimadas de cada uma das instâncias de consulta existentes na fila F. Assim, $fe_{q'_j}$ representa a folga estimada da consulta q'_j, a qual ocupa a posição j da fila F.

$$FE_F = \sum_{j=1}^{|F|-1} fe_{q'_j} \tag{5.22}$$

A folga estimada de uma determinada consulta q'_j, representada por $fe_{q'_j}$ é calculada pela Fórmula 5.23. Assim, $fe_{q'_j}$ consiste na diferença entre $SLO_{q'_j}$ e $tre_{q'_j}(tie_{q'_j})$, onde $SLO_{q'_j}$ é o tempo de resposta máximo (SLO) definido no SLA de q'_j, e $tre_{q'_j}(tie_{q'_j})$ é o tempo de resposta estimado para a execução de q'_j, considerando que q'_j inicie sua execução no tempo de execução pré-definido $tie_{q'_j}$ de q'_j. Vale salientar que o tempo $tre_{q'_j}$ é calculado pela Fórmula 5.7.

$$fe_{q'_j} = SLO_{q'_j} - tre_{q'_j}(tie_{q'_j}) \tag{5.23}$$

Toda vez que uma consulta termina sua execução, o Algoritmo 11 (algoritmo para recebimento do resultado da execução de uma consulta q'_i executada em uma instância me_k do módulo escalonador pelo módulo despachante) chama o Algoritmo 16. Este algoritmo irá avaliar a possibilidade de suspender uma das VMEs.

O Algoritmo 16 possui como dado de entrada a lista V de instâncias disponíveis do módulo escalonador e, como variáveis globais o fator de interação global f_{global} e o custo operacional c por unidade faturável do tipo de VM a ser instanciado.

Seja $me_k \in V$ uma instância do módulo escalonador na lista V e, F o conjunto das consultas na fila de escalonamento de me_k. Inicialmente, o Algoritmo 16 obtém FE_{max} (linha 2), o valor da maior folga estimada considerando-se a folga estimada de cada $me_k \in V$. Na linha 3, o algoritmo recupera o conjunto V_{max} de todas as VMEs cuja folga estimada seja igual a FE_{max}, ou seja, que apresentem a maior folga estimada. Na linha 4, a variável me_{max} recebe uma das VMEs pertencentes ao conjunto V_{max}. Na linha 5, a variável FE_{min} recebe o valor da menor folga estimada considerando-se a folga estimada de cada $me_k \in V$. Na linha 6, o algoritmo recupera o conjunto V_{min} de todas as VMEs cuja folga estimada seja igual a FE_{min}, ou seja, que apresentem a menor folga estimada. Na linha 7, a variável me_{min} recebe uma das VMEs pertencentes ao conjunto V_{min}.

Algoritmo 16: Algoritmo para avaliar a possibilidade de suspender uma VME

Entrada: lista V de instâncias do módulo escalonador
Dados: fator de interação global f_{global}, custo operacional c por unidade faturável

1 **início**
2 $FE_{max} \leftarrow \max\{FE_{me_k} \mid me_k \in V\}$;
3 $V_{max} \leftarrow \{me_k \in V \mid FE_{me_k} = FE_{max}\}$;
4 $me_{max} \leftarrow$ `obterElemento(`V_{max}`)`;
5 $FE_{min} \leftarrow \min\{FE_{me_k} \mid me_k \in V\}$;
6 $V_{min} \leftarrow \{me_k \in V \mid FE_{me_k} = FE_{min}\}$;
7 $me_{min} \leftarrow$ `obterElemento(`V_{min}`)`;
8 $tee_{min} \leftarrow \min\{tee_{me_k} \mid me_k \in V\}$;
9 $V_{tempo} \leftarrow \{me_k \in V \mid tee_{me_k} = tee_{min}\}$;
10 $me_{tempo} \leftarrow$ `obterElemento(`V_{tempo}`)`;
11 $F' \leftarrow$ `copiarFilaEscalonamento(`me_{max}`)`;
12 $F'_{max} \leftarrow$ `copiarFilaEscalonamento(`me_{max}`)`;
13 $F'_{min} \leftarrow$ `copiarFilaEscalonamento(`me_{min}`)`;
14 **para cada** $q'_j \in F'_{min}$ **faça**
15 `escalonar(`q'_j, F', f_{global}`)`;
16 **fim para cada**
17 **se** $SEF_{F'} > SEF_{F'_{min}} + SEF_{F'_{max}} - c$ **então**
18 `Algoritmo18(`me_{tempo}`)`;
19 `solicitarSuspensaoVME(`me_{tempo}`)`;
20 **fim se**
21 **fim**

Em seguida, na linha 8, a variável tee_{min} recebe o valor do menor tempo de execução estimado considerando-se o tempo de execução estimado de cada $me_k \in V$, tee_{me_k}. O tempo de execução estimado tee_{me_k} de uma determinada instância do módulo escalonador me_k consiste no somatório do tempo de execução estimado de cada uma das instâncias de consulta em sua fila de escalonamento. Na linha 9, o algoritmo recupera o conjunto V_{tempo} de todas as VMEs cujo tempo de execução estimado seja igual a tee_{min}, ou seja, que apresentem o menor tempo de execução estimado. Na linha 10, a variável me_{tempo} recebe uma das VMEs pertencentes ao conjunto V_{tempo}.

Nas linhas 11 e 12, são criadas duas cópias (F' e F'_{max}) da fila de escalonamento da instância me_{max} (obtida na linha 4). Na linha 13, é criada uma cópia (F'_{min}) da fila de escalonamento da instância me_{min} (obtida na linha 7). Posteriormente, o laço de repetição entre as linhas 14 e 16, escalonam todas as consultas existentes na fila F'_{min} na fila F'. Assim, incluem-se todas as consultas da fila com menor folga na cópia da fila que possui a maior folga.

Na linha 17, o algoritmo verifica se o saldo estimado da fila F' (que inclui as consultas das filas com menor e com maior folgas estimadas) é maior do que a soma da estimativa de saldo $SEF_{F'_{max}}$ da fila F'_{max} com a estimativa do saldo $SEF_{F'_{min}}$ da fila F'_{min} subtraída do custo operacional c. Em caso afirmativo, a instância me_{tempo} será suspensa (linha 19) após seu registro no módulo despachante ser removido (linha 18). A remoção do registro, no módulo despachante, da instância me_{tempo} é realizada pelo Algoritmo 18 (algoritmo para remoção do registro, no

módulo despachante, de uma dada instância me_k do módulo escalonador), o qual é executado pelo módulo despachante.

Algoritmo 17: Algoritmo para registro de uma nova instância do módulo escalonador no módulo despachante

```
   Entrada: url
   Dados: lista V de instâncias do módulo escalonador
1  início
2  |  me_k ← instanciarEscalonadora(url);
3  |  V.inserir(me_k);
4  fim
```

Seja me_k uma instância do módulo escalonador pertencente a uma VME recém-criada pelo provedor de IaaS. Para que o módulo despachante possa realizar o despacho de consultas, é necessário que existam instâncias do módulo escalonador disponíveis para execução de consultas. Com essa finalidade, o Algoritmo 17 (algoritmo para registro de uma nova instância do módulo escalonador no módulo despachante), executado pelo módulo despachante, registra a instância me_k do módulo escalonador no módulo despachante. Este algoritmo é chamado pelo módulo provedor de recursos quando este recebe uma notificação de instanciação de uma dada VME, originada do provedor de IaaS. O Algoritmo 17 possui como dado de entrada a localização *url* da instância me_k. Inicialmente, na linha 2, o Algoritmo 17 cria uma estrutura de dados responsável pela comunicação do módulo despachante com a instância me_k. Por fim, esta estrutura de dados é adicionada em uma lista V de instâncias do módulo escalonador disponíveis (linha 3).

Seja me_k a instância do módulo escalonador pertencente à próxima VME a ser suspensa. Quando o módulo provedor de recursos assume que está ocorrendo subutilização de VMEs, este solicita ao módulo despachante que remova o registro de me_k no próprio módulo despachante. A remoção de registro de me_k é realizada pelo Algoritmo 18 (algoritmo para remoção do registro, no módulo despachante, de uma dada instância me_k do módulo escalonador), que apresenta a instância me_k como dado de entrada. O Algoritmo 18 é chamado pelo Algoritmo 16 (algoritmo para avaliar a suspensão de VMEs) e, inicialmente, remove me_k da lista V de instâncias do módulo despachante disponíveis (linha 2). Em seguida, o módulo despachante (executor do Algoritmo 18) espera que me_k execute todas as consultas sob sua responsabilidade (linha 3). Assim, o módulo provedor de recursos poderá solicitar ao provedor de IaaS a suspensão da VME onde me_k está localizada.

Algoritmo 18: Algoritmo para remoção do registro, no módulo despachante, de uma dada instância me_k do módulo escalonador

```
   Entrada: referência me_k a uma dada instância do módulo escalonador
1  início
2  |  me_k ← V.remover(me_k);
3  |  esperarNotificacaoVazia(me_k);
4  fim
```

5.5 Experimentos

A fim de avaliar a utilização do *framework* GeDaNIC, o qual busca explorar os benefícios proporcionados pelas interações entre consultas, diversos testes foram executados. Os experimentos realizados avaliaram a eficácia e a eficiência do *framework* proposto.

5.5.1 Configuração do Ambiente de Testes

Os testes discutidos nesta seção foram executados na nuvem da Amazon. Para a execução dos testes, cinco máquinas virtuais do tipo *Small* foram instanciadas. Dessas, uma máquina virtual foi configurada como VMC (VM controladora), e as outras quatro máquinas virtuais foram configuradas como VMEs (VMs escalonadoras). As VMs do tipo *Small*, da Amazon, possuem 1,7 GB de memória principal. O sistema operacional utilizado nas máquinas virtuais foi o Ubuntu 12.04 LTS Precise. Além disso, o *benchmark* TPC-H foi escolhido para modelar tanto o banco de dados quanto as cargas de trabalho. Para gerar o banco de dados utilizamos o *Database Test 3* (DBT-3) com fator de escala de 1 GB. O SGBD PostgreSQL 9.1.3 foi utilizado para armazenar e gerenciar o banco de dados. O PostgreSQL 9.1.3 foi utilizado deixando-se todos os seus parâmetros com valores padrão. Ademais, cada uma das quatro VMEs armazenou uma réplica completa do banco de dados e hospedou uma instância do PostgreSQL 9.1.3.

Para compor as cargas de trabalho, todos os *templates* de consultas do TPC-H foram utilizados (22, no total). Cada *template* foi considerado como um tipo de consulta. Foram criadas 105 instâncias de cada tipo de consulta, das quais 5 foram utilizadas no pré-processamento para o cálculo do fator de interação pelas abordagens DRR e GBA. Essas instâncias de consulta foram concebidas atribuindo-se diferentes valores para seus parâmetros, que compõem cada tipo de consulta (*template*). A ferramenta QGEN (que faz parte do DBT-3) foi utilizada para gerar as instâncias de consulta. Essa ferramenta define os parâmetros aleatoriamente.

5.5.2 Resultado dos Testes

Com o objetivo de demonstrar que a interação entre consultas pode ser usada para aprimorar as tomadas de decisão que compõem um serviço de gerenciamento de dados em nuvem, o *framework* GeDaNIC foi avaliado em conjunto com cada uma das três abordagens propostas, no Capítulo 4, para medir as interações entre consultas: IS, DRR e GBA. Nos experimentos realizados, o *framework* GeDaNIC foi executado com apenas dois módulos habilitados: o despachante e o escalonador de consultas. O módulo provedor de recursos não foi utilizado nos experimentos, uma vez que sua implementação não havia sido concluída. Nos testes, foram utilizadas 20 cargas de trabalho. Cada carga de trabalho foi composta por 5 instâncias de consulta de cada um dos 22 tipos de consulta do TPC-H. Logo, cada carga de trabalho conteve 110 instâncias de consulta. Durante os testes, cinco estratégias diferentes para o gerenciamento de dados em nuvem (mais especificamente, para os problemas de despacho e escalonamento de consultas) foram avaliadas. A seguir, descreveremos cada uma delas:

- **Roundrobin/FIFO:** nesta estratégia, o módulo despachante realiza o despacho por meio da abordagem *roundrobin* e o módulo de escalonamento realiza o escalonamento usando *FIFO*;

- **SLA-Tree:** nesta estratégia, tanto o despacho quanto o escalonamento de uma dada consulta q_i' se utilizam da abordagem proposta por (CHI et al., 2011). Contudo, os autores não definiram o critério de desempate, na solução do despacho, quando mais de uma instância do módulo escalonador gera a mesma estimativa de ganho em saldo para a consulta a ser despachada. Deste modo, optamos por despachar uma dada consulta q_i', quando ocorrer empate, para a instância do módulo escalonador com menor tempo estimado para execução de todas as consultas sob sua responsabilidade;

- **GeDaNIC/IS:** utiliza o *framework* GeDaNIC para solucionar os problemas de despacho e escalonamento de consultas (conforme as seções 5.2 e 5.3, respectivamente). Porém, nesta estratégia o GeDaNIC foi configurado para utilizar a abordagem IS (definida na Subseção 4.2.1) para realizar o cálculo do fator de interação entre as consultas;

- **GeDaNIC/DRR:** utiliza o *framework* GeDaNIC para solucionar os problemas de despacho e escalonamento de consultas (conforme as seções 5.2 e 5.3, respectivamente). Porém, nesta estratégia o GeDaNIC foi configurado para utilizar a abordagem DRR (definida na Subseção 4.2.2) para realizar o cálculo do fator de interação entre as consultas;

- **GeDaNIC/GBA:** utiliza o *framework* GeDaNIC para solucionar os problemas de despacho e escalonamento de consultas (conforme as seções 5.2 e 5.3, respectivamente). Porém, nesta estratégia o GeDaNIC foi configurado para utilizar a abordagem GBA (definida na Subseção 4.2.3) para realizar o cálculo do fator de interação entre as consultas.

Com a finalidade de analisar a eficácia das estratégias SLA-Tree, GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA, utilizamos a estratégia Roundrobin/FIFO como *baseline*. Assim, executaram-se para cada uma dessas estratégias as 20 cargas de trabalho geradas aleatoriamente (conforme descrito anteriormente). Sempre reiniciando o ambiente após a avaliação de cada uma das estratégias (ou seja, após a execução das 20 cargas de trabalho). Para medir a eficácia de uma determinada estratégia utilizamos o número de casos (cargas de trabalho) em que esta apresentou saldo (receita - penalidade) maior do que o saldo proporcionado pelo Roundrobin/FIFO (*baseline*). A razão entre esse valor e a quantidade total de casos (20) consiste na eficácia de uma determinada estratégia. Vale destacar que uma vez que o custo operacional era o mesmo para todas as estratégias resolvemos desconsiderá-lo para efeitos de comparação. Por este motivo, utilizamos o saldo (receita - penalidade) e não o lucro. O saldo obtido foi medido em uma unidade monetária fictícia. Contudo, poderíamos ter utilizado como unidade monetária o dólar, ou outra moeda.

A Figura 5.7 mostra o resultado do teste realizado para avaliar a eficácia. Observe que o SLA-Tree apresentou uma eficácia de 60%, ou seja, o saldo proporcionado pela estratégia SLA-Tree foi maior que o saldo proporcionado pela estratégia Roundrobin/FIFO em 60% dos casos. Logo, em 12 das 20 cargas de trabalho o saldo proporcionado pelo SLA-Tree foi superior ao saldo proporcionado pelo Roundrobin/FIFO. Além disso, pode-se notar que todas

as abordagens orientadas ao lucro apresentaram eficácia maior que Roundrobin/FIFO. Porém, a estratégia SLA-Tree teve uma eficácia de 60% somente. Isto se deve ao fato de que a abordagem SLA-Tree não explorou os benefícios proporcionados pelas interações entre consultas para efetuar o despacho e escalonamento das mesmas. Já as estratégias GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA apresentaram 95%, 90% e 85% de eficácia, respectivamente. Esse resultado aponta que a utilização do conceito de interação entre consultas pode influenciar no lucro obtido pelo provedor do serviço de dados.

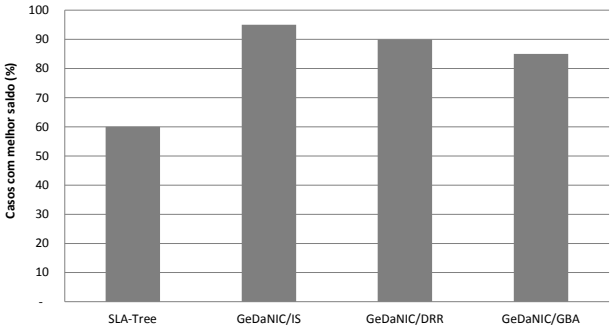
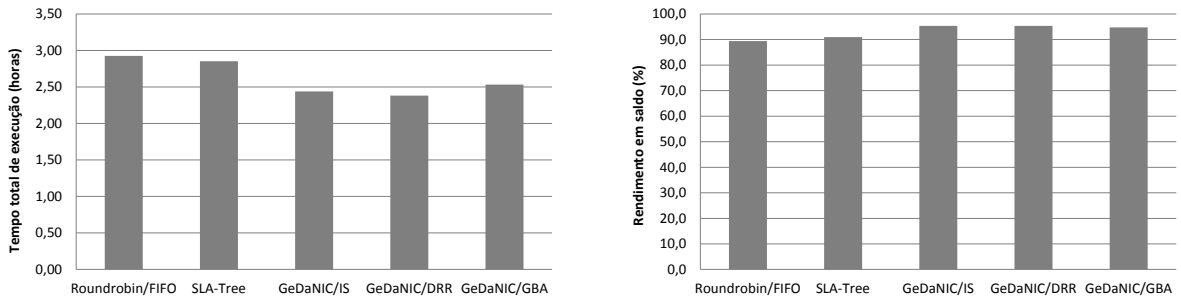


Figura 5.7: Análise comparativa da eficácia das estratégias Roundrobin/FIFO, SLA-Tree, GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA

Com a finalidade de avaliar a eficiência das estratégias Roundrobin/FIFO, SLA-Tree, GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA em relação ao gerenciamento de dados em nuvem (despacho e escalonamento de consultas), analisamos duas métricas distintas: o desempenho e o rendimento em saldo. Para mensurar o desempenho de cada uma das estratégias investigadas, medimos o tempo total que cada abordagem necessitou para executar as 20 cargas de trabalho. A Figura 5.8a ilustra o resultado da avaliação de desempenho. Por meio desta figura, pode-se observar que a estratégia Roundrobin/FIFO executou as 20 cargas de trabalho submetidas em 2,93 horas. A abordagem SLA-Tree efetuou a mesma tarefa em 2,85 horas (redução de 2,5%). As abordagens propostas neste trabalho conseguiram uma economia no tempo de resposta mais significativa: GeDaNIC/IS executou as 20 cargas de trabalho em 2,44 horas (16,71% de redução); GeDaNIC/DRR, em 2,38 horas (18,64% de redução) e, GeDaNIC/GBA, em 2,53 horas (economizando 13.42%).



(a) Desempenho

(b) Rendimento em saldo

Figura 5.8: Análise comparativa da eficiência das estratégias Roundrobin/FIFO, SLA-Tree, GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA

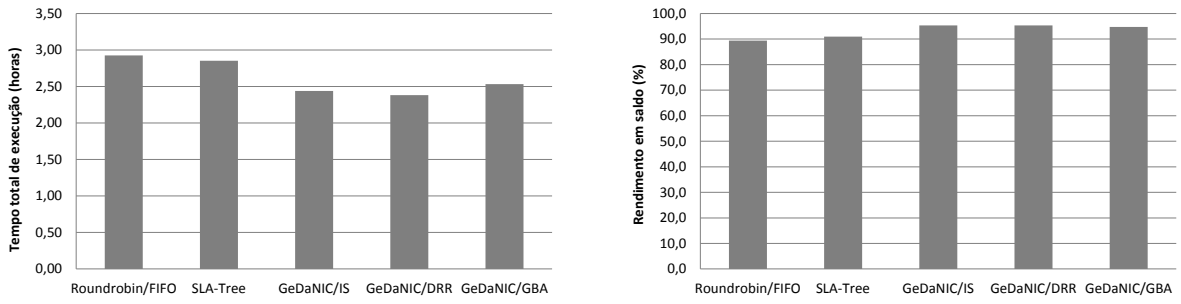
Para calcular o rendimento em saldo, utilizamos a seguinte estratégia. Primeiramente, para cada estratégia avaliada, executaram-se as 20 cargas de trabalho e somou-se o saldo

proporcionado após a execução de cada carga de trabalho. Denominamos esse valor de saldo total. Em seguida, calculamos o maior saldo que seria possível obter. Denominamos esse valor de saldo máximo possível. Esse valor pode ser calculado somando-se a receita de cada uma das instâncias de consulta que compõem as 20 cargas de trabalho. Esse valor está presente no SLA de cada instância de consulta. Assim, para este cálculo, assume-se que todas as instâncias de consulta foram executadas dentro dos limites acordados nos SLAs. Logo, nenhuma penalidade foi imposta. Agora, para calcular o rendimento em saldo proporcionado por uma determinada estratégia basta dividir o saldo total proporcionado por esta estratégia pelo saldo máximo possível.

A Figura 5.8a ilustra o resultado da avaliação de desempenho. Por meio desta figura, pode-se observar que a estratégia Roundrobin/FIFO executou as 20 cargas de trabalho submetidas em 2,93 horas. A abordagem SLA-Tree efetuou a mesma tarefa em 2,85 horas (redução de 2,5%). As abordagens propostas neste trabalho conseguiram uma economia no tempo de resposta mais significativa: GeDaNIC/IS executou as 20 cargas de trabalho em 2,44 horas (16,71% de redução); GeDaNIC/DRR, em 2,38 horas (18,64% de redução) e, GeDaNIC/GBA, em 2,53 horas (economizando 13,42%).

A Figura 5.8b mostra o rendimento em saldo proporcionado por cada uma das estratégias avaliadas. Observe que Roundrobin/FIFO e SLA-Tree, que são abordagens que não consideram a interação entre consultas, apresentaram 89,4% e 91% de rendimento em saldo, respectivamente. Já as estratégias GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA, por sua vez, obtiveram 95,3%, 95,4%, 94,8% de rendimento em saldo, respectivamente. Esse resultado aponta que a utilização do conceito de interação entre consultas pode influenciar positivamente no lucro obtido pelo provedor do serviço de dados na nuvem.

Outro teste realizado foi a avaliação das abordagens de despacho *Roundrobin*, *SLA-Tree*, GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA quando executadas com o escalonador FIFO. Utilizou-se a mesma carga de trabalho do teste anterior e a mesma configuração de VMs. Entretanto este experimento foi realizado em instâncias de VM diferentes daquelas utilizadas no teste anterior. Por isso a abordagem Roundrobin/FIFO foi reexecutada, e como é possível observar na Figura 5.9, os tempos de execução e o rendimento em saldo divergiram daquele encontrado no experimento anterior.



(a) Desempenho

(b) Rendimento em saldo

Figura 5.9: Análise comparativa da eficiência das estratégias de despacho utilizadas pelas abordagens Roundrobin/FIFO, SLA-Tree, GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA

A Figura 5.9a ilustra o resultado da avaliação de desempenho das estratégias de despacho avaliadas quando executadas conjuntamente com o escalonador FIFO. Por meio desta figura, pode-se observar que a estratégia Roundrobin/FIFO executou as 20 cargas de trabalho submetidas em 1,84 hora. A abordagem SLA-Tree efetuou a mesma tarefa em 1,94 hora (5,4% de aumento). Por sua vez, as abordagens propostas neste trabalho conseguiram os seguintes tempos de execução total, considerando as 20 cargas de trabalho utilizadas nos experimentos: GeDaNIC/IS executou em 1,98 hora (acréscimo de 7,6%); GeDaNIC/DRR, em 2,07 horas (12,5% de aumento) e, GeDaNIC/GBA, em 2 horas (acréscimo de 8,7%).

A Figura 5.9b mostra o rendimento em saldo proporcionado por cada uma das estratégias de despacho avaliadas quando executadas conjuntamente com o escalonador FIFO. As abordagens Roundrobin/FIFO, SLA-Tree, GeDaNIC/IS, GeDaNIC/DRR e GeDaNIC/GBA apresentaram 91,1%, 89,7%, 88,3%, 87,6% e 88,3% de rendimento em saldo, respectivamente.

No experimento representado pela Figura 5.9, as abordagens orientadas ao lucro e baseadas na interação entre consultas buscam alocar consultas a VMEs de forma a aumentar o lucro do provedor do serviço de dados. Todavia, a abordagem de escalonamento FIFO executa as consultas exclusivamente em função suas ordens de chegada na fila de escalonamento, não alterando a ordem de execução das consultas, caso seja verificado que essa modificação venha a trazer maior lucro ao provedor do serviço de dados. Deste modo, a tomada de decisão realizada pela abordagem de despacho do GeDaNIC, neste caso, acaba não influenciando positivamente no lucro do provedor do serviço de dados.

Adicionalmente, vale destacar que o saldo total obtido pelo provedor do serviço de dados em nuvem pode ser influenciado por diversos fatores. Dentre eles, destacam-se:

- **SLO:** quanto maior o tempo máximo de resposta acordado a uma dada consulta, maiores as chances de cumprimento de seu SLA;
- **receita:** a forma como a receita é definida entre os tipos de consulta impacta no peso de cada uma de suas instâncias para o lucro do provedor do serviço de dados;
- **penalidades:** analogamente à receita, a forma como a penalidade é acordada para os tipos de consulta impacta no peso de cada uma de suas instâncias para o lucro do provedor do serviço de dados;
- **instâncias de VMs:** a quantidade de instâncias de VMs, bem como seus tipos, influenciam no desempenho da carga de trabalho, aumentando/diminuindo a ocorrência de penalidades.

6 TRABALHOS RELACIONADOS

Este trabalho propõe uma solução para gerenciamento de dados em nuvem baseada na propriedade de interação entre consultas. Dessa forma, uma parte dos trabalhos relacionados pesquisados descreve abordagens que consideram aspectos de interação entre consultas (Seção 6.1) e a outra parte trata de soluções para gerenciamento de dados em nuvem (Seção 6.2).

6.1 Interação entre Consultas

Há pouco trabalho discutindo sobre interação entre consultas. (O'GORMAN; ABBADI; AGRAWAL, 2005), por exemplo, descreve um algoritmo de escalonamento baseado em uma matriz bidimensional que armazena em cada célula $c_{i,j}$ uma taxa que leva em consideração a execução de uma instância de consulta q_i antes de uma instância de consulta q_j, bem como a execução concorrente de q_i e q_j. (AHMAD et al., 2008, 2011) descrevem um modelo dirigido a experimentos para escalonamento de consultas em lote. Adicionalmente, os autores propõem um algoritmo de escalonamento para tomada de decisão em tempo real baseado em uma métrica denominada NRO (Normalized Runtime Overhead). Todas essas abordagens necessitam de um pré-processamento para serem utilizadas.

Alguns trabalhos são intrusivos, como (ROY et al., 2000) e (TAN; LU, 1995). Cada qual realiza modificações no otimizador de consultas do SGBD para explorar algumas propriedades e reaproveitar dados em comum entre consultas, que foram armazenados em memória.

Existem trabalhos (por exemplo, (NIU et al., 2007) e (NIU; MARTIN; POWLEY, 2009)) que se baseiam nas estimativas de custo geradas pelos planos de execução das consultas para dar suporte à decisão no gerenciamento de cargas de trabalho, incluindo decisões de escalonamento. Entretanto, estas estimativas não dão indícios se uma instância de consulta q_i interage eficientemente com uma instância de consulta q_j. De fato, algumas das abordagens propostas pela nossa solução são baseadas no plano de execução das consultas, porém suas estimativas de custo não são utilizadas.

6.2 Gerenciamento de Dados em Nuvem

Os problemas de gerenciamento de dados discutidos neste trabalho são: despacho, escalonamento e planejamento de capacidade (definidos no Capítulo 3). Dentre todos os trabalhos relacionados ao gerenciamento de dados em nuvem, não identificamos nenhum que explorasse a propriedade de interação entre consultas nas soluções propostas.

Alguns trabalhos tratam especificamente do problema de despacho. Por exemplo, (XIONG et al., 2011a) propõe um *framework* orientado ao lucro para despacho de consultas no serviço de dados em nuvem. Este *framework* estima a probabilidade de uma dada consulta q_i cumprir seu SLO por meio de técnicas de aprendizagem de máquina e, baseado nessa probabilidade, despacha ou não a consulta q_i. (SCHROEDER; HARCHOL-BALTER, 2004) propõe

uma abordagem que classifica as consultas em grupos e cada servidor disponível pode receber somente consultas de um determinado grupo. Todavia, esta solução não é voltada para a nuvem.

Com relação ao escalonamento de consultas, (SHARAF et al., 2009) propõe uma abordagem de escalonamento de requisições de I/O com o objetivo de diminuir o atraso de consultas, quando seus SLAs não são cumpridos. Vale salientar que esse trabalho não define a ordem de execução das consultas, mas sim, a ordem de execução das requisições de I/O originadas pela execução de consultas. Já em (TILGNER, 2010) é proposto um escalonador, o qual trata do problema de escalonamento utilizando-se da métrica do SLA de consistência. Neste escalonador, as consultas são executadas concorrentemente e enviadas, em lote, para o SBD, que será responsável pelo escalonamento desse lote de consultas. (COSTA; FURTADO, 2008) propõe um escalonador que varia a quantidade de consultas executando concorrentemente baseado nos SLOs das consultas e em estimativas de tempo baseadas nas estimativas de custo do otimizador de consultas.

Alguns trabalhos lidam especificamente com o planejamento de capacidade. (IQBAL et al., 2011) propõe uma abordagem reativa para instanciação de VMs (recursos são adicionados somente quando o serviço de dados está sobrecarregado). Esta abordagem monitora o tempo de resposta das consultas, bem como coleta estatísticas de utilização da CPU, para decidir sobre a quantidade de VMs a serem instanciadas. Todavia, a suspensão de VMs se dá por meio previsões baseadas em execuções anteriores de consultas. Por sua vez, (SHARMA; SHENOY; TOWSLEY, 2012) propõe uma solução cujo objetivo é o de minimizar o custo operacional (custo com utilização de VMs), respeitando a restrição sobre a porcentagem de consultas com tempo de resposta de no máximo t. Este trabalho utiliza a métrica do SLA de tempo resposta das consultas. Porém, o SLA é definido em função de um percentil de consultas que atendam um determinado tempo de resposta t (por exemplo, 90% das consultas devem possuir no máximo 3 segundos de tempo de resposta), diferentemente do SLA utilizado no GeDaNIC. Já (SAKR; LIU, 2012) propõe um *framework* que monitora as cargas de trabalho com relação ao cumprimento dos SLAs, tendo como métrica o tempo de resposta das consultas, e instancia/suspende VMs baseado em regras definidas pela aplicação cliente.

O *framework* proposto neste trabalho (GeDaNIC) foi baseado na abordagem proposta em (CHI et al., 2011), a qual utiliza uma árvore (*SLA-Tree*) com o objetivo de facilitar o acesso às informações que auxiliam a tomada de decisão baseada no lucro. A estrutura *SLA-Tree* apresenta sobrecarga em sua criação e atualização, de ordem de complexidade superior à linear, além de não estar adaptada à propriedade de interação entre consultas. Desta forma, optamos por uma estrutura de fila para compor o GeDaNIC por ser uma estrutura de dados mais simples e de fácil adaptação à estrutura de dados EDConsultaSLA (descrita na Seção 5.2) utilizada para representar as consultas e seus SLAs. Além disso, o mecanismo proposto no GeDaNIC busca agrupar consultas com maior interação na mesma instância do módulo escalonador, a fim de melhorar o desempenho das consultas.

A fim de mostrar pontos possíveis de melhoria que o *framework* proposto visa solucionar, a seguir será feito um comparativo entre alguns trabalhos relacionados (CHI et al., 2011; ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010; PATON et al., 2009; SOROR et al., 2010; SHIVAM et al., 2007). Os trabalhos apresentados em (ROGERS; PAPAEMMA-

NOUIL; ÇETINTEMEL, 2010; SHIVAM et al., 2007) propõem soluções para o planejamento de capacidade. Por sua vez, (SOROR et al., 2010) aborda o planejamento de capacidade sob o aspecto de modificação dos recursos de *hardware* das VMs disponíveis, ou seja, não instancia/suspende VMs. Já a abordagem proposta em (PATON et al., 2009) busca solucionar somente o problema do despacho de consultas. Em (CHI et al., 2011), os autores propõem uma nova estrutura de dados, denominada *SLA-Tree*, com a finalidade de fornecer suporte para a tomada de decisão baseada no lucro. Além disso, (CHI et al., 2011) propõe maneiras de se utilizar a *SLA-Tree* em soluções de despacho, escalonamento e provisionamento de recursos. Contudo, nenhum dos trabalhos relacionados considera a interação entre consultas para o despacho, escalonamento ou planejamento de capacidade.

6.2.1 Resolução dos Principais Problemas de Gerenciamento de Dados em Nuvem

O gerenciamento de dados em nuvem traz os seguintes problemas: despacho, escalonamento e provisionamento de recursos. A Tabela 6.1 sintetiza o tratamento dos trabalhos relacionados sobre esses problemas e, como se pode observar, somente (CHI et al., 2011) lida com todos os problemas apresentados.

Tabela 6.1: Análise comparativa: principais problemas do gerenciamento de dados em nuvem

Trabalho relacionado	Despacho	Escalonamento	Provisionamento de recursos
(CHI et al., 2011)	Sim	Sim	Sim
(ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010)	Sim	Não	Sim
(PATON et al., 2009)	Sim	Não	Não
(SOROR et al., 2010)	Não	Não	Não
(SHIVAM et al., 2007)	Não	Não	Sim

(CHI et al., 2011) propõe uma solução centrada em uma estrutura de dados que serve para auxiliar na tomada de decisões sobre o gerenciamento de bancos de dados em nuvem. Este trabalho mostra exemplos de utilização dessa estrutura, que auxilia na obtenção das respostas para perguntas pré-definidas, tratando, dessa forma, os problemas de despacho, escalonamento e provisionamento de recursos. Entretanto, não há descrição de uma solução para o provisionamento de recursos do ponto de vista da suspensão de VMs.

O trabalho de (ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010) descreve uma solução que lida com os problemas de despacho e de provisionamento de recursos. Porém o único critério de despacho é enviar uma consulta para uma VM que corresponda à sua classe (tipo) de consulta de acordo com a disponibilidade dessas VMs. Ou seja, as consultas não podem ir para todas as VMs, gerando limitações quanto aos destinos e podendo demandar, portanto, maior quantidade de instâncias de VMs. Com relação ao provisionamento de recursos, é proposta uma abordagem que se utiliza de estatísticas do SGBD e outra que é baseada

em cargas de trabalho que representem as cargas reais. Adicionalmente, o escalonamento de consultas não é contemplado.

Seja W uma carga de trabalho. Uma proposta de despacho de consultas é apresentada por (PATON et al., 2009), que consiste em alocar consultas a *sites* de execução, baseando-se em funções de utilidade. Estas funções, no contexto desse trabalho, consideram parâmetros de qualidade de serviço (QoS), como o tempo de resposta de cada consulta $q_i \in W$. A abordagem proposta não trata do escalonamento, apesar de comentá-lo como problema, além de não contemplar o provisionamento de recursos.

Outro trabalho relacionado é o apresentado em (SOROR et al., 2010). Este descreve uma ferramenta que auxilia na alocação de recursos para máquinas virtuais que hospedam SBDs e é baseada em estimativas de custo de *hardware*. Este trabalho não possui o contexto da nuvem, bem como só é aplicado ao planejamento de capacidade em uma modalidade mais específica, denominada configuração automática de VMs, que consiste, basicamente, em modificar os recursos de *hardware* (memória, espaço em disco, CPU, etc.) de cada uma das VMs, individualmente, caso seja previsto esse tipo de necessidade.

(SHIVAM et al., 2007) descreve duas ferramentas que em conjunto realizam o provisionamento de recursos, baseado na modelagem de desempenho das possíveis configurações de VMs, dados uma determinada carga de trabalho e uma base de dados que represente o conjunto de dados acessados pela aplicação cliente. Logo, este trabalho soluciona somente o provisionamento de recursos, dentre os três problemas presentes na Tabela 6.1.

6.2.2 Requisitos para uma Solução em Computação em Nuvem

Quando se propõe uma solução de gerenciamento de dados para o ambiente em nuvem, alguns aspectos são relevantes, dentre eles: SLA, elasticidade rápida, orientado ao lucro e ajuste automático da solução. Pode-se observar na Tabela 6.2 como os trabalhos relacionados se comportam quanto a esses requisitos.

Tabela 6.2: Análise comparativa: requisitos para uma solução em computação em nuvem

Trabalho relacionado	SLA	Elasticidade rápida	Orientado ao lucro	Ajuste automático da solução
(CHI et al., 2011)	Sim	Sim	Sim	Sim
(ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010)	Sim	Não	Sim	Sim
(PATON et al., 2009)	Sim	Não	Não	Sim
(SOROR et al., 2010)	Não	Não	Não	Sim
(SHIVAM et al., 2007)	Sim	Não	Não	Sim

A solução proposta por (CHI et al., 2011) é orientada ao lucro. A tomada de decisão é baseada no SLA. O SLA é definido por consulta e a métrica utilizada é o tempo de resposta. (CHI et al., 2011) se preocupa com a elasticidade e ajuste automático da solução, exemplifi-

cando uma possível utilização do *SLA-Tree* para previsão da necessidade de instanciação de VMs. Porém, como a solução não é baseada na interação entre consultas, podem ser requisitadas mais VMs do que o necessário para atender os SLAs acordados, devido às interações não previstas. Ademais, (CHI et al., 2011) não propõe uma solução para identificar a necessidade de suspensão de VMs.

(ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010) também apresenta uma solução orientada ao lucro. Este trabalho se baseia no tempo de resposta e vazão para a tomada de decisões sobre provisionamento de recursos. Porém, os autores não mencionam como os SLOs são definidos. Nesta abordagem, se a carga de trabalho aumentar inesperadamente, somente haverá ajuste na quantidade de VMs após o período estipulado para possíveis modificações na quantidade de VMs (no ambiente descrito, era a cada hora). Ou seja, o ajuste automático das VMs só é feito em um grande intervalo de tempo, não atendendo eficientemente a demanda inesperada por novos recursos.

A solução proposta por (PATON et al., 2009) não descreve como utilizar o SLA para realizar o despacho de consultas, apesar de mencionar a possibilidade de utilização do SLA nesta solução. Por se tratar de uma solução para despacho, não há preocupação quanto à elasticidade. Todavia, quando se identifica que determinado servidor está sobrecarregado, a solução de despacho envia as requisições para servidores com menor carga de execução.

A ferramenta desenvolvida por (SOROR et al., 2010) não é orientada ao lucro, nem se baseia em SLA. Seu objetivo é melhorar o tempo de resposta de cargas de trabalho por meio de ajustes na solução atual de configurações de VMs. Estes ajustes inicialmente são realizados por meio de estimativas do otimizador do SGBD. Contudo, com o decorrer do tempo há uma avaliação de tempos de resposta reais com o intuito de corrigir erros de precisão das estimativas feitas. Esta solução não lida com elasticidade porque é uma solução que, dada uma máquina física e um conjunto de VMs associadas a essa máquina, procura distribuir de forma eficiente os recursos de *hardware* entre essas VMs. Logo, quando a carga de trabalho aumentar e forem exigidas novas VMs, esta solução não atenderá a demanda.

A abordagem proposta por (SHIVAM et al., 2007) busca atender os SLAs acordados com o mínimo de recursos possíveis, porém o custo monetário não é levado em consideração nessa tomada de decisão. Ademais, (SHIVAM et al., 2007) ajusta automaticamente a disposição de VMs no serviço de dados. Entretanto, seja W a carga de trabalho a executar, se a carga W demandar um aumento súbito da quantidade de VMs disponíveis no serviço de dados, somente após a execução da carga W (ou de parte dela), a solução proposta por (SHIVAM et al., 2007) instanciará as VMs necessárias.

6.2.3 Requisitos Relacionados ao Gerenciamento de Dados em Nuvem

Alguns requisitos para uma solução em computação em nuvem estão mais diretamente ligados ao gerenciamento de dados. Deste modo, a Tabela 6.3 mostra como os trabalhos relacionados se comportam quanto a alguns deles.

A solução proposta por (CHI et al., 2011) suporta cargas de trabalho imprevistas, uma vez que não há suposições sobre que consultas serão submetidas ao provedor do serviço

Tabela 6.3: Análise comparativa: requisitos relacionados ao gerenciamento de dados em nuvem

Trabalho relacionado	Carga de trabalho imprevista	Replicação dos dados	Não intrusivo
(CHI et al., 2011)	Sim	Sim	Sim
(ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010)	Não	Sim	Sim
(PATON et al., 2009)	Não	Sim	Não informa
(SOROR et al., 2010)	Não	Sim	Sim
(SHIVAM et al., 2007)	Não	Sim	Sim

de dados em nuvem. Ademais, a solução proposta é não intrusiva e utiliza diversas réplicas do SBD acessado pela aplicação cliente.

No trabalho de (ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010), há uma etapa de pré-processamento, que supõe a utilização de uma dada VM por somente uma classe (tipo) de consulta. Além disso, a solução proposta é não intrusiva e as instâncias dos SGBDs são réplicas.

Seja W uma carga de trabalho. Em (PATON et al., 2009), funções de utilidade são propostas para auxiliar na tomada de decisão para o despacho de consultas. A aplicação dessas funções de utilidade depende da carga W. Assim, esta carga de trabalho precisa ser conhecida à priori. Além disso, a replicação dos dados é levada em consideração e não há indícios sobre a solução ser não intrusiva.

Dentre os requisitos relacionados ao gerenciamento de bancos de dados em nuvem, a solução proposta em (SOROR et al., 2010) é destinada a ambientes com bancos de dados replicados, bem como não é intrusiva. Esta solução busca definir a configuração das VMs disponíveis baseada em cargas de trabalho conhecidas a priori. Em contrapartida, operações de atualização sobre o SGBD são abrangidas.

(SHIVAM et al., 2007) propõe uma solução que se utiliza de cargas de trabalho anteriormente executadas para auxiliar na decisão sobre a instanciação de VMs . Além disso, este trabalho aborda a replicação de dados, bem como propõe uma solução não intrusiva.

6.2.4 Realização de Experimentos

Além dos requisitos descritos anteriormente, um aspecto importante de um trabalho é como os experimentos foram realizados para validar a solução proposta. A Tabela 6.4 mostra quais trabalhos descreveram algum tipo de experimentação. Vale salientar que (SHIVAM et al., 2007) não realizou experimentos para avaliar a solução proposta.

(CHI et al., 2011) avalia experimentalmente a solução proposta quanto ao escalonamento, despacho, provisionamento de recursos, verificação de robustez (se mesmo com estimativas imprecisas a solução traz bons resultados) e verificação do tempo de execução (tes-

Tabela 6.4: Análise comparativa: realização de experimentos

Trabalho relacionado	Experimentos
(CHI et al., 2011)	Sim
(ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010)	Sim
(PATON et al., 2009)	Sim
(SOROR et al., 2010)	Sim
(SHIVAM et al., 2007)	Não

tando com cargas de trabalho variadas). Foram utilizados três conjuntos de dados diferentes. O primeiro segue uma distribuição exponencial com tempo de execução médio de consultas de 20 ms; o segundo, uma distribuição de Pareto, derivada de sistemas reais, com tempo médio de execução de consultas de 25 ms e o terceiro, um *benchmark* derivado do TPC-H com tempo de execução médio de consultas de 10,2 ms. Segundo (CHI et al., 2011), as fontes desses conjuntos de dados foram (POPOVICI; WILKES, 2005), (GUPTA et al., 2009) e (ABADI; MADDEN; HACHEM, 2008), respectivamente. Em cada experimento, vinte mil consultas foram geradas, sendo dez mil para carregar dados na memória e os outros dez mil para medir o desempenho da solução proposta.

Os experimentos em (ROGERS; PAPAEMMANOUIL; ÇETINTEMEL, 2010) são realizados no PostgreSQL 8.4.1 com uma variedade de instâncias AWS (*Amazon Web Services*) configuradas em termos de utilização de CPU e de utilização de dispositivos de I/O. Um dos experimentos realizados tem como objetivo mostrar a eficácia que as estimativas do otimizador de consulta apresentam quando consultas são divididas em classes e somente consultas de determinada classe podem ser executadas em uma dada VM. Outro experimento foi realizado para mostrar que cargas de trabalho de naturezas (tipos) distintas podem necessitar de configurações distintas de VMs.

(PATON et al., 2009) fez somente um experimento, que consistiu em uma análise comparativa da solução proposta. Esse experimento consistia na execução de quatro consultas compostas por uma única junção em um ambiente composto por doze *sites* de execução em que um deles sofria interferência de outras tarefas em metade do tempo de experimento.

Os experimentos descritos em (SOROR et al., 2010) usaram o DB2 V9 e o PostgreSQL 8.1.3. O ambiente de virtualização utilizado foi o Xen (BARHAM et al., 2003). Dois *benchmarks* foram utilizados: TPC-H, em que cada SGBD (PostgreSQL e DB2) teve uma versão modificada desse *benchmark*, e uma carga de trabalho OLTP, que foi usada somente pelo DB2. Além disso, as VMs competiam por recursos na mesma máquina física. Os testes focaram apenas na alocação de CPU. A verificação de eficiência da abordagem proposta foi realizada comparando-se com uma distribuição igual de recursos entre as VMs. Foram realizados testes sobre a calibragem do otimizador de consultas e sobre o algoritmo de busca utilizado no trabalho, bem como experimentos sobre a precisão da solução quanto às necessidades de recursos de diferentes cargas de trabalho, sobre o suporte adequado à métrica de qualidade de serviço descrita em (SOROR et al., 2010), com cargas de trabalho aleatórias, e sobre o monitoramento e ajuste automático da configuração das VMs disponíveis.

7 CONCLUSÕES E TRABALHOS FUTUROS

A Computação em nuvem é uma tendência recente de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda, com pagamento baseado no uso. A nuvem computacional é um modelo de computação em que dados, arquivos e aplicações residem em servidores físicos ou virtuais, acessíveis por meio de uma rede em qualquer dispositivo compatível (fixo ou móvel), e que podem ser acessados a qualquer hora, de qualquer lugar, sem a necessidade de instalação ou configuração de programas específicos. Assim, a infraestrutura da computação em nuvem pode ser vista como um *pool* de recursos computacionais (virtualmente) infinito (e elástico) oferecido no modo *self-service*, por um terceiro, via um modelo "pague o quanto usa" (SOUSA et al., 2010).

As aplicações voltadas ao gerenciamento de dados são candidatas potenciais para a implantação em nuvem, uma vez que podem envolver grande complexidade e custos elevados. Neste cenário, o serviço de gerenciamento de dados assume a responsabilidade pela instalação, configuração e manutenção do sistema de banco de dados, bem como pelo acesso eficiente aos dados armazenados. O provedor do serviço de dados é o responsável por fornecer o serviço (incluindo toda infraestrutura de *hardware* e *software*), contabilizar a sua utilização, assegurar a sua qualidade, por meio de SLAs, e cobrar pelo seu uso.

Em um serviço de gerenciamento de dados em nuvem, os sistemas de gerenciamento de bancos de dados (SGBDs) são executados em máquinas virtuais e os dados estão distribuídos, particionados e/ou replicados, de forma a melhorar o desempenho, a escalabilidade e a disponibilidade do serviço. Além disso, a qualidade do serviço é assegurada por meio de SLAs. O SLA especifica o tempo de resposta máximo para cada consulta (ou a vazão esperada), além das penalidades (multas) a serem aplicadas ao provedor do serviço de dados caso os níveis de qualidade acordados não sejam atendidos.

Do ponto de vista do provedor do serviço de dados, há a necessidade de assegurar que o serviço de dados apresente um desempenho dentro dos limites acordados nos SLAs, mesmo na ocorrência de um aumento inesperado da demanda. Por este motivo, o provedor necessita monitorar o desempenho do serviço de dados e, caso o seu desempenho esteja inferior ao valor contratado (em termos de tempo de resposta ou vazão, por exemplo), ajustar a configuração da infraestrutura que compõe o serviço de dados com a finalidade de melhorar o seu desempenho, assegurando o atendimento dos SLAs e, consequentemente, a qualidade do serviço prestado. Esse ajuste pode ser realizado de diferentes maneiras, como por exemplo: ajuste nos parâmetros e configurações dos SGBDs que compõem o serviço, aumento da quantidade de máquinas virtuais que hospedam os SGBDs e as réplicas do banco de dados (e, consequentemente, possuem a capacidade de executar consultas), alteração dos recursos alocadas para as máquinas virtuais (CPU, memória, capacidade de armazenamento, etc.), dentre outros.

Neste trabalho, consideramos que um serviço de gerenciamento de dados em nuvem consiste de um sistema de banco de dados distribuído (SBDD) executado em máquinas virtuais (*Virtual Machines* - VMs) pré-configuradas e disponibilizadas por um provedor de infraestrutura como um serviço (*Infrastructure as a Service* - IaaS). Assim, cada máquina virtual hospeda uma

cópia do SGBD e uma réplica total do banco de dados. Mas, o conjunto de máquinas virtuais atua como um único sistema de banco de dados lógico.

Recentemente, diversas pesquisas têm sido realizadas com o objetivo de fornecer suporte ao gerenciamento de dados em nuvem. Contudo, a maioria dessas iniciativas busca solucionar problemas específicos, tais como: despacho (PATON et al., 2009), escalonamento (COSTA; FURTADO, 2008) e planejamento de capacidade (SHIVAM et al., 2007).

Este trabalho apresenta um *framework*, denominado GeDaNIC (SIQUEIRA; MONTEIRO; MACHADO, 2011), para o gerenciamento de sistemas de bancos de dados em nuvem. O *framework* proposto tem por objetivo fornecer a infraestrutura de *software* necessária para a disponibilização de serviços de dados em ambientes de computação em nuvem. Neste sentido, o mecanismo concebido busca solucionar alguns problemas ainda em aberto no contexto de sistemas de bancos de dados em nuvem, tais como: despacho, escalonamento de consultas e provisionamento de recursos.

A abordagem concebida estende os trabalhos anteriores adicionando importantes características, como: o suporte às cargas de trabalho imprevistas e a utilização de informações sobre as interações entre consultas. O suporte às cargas de trabalhos sazonais está relacionado a uma das principais propriedades da computação em nuvem: a elasticidade rápida. Já as interações entre consultas podem proporcionar impactos significativos no desempenho dos sistemas de bancos de dados. Por este motivo, o GeDaNIC utiliza informações sobre essas interações com a finalidade de reduzir o tempo de execução das cargas de trabalho submetidas ao serviço de dados e, consequentemente, aumentar o lucro do provedor deste serviço. Para isso, três novas abordagens para modelar e mensurar as interações entre instâncias e tipos de consultas são propostas (SIQUEIRA et al., 2012).

Em suma, este trabalho procurou demonstrar a veracidade das seguintes hipóteses: é possível modelar e medir a interação entre consultas; a modelagem da interação entre consultas pode ser utilizada em algoritmos de bancos de dados; a propriedade de interação entre consultas pode ser aplicada no gerenciamento de dados em nuvem. Para a primeira hipótese, as abordagens IS, DRR e GBA representam possibilidades de se modelar a interação entre consultas. Com relação à segunda hipótese, um escalonador orientado à interação entre consultas foi desenvolvido e validado por meio de testes, os quais comprovam seu potencial. Já para validar a terceira hipótese, o *framework* GeDaNIC foi desenvolvido e experimentos foram realizados utilizando-o. Este *framework* é orientado ao lucro, contudo a interação entre consultas é levada em consideração em suas tomadas de decisão.

7.1 Principais Contribuições

As principais contribuições desta dissertação foram:

- Três novas abordagens para modelar e mensurar as interações entre instâncias e tipos de consultas, as quais foram denominadas: *Intercalation Strategy* (IS), *Data Retrieving Rate* (DRR) e *Greedy with Bidimentional Array* (GBA). A interação entre duas instâncias

de consulta q_i e q_j é quantificada por meio de um novo conceito denominado fator de interação entre consultas. Assim, os algoritmos concebidos para implementar as abordagens propostas recebem como entrada duas instâncias de consulta q_i e q_j (ou os tipos de consulta dessas instâncias, denominados Q_{k_i} e Q_{k_j}, como no caso da abordagem GBA) e retornam como saída o fator de interação entre q_i e q_j.

- Um escalonador orientado à interação entre consultas, denominado ISO (*An Intelligent Scheduler for Multiple-query Execution Ordering*), voltado para a tomada de decisão em tempo real, mas que facilmente pode ser adaptado a cargas de trabalho em lote. Este escalonador mantém uma fila de escalonamento e utiliza o fator de interação entre consultas para definir a ordem de execução das instâncias de consulta. O escalonador ISO foi concebido e implementado com a finalidade de avaliar a utilização das abordagens propostas para modelar e medir as interações entre consultas (IS, DRR e GBA). Mais precisamente, o escalonador pode ser configurado para utilizar qualquer uma das soluções propostas: IS, DRR e GBA.

- Uma avaliação experimental do escalonador ISO e das abordagens IS, DRR e GBA. Testes de desempenho foram conduzidos para avaliar o comportamento do escalonador ao usar cada uma das três estratégias concebidas: IS, DRR e GBA. Os testes foram realizados utilizando-se o *benchmark* TPC-H sobre o PostgreSQL. Os resultados demonstram que as abordagens propostas (IS, DRR e GBA) têm potencial para aprimorar não só os algoritmos de escalonamento, mas muitos dos algoritmos de sintonia de banco de dados;

- Uma solução integrada para os problemas de despacho, escalonamento de consultas e provisionamento de recursos. Neste trabalho especificamos e implementamos um *framework*, denominado GeDaNIC, voltado para o gerenciamento de bancos de dados em nuvem. O *framework* proposto fornece a infraestrutura de *software* necessária para a disponibilização de serviços de dados em ambientes de computação em nuvem. Adicionalmente o GeDaNIC apresenta ainda seguintes características:

  1. Fornece suporte às cargas de trabalho imprevistas;
  2. Explora informações sobre as interações entre consultas.
  3. Não requerem suposições sobre aspectos internos do SGBD, fazendo com que GeDaNIC seja não intrusivo, ou seja, independente de SGBD.

- Uma avaliação experimental do *framework* proposto, comparando-o com os principais trabalhos anteriores. Os testes foram realizados utilizando-se o *benchmark* TPC-H sobre o PostgreSQL. Os resultados apontaram que a solução concebida tem potencial para aumentar o lucro do provedor do serviço de dados em nuvem, solucionando satisfatoriamente os problemas de despacho e escalonamento de consultas;

- Um levantamento do estado da arte na área de gerenciamento de dados em nuvem.

7.2 Oportunidades para Trabalhos Futuros

A seguir, escrevemos algumas oportunidades para trabalhos futuros:

- Conceber novas abordagens para modelar e medir as interações entre consultas, utilizando aprendizagem de máquina e computação bio-inspirada;
- Utilizar estatísticas para calcular o fator de interação global do serviço de dados. Atualmente é um valor fixo;
- Otimizar os algoritmos do módulo despachante a fim de possibilitar o despacho de consultas em lote;
- Investigar a configuração automática de VMs, que é um problema discutido em (SOROR et al., 2010) mas não avaliado o contexto de computação em nuvem;
- As avaliações experimentais mostraram que as abordagens propostas para modelar e medir as interações entre consultas (IS, DRR e GBA) têm potencial para melhorar muitos algoritmos de sintonia em bancos de dados. Logo, pode-se investigar a incorporação das abordagens propostas em outros componentes de SGBDs, tais como: otimizadores de consultas, ferramentas para ajuste do projeto físico de bancos de dados (*physical design advisors*) e gerenciadores de *buffers*, tornando-os orientados a interações entre consultas. Estas possibilidades representam possíveis direcionamentos para trabalhos futuros.
- Melhorar a modelagem e a medição do fator de interação entre consultas, por exemplo, levando em consideração a ordem de acesso às tabelas em cada consulta. Isso pode possibilitar um cálculo mais preciso para o fator de interação;
- Avaliar o comportamento do GeDaNIC em cenários de execução concorrente de consultas;
- Avaliar, a partir de experimentos, o módulo provisionador de recursos.

REFERÊNCIAS BIBLIOGRÁFICAS

ABADI, D. J. Data management in the cloud: Limitations and opportunities. *IEEE Data Engineering Bulletin*, v. 32, p. 3–12, 2009.

ABADI, D. J.; MADDEN, S. R.; HACHEM, N. Column-stores vs. row-stores: how different are they really? In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. Vancouver, Canada: SIGMOD, 2008. p. 967–980.

AHMAD, M.; ABOULNAGA, A.; BABU, S. Query interactions in database workloads. In: *Proceedings of the 2nd International Workshop on Testing Database Systems*. Rhode Island, USA: DBTest, 2009. p. 11:1–11:6.

AHMAD, M.; ABOULNAGA, A.; BABU, S.; MUNAGALA, K. Modeling and exploiting query interactions in database systems. In: *Proceedings of the 17th ACM Conference on Information and Knowledge Management*. California, USA: CIKM, 2008. p. 183–192.

AHMAD, M.; ABOULNAGA, A.; BABU, S.; MUNAGALA, K. Interaction-Aware Scheduling of Report-Generation Workloads. *The VLDB Journal*, v. 20, n. 4, p. 589–615, 2011.

BARHAM, P.; DRAGOVIC, B.; FRASER, K.; HAND, S.; HARRIS, T.; HO, A.; NEUGEBAUER, R.; PRATT, I.; WARFIELD, A. Xen and the art of virtualization. In: *Proceedings of the 19th ACM symposium on Operating systems principles*. New York, USA: SOSP, 2003. p. 164–177.

CHI, Y.; MOON, H. J.; HACIGÜMÜS, H.; TATEMURA, J. Sla-tree: a framework for efficiently supporting sla-based decisions in cloud computing. In: *Proceedings of the 14th International Conference on Extending Database Technology*. Uppsala, Sweden: EDBT, 2011. p. 129–140.

COMELLAS, J. O. F.; PRESA, I. G.; FERNÁNDEZ, J. G. Sla-driven elastic cloud hosting provider. In: *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. Pisa, Italy: PDP, 2010. p. 111–118.

COSTA, R. L. de C.; FURTADO, P. A qos-oriented external scheduler. In: *Proceedings of the 2008 ACM symposium on Applied computing*. Ceara, Brazil: SAC, 2008. p. 1029–1033.

CURINO, C.; JONES, E.; ZHANG, Y.; WU, E.; MADDEN, S. *Relational Cloud: The case for a database service*. 2010. `http://www-users.cselabs.umn.edu/classes/Fall-2012/csci8980-2/papers/relational.pdf`.

FERRETTI, S.; GHINI, V.; PANZIERI, F.; PELLEGRINI, M.; TURRINI, E. Qos-aware clouds. In: *3rd IEEE International Conference on Cloud Computing*. Florida, USA: IEEE CLOUD, 2010. p. 321–328.

FLORESCU, D.; KOSSMANN, D. Rethinking Cost and Performance of Database Systems. *SIGMOD Record*, v. 38, n. 1, p. 43–48, 2009.

GUPTA, C.; MEHTA, A.; WANG, S.; DAYAL, U. Fair, effective, efficient and differentiated scheduling in an enterprise data warehouse. In: *Proceedings of the 12th International Conference on Extending Database Technology*. Saint Petersburg, Russia: EDBT, 2009. p. 696–707.

IQBAL, W.; DAILEY, M. N.; CARRERA, D.; JANECEK, P. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, v. 27, n. 6, p. 871–879, 2011.

LSCR. *SLA for database projects*. 2011. `http://lscr.berkeley.edu/rates/sla/database.php`.

MALKOWSKI, S.; HEDWIG, M.; JAYASINGHE, D.; PU, C.; NEUMANN, D. Cloudxplor: a tool for configuration planning in clouds based on empirical data. In: *Proceedings of the 2010 ACM symposium on Applied computing*. Sierre, Switzerland: SAC, 2010. p. 391–398.

MAZZUCCO, M. Towards autonomic service provisioning systems. In: *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. Victoria, Australia: CCGRID, 2010. p. 273–282.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. 2009. `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`.

NIU, B.; MARTIN, P.; POWLEY, W. Towards Autonomic Workload Management in DBMSs. *Journal of Database Management*, v. 20, n. 3, p. 1–17, 2009.

NIU, B.; MARTIN, P.; POWLEY, W.; BIRD, P.; HORMAN, R. Adapting mixed workloads to meet slos in autonomic dbmss. In: *Proceedings of the IEEE 23rd International Conference on Data Engineering Workshop*. Istanbul, Turkey: ICDE, 2007. p. 478–484.

O'GORMAN, K.; ABBADI, A. E. E.; AGRAWAL, D. Multiple Query Optimization in Middleware Using Query Teamwork. *Software - Practice & Experience*, v. 35, n. 4, p. 361–391, 2005.

OPENCLOUD. *The Open Could Manifesto*. 2010. `http://www.opencloudmanifesto.org`.

PATON, N. W.; ARAGÃO, M. A. T.; LEE, K.; FERNANDES, A. A. A.; SAKELLARIOU, R. Optimizing utility in cloud computing through autonomic workload execution. *IEEE Data Engineering Bulletin*, v. 32, n. 1, p. 51–58, 2009.

POPOVICI, F. I.; WILKES, J. Profitable services in an uncertain world. In: *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. Washington, USA: SC, 2005. p. 36.

ROGERS, J.; PAPAEMMANOUIL, O.; ÇETINTEMEL, U. A generic auto-provisioning framework for cloud databases. In: *Proceedings of the IEEE 26th International Conference on Data Engineering Workshop*. California, USA: ICDE, 2010. p. 63–68.

ROY, P.; SESHADRI, S.; SUDARSHAN, S.; BHOBE, S. Efficient and Extensible Algorithms for Multi Query Optimization. *SIGMOD Record*, v. 29, n. 2, p. 249–260, 2000.

SAKR, S.; LIU, A. Sla-based and consumer-centric dynamic provisioning for cloud databases. In: *5th IEEE International Conference on Cloud Computing*. Hawaii, USA: IEEE Cloud, 2012. p. 360–367.

SCHAD, J.; DITTRICH, J.; QUIANÉ-RUIZ, J.-A. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceedings of the VLDB Endowment*, v. 3, n. 1, p. 460–471, 2010.

SCHNJAKIN, M.; ALNEMR, R.; MEINEL, C. Contract-based cloud architecture. In: *Proceedings of the 2nd international workshop on Cloud data management*. Toronto, Canada: CloudDB, 2010. p. 33–40.

SCHROEDER, B.; HARCHOL-BALTER, M. Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness. v. 7, n. 2, p. 151–161, 2004.

SHARAF, M. A.; CHRYSANTHIS, P. K.; LABRINIDIS, A.; AMZA, C. Optimizing i/o-intensive transactions in highly interactive applications. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. Rhode Island, USA: SIGMOD, 2009. p. 785–798.

SHARMA, U.; SHENOY, P.; TOWSLEY, D. F. Provisioning multi-tier cloud applications using statistical bounds on sojourn time. In: *Proceedings of the 9th international conference on Autonomic computing*. California, USA: ICAC, 2012. p. 43–52.

SHIVAM, P.; DEMBEREL, A.; GUNDA, P.; IRWIN, D.; GRIT, L.; YUMEREFENDI, A.; BABU, S.; CHASE, J. Automated and on-demand provisioning of virtual machines for database applications. In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. Beijing, China: SIGMOD, 2007. p. 1079–1081.

SILVA, T. L. C.; NASCIMENTO, M. A.; MACêDO, J. A. F.; SOUSA, F. R. C.; MACHADO, J. C. Towards non-intrusive elastic query processing in the cloud. In: *Proceedings of the 4th International Workshop on Cloud Data Management*. Hawaii, USA: CloudDB, 2012. p. 9–16.

SIQUEIRA, M.; MONTEIRO, J. M.; MACEDO, J.; MACHADO, J. de C. Approaches to model query interactions. In: *Proceedings of the 27th Brazilian Symposium on Databases (Short Paper)*. São Paulo, Brazil: SBBD, 2012.

SIQUEIRA, M.; MONTEIRO, J. M.; MACHADO, J. de C. Um *Framework* para o Gerenciamento Autonômico de Bancos de Dados em Nuvem Baseado nas Interações entre Consultas. In: *Workshop de Teses e Dissertações em Banco de Dados*. Florianópolis, Brazil: SBBD, 2011.

SOROR, A. A.; MINHAS, U. F.; ABOULNAGA, A.; SALEM, K.; KOKOSIELIS, P.; KAMATH, S. Automatic virtual machine configuration for database workloads. *ACM Transactions on Database Systems*, v. 35, n. 1, p. 1–47, 2010.

SOUSA, F. R. C.; MOREIRA, L. O.; MACÊDO, J. A. F.; MACHADO, J. C. *Gerenciamento de Dados em Nuvem: Conceitos, Sistemas e Desafios*. In: PEREIRA, A. C. M.; PAPPA, G. L.; WINCKLER, M.; GOMES, R. L. (Org.). Tópicos em Sistemas Colaborativos, Interativos, Multimídia, Web e Bancos de Dados, SIWB 2010, 1. ed. Belo Horizonte: SBC, 2010. 101-130 p.

SOUSA, F. R. C.; MOREIRA, L. O.; SANTOS, G. A. C.; MACHADO, J. C. Quality of service for database in the cloud. In: *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*. Porto, Portugal: CLOSER, 2012. p. 595–601.

TAN, K.-L.; LU, H. Workload scheduling for multiple query processing. *Information Processing Letters*, v. 55, n. 5, p. 251–257, 1995.

TILGNER, C. Declarative scheduling in highly scalable systems. In: *Proceedings of the 2010 EDBT/ICDT Workshops*. Lausanne, Switzerland: EDBT, 2010. p. 41:1–41:6.

TPC. *TPC-H Benchmark*. 2012. `http://www.tpc.org/tpch/spec/tpch2.14.4.pdf`.

WU, L.; GARG, S. K.; BUYYA, R. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In: *Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. California, USA: CCGRID, 2011. p. 195–204.

XIONG, P.; CHI, Y.; ZHU, S.; TATEMURA, J.; PU, C.; HACIGUMUS, H. Activesla: a profit-oriented admission control framework for database-as-a-service providers. In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. Cascais, Portugal: SOCC, 2011. p. 15:1–15:14.

XIONG, P.; CHI, Y.; ZHU, S.; MOON, H. J.; PU, C.; HACIGUMUS, H. Intelligent management of virtualized resources for database systems in cloud environment. In: *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*. Hannover, Germany: ICDE, 2011. p. 87–98.

YANG, F.; SHANMUGASUNDARAM, J.; YERNENI, R. A scalable data platform for a large number of small applications. In: *4th Biennial Conference on Innovative Data Systems Research*. California, USA: CIDR, 2009. p. 1–10.