UNIVERSIDADE FEDERAL DO CEARÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

MANOEL RUI PESSOA DE PAULA

BACOS: UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA PARA SISTEMAS DE ARMAZENAMENTO DE OBJETOS EM NUVEM

FORTALEZA

2017

MANOEL RUI PESSOA DE PAULA

BACOS: UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA PARA SISTEMAS DE ARMAZENAMENTO DE OBJETOS EM NUVEM

Dissertação apresentada ao Curso de Mestrado em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Departamento de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Banco de Dados

Orientador: Prof. Dr. Javam de Castro Machado

FORTALEZA

2017

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

P568b Pessoa de Paula, Manoel Rui.
BACOS : UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA PARA SISTEMAS DE ARMAZENAMENTO DE OBJETOS EM NUVEM / Manoel Rui Pessoa de Paula. – 2017.
111 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2017.
Orientação: Prof. Dr. Javam de Castro Machado.

1. Computação em nuvem. 2. Armazenamento de objetos. 3. Balanceamento de carga. 4. Dispositivo de armazenamento heterogêneo. I. Título.

CDD 005

MANOEL RUI PESSOA DE PAULA

BACOS: UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA PARA SISTEMAS DE ARMAZENAMENTO DE OBJETOS EM NUVEM

Dissertação apresentada ao Curso de Mestrado em Ciência da Computação do Programa de Pós-Graduação em Ciência da Computação do Departamento de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Banco de Dados

BANCA EXAMINADORA

Prof. Dr. Javam de Castro Machado (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Flávio Rubens de Carvalho Sousa
Universidade Federal do Ceará (UFC)

Prof. Dr. José Maria da Silva Monteiro Filho
Universidade Federal do Ceará (UFC)

Prof. Dra. Carmem Satie Hara
Universidade Federal do Paraná (UFPR)

À minha família, por sempre ter acreditado em mim.

AGRADECIMENTOS

Aos meus pais, Estevão Pessoa de Paula e Maria José de Sousa de Paula, pelo suporte e motivação para seguir em frente nos momentos mais importantes da minha vida.

Ao meu orientador e mentor, Javam Machado, pelos conselhos, orientações e oportunidades concedidas que foram fundamentais para o sucesso desta dissertação.

Aos professores Flávio Rubens de Sousa, José Maria Monteiro e a professora Carmem Hara pela disponibilidade de participar da minha banca de mestrado e pelas valiosas sugestões no aprimoramento desta dissertação.

Ao Laboratório de Sistemas e Banco de Dados (LSBD) por ter fornecido toda a estrutura necessária para o desenvolvimento da pesquisa científica contida na minha dissertação e auxílio financeiro ao custear publicações e eventos científicos.

Aos meus amigos Victor Farias, Denis Cavalcante e Flávio Rubens de Sousa pelo apoio contínuo e a paciência durante as inúmeras discussões envolvendo computação e nuvem e balanceamento de carga no LSBD.

A todos os meus amigos colaboradores do *Advanced Research in Database* (ARiDa) e do LSBD pela ajuda em diversas situações que de uma forma direta ou indireta contribuíram para construção da minha pesquisa científica.

E à Fundação Cearense de Apoio ao Desenvolvimento (Funcap) pelo financiamento da pesquisa de mestrado via bolsa de estudos.

“Se você não sabe onde quer ir, qualquer caminho serve.”

(Lewis Carroll)

RESUMO

Computação em nuvem é um modelo de computação emergente e eficiente para processar e armazenar grandes quantidades de dados. Provedores de serviços de armazenamento em nuvem utilizam dispositivos de armazenamento heterogêneos como forma de ampliar recursos de infraestrutura de um sistema de armazenamento considerando a melhor compensação entre os gastos de manutenção e desempenho. Os sistemas de armazenamento de objetos em nuvem surgem como soluções escaláveis e eficientes para gerenciar dados usando dispositivos heterogêneos, no que diz respeito à capacidade de armazenamento e ao desempenho. Na nuvem, uma vez que a carga de trabalho muda dinamicamente, a atribuição dinâmica da carga de trabalho para as capacidades dos dispositivos de armazenamento é feita para melhorar a utilização dos recursos e otimizar o desempenho geral de um sistema de armazenamento de objetos. Assim, as técnicas de balanceamento de carga são cruciais para redistribuir a carga de trabalho entre os nós de processamento e armazenamento para evitar subutilização ou sobrecarga de recursos. A maioria das estratégias convencionais de balanceamento de carga em sistemas de armazenamento em nuvem priorizam o espaço de armazenamento ao invés do desempenho dos dispositivos de armazenamento ou fazem suposições sobre os dispositivos serem homogêneos, resultando na degradação do desempenho do sistema de armazenamento. Para atender essas limitações, este trabalho apresenta uma estratégia de balanceamento de carga não intrusiva denominada BACOS. A estratégia BACOS tira proveito da heterogeneidade dos dispositivos de armazenamento pertencente a um sistema de armazenamento de objetos na nuvem com o intuito de melhorar o desempenho geral do sistema. Os resultados da avaliação experimental confirmam que a estratégia BACOS consegue melhorar o desempenho de um sistema de armazenamento de objetos em termos de tempo de resposta, vazão e taxa de sucesso de requisições do tipo leitura/escrita em cenários variando a carga de trabalho.

Palavras-chave: Computação em nuvem. Armazenamento de objetos. Balanceamento de carga. Dispositivo de armazenamento heterogêneo.

ABSTRACT

Cloud computing is an emerging and efficient computing model for processing and storing large amounts of data. Cloud storage service providers use heterogeneous storage devices as a way to extend the resources of a storage system by considering the best tradeoff between maintenance and performance costs. Cloud object storage systems emerge as scalable solutions and efficient data managers using heterogeneous devices, in terms of storage capacity and performance. In the cloud, as the workload changes, dynamic matching between load and storage device capabilities is needed to improve resource utilization and optimize the overall performance of an object storage system. Thus, load balancing techniques are crucial for redistributing the workload among processing and storage nodes to avoid underloading or overloading. Most conventional load balancing strategies in cloud storage systems are only aware of the storage capacity of storage devices or make assumptions about them being homogeneous, resulting in degradation of the storage system. To address these limitations, this work presents a non-intrusive load balancing strategy called BACOS that takes advantage of storage devices with heterogeneous performance in a cloud storage system. The results of the experimental evaluation confirm that BACOS can improve the performance of an object storage system in terms of response time, throughput and success rate of read/write requests in scenarios that vary the workload.

Keywords: Cloud Computing. Object storage. Load balancing. Heterogeneous storage device

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SÍMBOLOS

C	Classe de desempenho
T_C	Valor limiar de IOPS de uma classe de desempenho
w	Peso de uma classe de desempenho
W	Conjunto dos pesos das classes de desempenho
d	Dispositivo de armazenamento
N	Número de clientes
R	Taxa de leitura
RT	Tempo de Resposta
RT_{max}	Tempo de Resposta máximo
RT_{min}	Tempo de Resposta mínimo
REQ_{total}	Número total de requisições
REQ_{error}	Número total de requisições com erros
I	Taxa de insucesso das requisições
I_{mean}	Taxa de insucesso médio das requisições
W_{opt}	Conjunto de pesos ótimos para classes de desempenho
α	Fator de migração
F	Função de custo
L_W	Lista dos conjuntos dos pesos das classes de desempenho para uma carga (N, RW)
T	Tabela de consulta rápida dos W_{opts}
t	Instante de tempo

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

Computação em nuvem é um paradigma de grande sucesso para serviços de processamento distribuído e gerenciamento de dados. Aplicações modernas, orientadas a dados na nuvem, demandam grandes quantidades de recursos devido à taxa crescente e contínua de dados produzidos. Segundo o *International Data Corporation (IDC)* (ZWOLENSKI *et al.*, 2014), o montante de dados digitais produzidos ao redor do mundo cresce mais do que o dobro a cada dois anos. O IDC estima que em 2020, o universo digital irá atingir cerca de 44 trilhões de gigabytes de informação. Nesse cenário, sistemas de armazenamento em larga escala são componentes fundamentais na camada de serviços em computação em nuvem.

Sistemas de armazenamento de objetos – *object storages* – emergiram como uma tecnologia de armazenamento para dar suporte aos vários requisitos das aplicações em nuvem. *Object storages* são sistemas de armazenamento que possuem arquitetura escalável e são compostos por servidores distribuídos e dispositivos de armazenamento chamados de *Object Storage Devices (OSDs)*. Servidores e OSDs são encarregados por armazenar e recuperar informações dos usuários por meio de redes de computadores (MESNIER *et al.*, 2005). Os *object storages* expõem interfaces de alto nível capaz de abstrair a complexidade de comunicação entre aplicações e a camada de persistência de dados que são encapsuladas pelos OSDs. Essas interfaces de alto nível de abstração são utilizados por programas clientes dos sistemas de armazenamento para executar requisições de leitura e escrita sobre dados no formato de objeto. Objetos são unidades de armazenamento que encapsulam dados e metadados com tamanho variável. Devido a natureza escalável e a facilidade de gerenciamento dos dados e metadados dos objetos, eles são bastante úteis para armazenar dados não estruturados e multimídia como filmes, músicas, imagens, *logs* e *backups*.

Dispositivos de armazenamento de baixo custo, chamados comumente de *commodity devices*, como discos rígidos *(HDDs)* são amplamente utilizados por *object storages* em nuvem como solução básica de persistência em *hardware*. Dessa forma, torna-se fácil estender a capacidade total do espaço de armazenamento do sistema com custos reduzidos. Em geral, dispositivos de armazenamento de baixo custo tendem a apresentar falhas com bastante frequência (XIN *et al.*, 2003), exigindo trocas frequentes. Portanto, propriedades de durabilidade e disponibilidade dos dados são alcançadas por meio de mecanismos de replicação com dispositivos de baixo custo

fornecidos pelos *object storages* em nuvem (HONICKY; MILLER, 2004). Porém, esse tipo de dispositivo pode se tornar um gargalo computacional para um sistema de armazenamento devido ao seu baixo desempenho (GUNAWI *et al.*, 2005). Com o objetivo de melhorar o desempenho geral de um sistema de armazenamento em termos de vazão e latência na camada de armazenamento, provedores de serviços utilizam em sua infraestrutura de nuvem, em conjunto com os *commodity devices*, dispositivos de alto desempenho como discos de estado sólido *(SSDs)*. Embora esse tipo de dispositivo proporcione ótimo desempenho e eficiência no gerenciamento de energia (WANG *et al.*, 2014), eles possuem um alto custo de aquisição e capacidade de armazenamento reduzida.

Assim, não é interessante para um provedor de serviço manter uma infraestrutura de armazenamento formada apenas por dispositivos de alto desempenho, devido à relação custo financeiro e capacidade de armazenamento. A utilização contínua em larga escala desse tipo de dispositivo também implica em substituições frequentes. Além disso, soluções com arquiteturas heterogêneas mostram que a combinação dos dois tipos de dispositivos de armazenamento conseguem fornecer um interessante custo benefício em relação aos custos de manutenção e desempenho do sistema (XU *et al.*, 2015). *Object storages* comerciais populares utilizados em nuvem como *Ceph* (WEIL *et al.*, 2006a), *OpenStack Swift* (ARNOLD, 2014) e *Dynamo* (DECANDIA *et al.*, 2007), são projetados principalmente para armazenar grande volume de dados sobre um conjunto de dispositivos baseados em sua capacidade de armazenamento.

A computação em nuvem é caracterizada por ser um ambiente de natureza dinâmica. Normalmente, em um sistema de armazenamento em nuvem como um *object storage*, os usuários conseguem realizar requisições de leitura e escrita de dados de forma paralela. À medida que o *object storage* conclui as requisições dos usuários, o número de requisições tende a aumentar pois novas requisições são geradas pelos usuários e submetidas ao sistema de armazenamento. Assim, cada vez mais é exigido do desempenho do sistema de armazenamento em termos de vazão e tempo de resposta para atender a demanda das requisições. Além disso, outros componentes da carga de trabalho como o padrão de leitura e escrita podem necessitar mais recursos de desempenho dos dispositivos de armazenamento (HOU *et al.*, 2016). A dificuldade de prever a carga de trabalho que é submetida ao sistema de armazenamento torna o problema de atribuir a carga aos recursos disponíveis dos dispositivos de armazenamento da melhor forma possível não trivial. Em particular, esse problema se torna ainda mais desafiador em sistemas de armazenamento heterogêneos compostos por dispositivos de armazenamento que apresentam

desempenhos distintos, por ser difícil estimar e atribuir a quantidade ideal de carga para cada dispositivo de armazenamento dinamicamente.

Dessa forma, o problema de balanceamento de carga em sistemas de armazenamento surge quando o sistema de armazenamento precisa lidar com diferentes tipos de requisições. Eventualmente, alguns dispositivos de armazenamento podem não conseguir gerenciar de maneira eficiente algumas requisições por conta da limitação de recursos de armazenamento e desempenho dos dispositivos de armazenamento, enquanto que naquele exato momento, outros dispositivos podem apresentar recursos disponíveis capaz de satisfazer tal demanda. Logo, o principal desafio está em evitar a sobrecarga ou a subutilização de recursos de armazenamento e desempenho com o objetivo de melhorar a utilização dos recursos e o desempenho geral do sistema de armazenamento (NUAIMI *et al.*, 2012).

Algumas estratégias de balanceamento de carga, tais como as descritas em (YANG *et al.*, 2015) e (HSIAO *et al.*, 2013) rebalanceiam os dados armazenados no sistema de armazenamento por meio de esquemas de replicação e migração de dados entre os dispositivos de armazenamento, reajustando os recursos subutilizados ou sobrecarregados enquanto garante a disponibilidade das informações. Outras estratégias como as propostas em (WANG *et al.*, 2015) e (TAN *et al.*, 2013), além de utilizarem técnicas de migração e replicação, caracterizam a carga de trabalho de forma particular para fortalecer a decisão de balanceamento e ajustar a carga as capacidades do sistema, seja por meio dos recursos de rede, processamento, memória e armazenamento. Contudo, estas e outras estratégias na literatura não consideram os aspectos de desempenho dos dispositivos de armazenamento ou fazem suposições que eles são homogêneos. Então, estratégias de balanceamento que se baseiam na variação dinâmica da carga e tiram proveito dos recursos heterogêneos surgem como mecanismos eficientes para melhorar a utilização dos recursos e otimizar o desempenho geral dos sistemas de armazenamento em nuvem.

1.2 Objetivos

1.2.1 Objetivo geral

O objetivo geral desse trabalho é desenvolver uma estratégia de balanceamento de carga para *object storages* em ambiente de nuvem, composto por dispositivos de armazenamento com desempenho e capacidade de armazenamento heterogêneos que seja capaz de melhorar o desempenho geral do sistema.

1.2.2 Objetivos específicos

Visando atingir este objetivo geral, podem ser relacionados os seguintes objetivos específicos:

- Realizar um levantamento do estado da arte sobre as principais estratégias de balanceamento de carga em sistemas de armazenamento em nuvem e analisar seus pontos positivos e negativos.
- Propor uma estratégia de balanceamento de carga para *object storages* baseado na caracterização da carga de trabalho e heterogeneidade dos dispositivos de armazenamento.
- Avaliar a efetividade da estratégia de balanceamento de carga em um *object storage* por meio de experimentos práticos em um ambiente de teste para computação em nuvem.

1.3 Contribuições

Como o principal resultado dessa dissertação, tem-se as seguintes contribuições:

- Uma estratégia de balanceamento de carga adequada para *object storages* compostos por dispositivos de armazenamento com desempenho distintos a qual não é necessário alterar o código fonte interno do sistema de armazenamento em sua implantação (não intrusivo).
- Uma estratégia dinâmica de balanceamento de carga capaz de se adaptar a determinados padrões de carga compostos pela taxa de leitura e escrita e a quantidade de requisições.
- Implantação e avaliação experimental em um *object storage* popular utilizado no ambiente de nuvem.

1.3.1 Produção científica

As contribuições científicas apresentadas neste trabalho possibilitou a seguinte publicação:

- PAULA, M. R. P.; RODRIGUES, E.; FARIAS, V. A. E.; SOUSA, F. R. C.; MACHADO, J. C. BACOS: A dynamic load balancing strategy for cloud object storage. In: Simpósio Brasileiro de Redes de Computadores e Sistemas

Distribuídos (SBRC), 2017 (PAULA *et al.*, 2017).

1.4 Estrutura da dissertação

Esta dissertação está organizada da seguinte forma: No Capítulo 2 são apresentados os conceitos e as definições fundamentais sobre sistemas de armazenamento, alocação de dados e os principais desafios relacionados ao problema de balanceamento de carga no ambiente de computação em nuvem. O Capítulo 3 descreve e discute os trabalhos relacionados mais relevantes da literatura. Em seguida, o Capítulo 4 apresenta a estratégia proposta neste trabalho para solucionar o problema de balanceamento de carga em *object storages*, chamada BACOS. O Capítulo 5 apresenta a validação da estratégia de balanceamento BACOS. Esse capítulo mostra os resultados obtidos por um conjunto de experimentos realizados em um ambiente de nuvem considerando diferentes cenários e métricas de desempenho. Finalmente, o Capítulo 6 conclui o trabalho, apresentando um resumo dos resultados alcançados e apontando as principais direções para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Computação em nuvem

Computação em nuvem vem se consolidando ao longo dos anos como um modelo de computação eficiente na indústria da Tecnologia da Informação (TI). As aplicações ficaram muito mais atrativas como serviço, fabricantes mudaram a maneira como os *hardwares* são produzidos e como provedores de serviços otimizam a utilização dos componentes de *hardware*. A nuvem computacional é uma abstração referente às aplicações entregues como serviços via *internet* e toda a infraestrutura de *hardware* e sistema de *software* em centro de dados (*data centers*) capaz de fornecer tais serviços. Além disso, as aplicações podem ser acessadas sob demanda e em qualquer lugar (ARMBRUST *et al.*, 2010).

Trabalhos como (VAQUERO *et al.*, 2008) buscam acompanhar e entender a evolução da TI, ao longo do desenvolvimento das tecnologias de computação em *cluster* e *grid*, com o objetivo de definir computação em nuvem. Contudo, a escolha da definição do *National Institute of Standard and Technology (NIST)* é razoável por ser o instituto responsável por definir padrões de TI para a indústria americana (MELL *et al.*, 2011). Ademais, a definição do NIST se aproxima mais dos objetivos deste trabalho. Segundo o NIST, computação em nuvem é um paradigma em evolução e é definida da seguinte maneira: *Computação em nuvem é um modelo capaz de prover acesso pela rede de forma ubíqua, conveniente e sob demanda a um conjunto compartilhado de recursos computacionais configuráveis (por exemplo, rede, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com o mínimo esforço de gestão ou iteração com o provedor de serviço*. A definição do NIST engloba características essenciais como meio de comparação entre os diversos modelos de serviços e estratégias de implantação em centro de dados.

2.1.1 Características essenciais

A computação em nuvem é formada por um conjunto de características essenciais capazes de prover soluções eficientes e de baixo custo. As principais características de computação em nuvem são:

- ***Self-service* sob demanda.** Um determinado cliente pode provisionar unilateralmente recursos computacionais, tais como tempo de processamento e armazenamento via rede, quando necessário sem a necessidade de intervenção humana com os provedores de

serviços.

- **Amplo acesso pela rede**. O recursos estão disponíveis pela rede e podem ser acessados através de mecanismos padronizados que suportam o uso de plataformas clientes heterogêneas. O acesso pode ser feito tanto por plataformas clientes com alta capacidade de processamento quanto clientes leves com baixa capacidade de processamento. Por exemplo, *smartphones*, *tablets*, *notebooks* e *laptops* (TOLIA *et al.*, 2006).
- **Conjunto (*pool*) de recursos**. Os recursos computacionais de um provedor são agrupados para servir múltiplos clientes seguindo um modelo multi-inquilinos (*multi-tenant*). Diferentes recursos físicos e virtuais são dinamicamente atribuídos e reatribuídos de acordo com a demanda dos clientes. Em geral, estes não possuem o controle e desconhecem a localização física exata dos recursos do provedor, mas eles usufruem da possibilidade de especificar a localização em um alto nível de abstração (por exemplo: país, estado ou centro de dados). O armazenamento, processamento, memória e largura de banda de rede são exemplos de recursos que seguem o conceito e abstração da localização física.
- **Elasticidade rápida**. Os recursos computacionais podem ser elásticos, ou seja, eles podem ser provisionados e liberados. O interessante é que esse processo seja feito de forma automática para escalar os recursos rapidamente a medida que a demanda aumenta, e liberá-los à medida que a demanda diminui. Do ponto de vista dos clientes, os recursos computacionais disponíveis para provisionamento, normalmente, aparentam ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento.
- **Medição do serviço**. Sistemas de computação em nuvem automaticamente controlam e otimizam a utilização dos recursos por meio da capacidade de medição do sistema sobre algum nível de abstração apropriada para o tipo de serviço. Armazenamento, processamento, largura de banda e número de clientes ativos são exemplos de abstrações. A utilização dos recursos pode ser monitorada, controlada e reportada com o propósito de fornecer transparência para provedores e clientes que utilizam um determinado serviço. Esse tipo de característica é útil para modelos de serviços em que o cliente paga apenas o que ele utiliza (*pay-per-use*) (GONG *et al.*, 2010).

2.1.2 Modelos de serviço

A computação em nuvem implementa três tipos de serviços que contribuem para construir modelos orientados ao negócio para provedores de serviços. A nuvem pode oferecer

serviços que podem ser agrupados em uma das três categorias:

- ***Infrastructure as a Service (IaaS)***. Esse tipo de serviço refere-se ao provisionamento de recursos da infraestrutura. A IaaS facilita provedores provisionarem recursos de processamento, armazenamento, rede e outros recursos fundamentais para que o cliente construa um ambiente sob demanda capaz de executar suas aplicações, onde é possível incluir sistemas operacionais e aplicações customizadas. Os usuários não conseguem gerenciar ou controlar os recursos físicos da infraestrutura, mas possuem controle sobre sistemas operacionais, armazenamento e aplicações disponíveis pelo provedor. Eventualmente, os clientes possuem controles limitados ao selecionar componentes de rede como *firewalls*. O avanço da tecnologia de virtualização viabilizou o uso de dispositivos físicos de baixo custo (*physical machines PMs*) como uma solução barata capaz de executar aplicações e compartilhar recursos de forma eficiente no ambiente de nuvem através de máquinas virtuais (*virtual machines VMs*). As VMs conseguem isolar aplicações da camada física e ainda manter a flexibilidade de se ajustarem às necessidades dos usuários finais (BUYYA *et al.*, 2009). *Amazon EC2* (AMAZON, 2017b) e *FlexiScale* (FLEXISCALE, 2017) são exemplos de IaaS.
- ***Platform as a Service (PaaS)***. O modelo PaaS fornece ao cliente a facilidade de criar novas aplicações, usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. De forma geral, o usuário não gerencia ou controla a camada inferior da infraestrutura da nuvem como rede, servidores, sistemas operacionais ou dispositivos de armazenamento. Porém, o usuário tem controle sobre algumas aplicações da infraestrutura e com possibilidade de configurar parâmetros das aplicações hospedadas nela. Como exemplos de PaaS pode-se destacar o *Google App Engine* (GOOGLE, 2017a) e *Microsoft Azure* (MICROSOFT, 2017).
- ***Software as a Service (SaaS)***. O modelo SaaS refere-se a disponibilidade de aplicações sob demanda pela rede fornecida pelo provedor de serviço. A aplicações são acessíveis por vários tipos de dispositivos dos usuários através de uma interface cliente leve tal como um navegador de internet ou um programa com interface similar. Os clientes não gerenciam ou controlam a camada da infraestrutura da nuvem, incluindo a rede, servidores, sistemas operacionais, armazenamento ou mesmo parâmetros individuais da aplicação. Dificilmente, o cliente possui a possibilidade de modificar parâmetros de configuração internos das aplicações e, quando possível, a configuração é limitada. Exemplos de provedores de SaaS

incluem *Salesforce* (SALESFORCE, 2017), *Google Docs* (GOOGLE, 2017b) e *Dropbox* (DROPBOX, 2017).

2.1.3 Modelos de implantação

Diferentes tipos de modelos de implantação estão presentes no ambiente de computação em nuvem quanto a restrição de acesso e disponibilidade das informações em domínios administrativos. Cada modelo de implantação apresenta características particulares baseadas na natureza do negócio, tipo de informação e gerenciamento da infra estrutura de nuvem. Destaca-se os seguintes modelos de implantação:

- **Nuvem privada.** A infraestrutura de nuvem é utilizada para uso exclusivo por parte de uma organização. A nuvem pode ser gerenciada pela própria organização, terceiros ou uma combinação de ambos.
- **Nuvem comunitária.** A infraestrutura de nuvem é utilizada por comunidades específicas ou clientes de organizações que desejam compartilhar informações de caráter consensual. A nuvem pode ser mantida pela comunidade, terceiros ou uma combinação de ambos.
- **Nuvem pública.** A infraestrutura de nuvem é utilizada para uso público e com propósitos gerais. A nuvem pode ser gerenciada e mantida por uma empresa, organização governamental, acadêmica ou uma combinação delas.
- **Nuvem Híbrida.** A infraestrutura de nuvem é composta por duas ou mais infraestruturas de nuvem distintas que permanecem como entidades únicas, mas são conectadas por padrões e tecnologias proprietárias que permitem a portabilidade de aplicações e dados.

2.2 Sistemas de armazenamento em nuvem *(cloud storages)*

Um sistema de armazenamento em nuvem (*Cloud Storage*) é um conceito que veio acompanhado com o modelo de computação em nuvem como uma solução de baixo custo para atender a crescente necessidade das aplicações por armazenamento massivo e de alto desempenho. Tradicionalmente, Sistema de Banco de Dados (SGBDs) tem sido utilizados pela maioria das aplicações empresariais como modelo para armazenamento e recuperação de dados. Apesar de SGBDs serem ricos em termos de consultas, a complexidade inserida pelo seu modelo de acesso juntamente com inflexibilidade de escalar a infraestrutura, formam um gargalo para aplicações que precisam de uma solução de armazenamento simples e rápida, capaz de suportar grande

volume de dados (DEWAN; HANSDAH, 2011). Atualmente, banco de dados relacionais e outras formas de armazenar dados estruturados ainda são utilizados em negócios. Entretanto, o crescimento de dados não estruturados vem ocorrendo rapidamente ao longo dos anos. Segundo um estudo realizado pela *International Data Corporation* (IDC) em 2014, o universo digital no período de 2013 a 2020 irá crescer de 4.4 trilhões de gigabytes para 44 trilhões, o que representa mais que o dobro a cada dois anos (ZWOLENSKI *et al.*, 2014).

Um sistema de armazenamento em nuvem é compreendido como um serviço capaz de prover armazenamento para usuários e clientes por intermédio da rede, usualmente a *internet*. Os usuários podem usar o serviço de armazenamento de diferentes maneiras, pagar apenas pelo tempo de uso, espaço de armazenamento, desempenho ou uma combinação dessas formas de pagamento (WU *et al.*, 2010). Um sistema de armazenamento em nuvem ainda pode ser compreendido simplesmente como a parte de armazenamento de uma nuvem computacional. Aplicações consomem serviços de armazenamento da nuvem computacional baseado no *pool* de recursos de armazenamento. Um *pool* de recursos de armazenamento pode ser virtualizado para servir aplicações de dentro ou fora da nuvem computacional,mas nem toda a parte do armazenamento físico pode ser desacoplada da nuvem computacional (JU *et al.*, 2011).

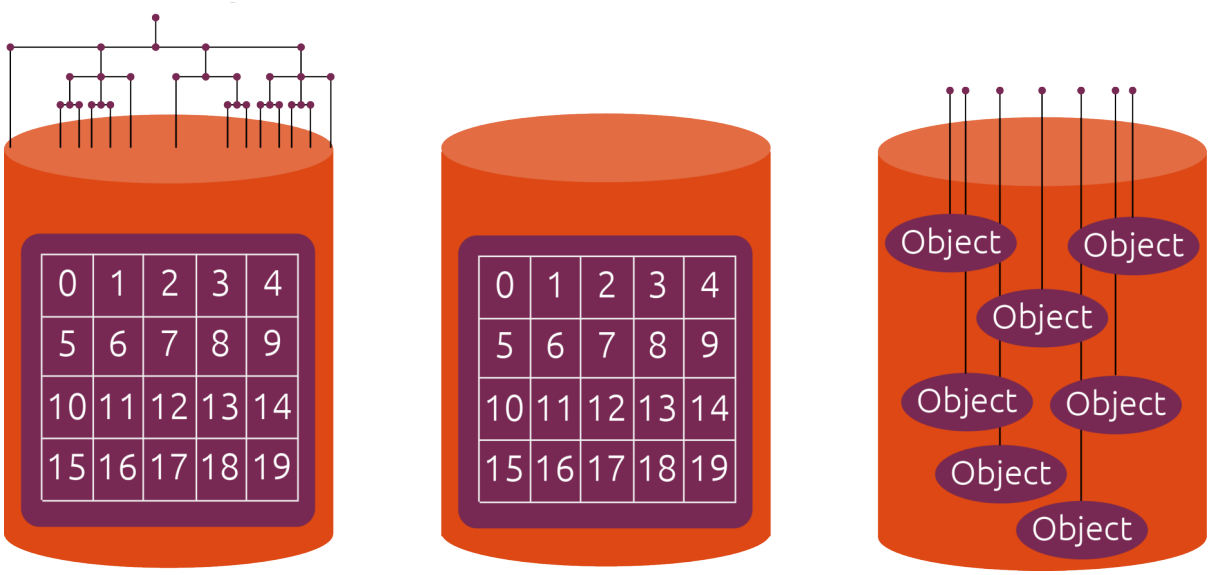
Os sistemas de armazenamento em nuvem normalmente são utilizados para armazenar dados provenientes diretamente dos usuários ou resultados de computação das aplicações em nuvem. Um *cloud storage* possui infraestrutura de armazenamento escalável e os dados são armazenados por múltiplos servidores de armazenamento em vez de um único servidor dedicado usado em soluções de armazenamento tradicionais. O usuário enxerga o sistema de armazenamento como um servidor virtual remoto e não como um componente físico manipulável. Isto é, o usuário possui uma referência do dado armazenado e espaço de armazenamento, mas não possui conhecimento sobre a localização física ou controle sobre os componentes da infraestrutura. A localização atual dos dados dos usuários pode mudar frequentemente em consequência do gerenciamento dinâmico do desempenho dos servidores e espaço de armazenamento disponível da nuvem (WU *et al.*, 2010).

Além de sistemas de armazenamento em nuvem serem escaláveis, eles tendem a serem soluções de baixo custo financeiro capaz de prover alta durabilidade e disponibilidade dos dados. Sistemas de armazenamento em nuvem podem utilizar a virtualização dos recursos dos dispositivos de armazenamento para suportar quantidades maiores de clientes em paralelo e diminuir os custos de manutenção, se comparado com soluções proprietárias com recursos

físicos dedicados acessíveis pela rede. Dependendo do sistema de armazenamento em nuvem, ainda é possível garantir a durabilidade e disponibilidade dos dados através da replicação das informações entre outros servidores de armazenamento físicos. Mesmo que um ou mais servidores se encontrem indisponíveis, o sistema consegue funcionar normalmente e prover os dados para os usuários sem que eles estejam cientes dos problemas da infraestrutura de armazenamento.

2.2.1 Tipos de serviços de armazenamento em nuvem

Existem diversos tipos de serviços de armazenamento em nuvem propostos para atender a necessidades específicas de aplicações centradas em dados. Provedores de serviços de armazenamento em nuvem podem fornecer serviços de armazenamento focados no gerenciamento e manipulação dos dados como uma camada de abstração mais próximo ou distante do nível de persistência dos dispositivos de armazenamento (GROSSMAN *et al.*, 2009). Basicamente, as tecnologias de armazenamento podem manipular as unidades de armazenamento em três tipos de categorias: arquivo, bloco ou objeto. A Figura 1 ilustra os tipos de armazenamento de dados em três grandes categorias. A seguir, são discutidos com mais detalhes os níveis de abstração dos dados que os provedores de serviços de armazenamento oferecem.



(a) File storage. (b) Block storage. (c) Object storage.

Figura 1 – Tipos de armazenamento de dados (Fonte:(CANONICAL, 2015)).

Um sistema de armazenamento baseado em arquivo é uma abordagem clássica de armazenamento de dados. Os dados são acessados com a abstração de arquivos, onde um arquivo

é identificado por um nome e organizado hierarquicamente através de diretórios e subdiretórios como mostrado na Figura 1a. Um arquivo qualquer pode ser encontrado descrevendo o caminho dos diretórios até o nome do arquivo. Alguns atributos sobre um arquivo como o nome do criador, data de criação e tamanho são armazenados como metadados estáticos. Essa convenção facilita o compartilhamento pela rede de forma segura através de tecnologias como *Network-Attached Storage (NAS)* (GIBSON; METER, 2000) e protocolos de comunicação como o *Network File System (NFS)* (SHEPLER *et al.*, 2003). Sistemas baseado em arquivos apresentam bom desempenho e são adequados para armazenamento local, porém não são interessantes em ambiente de nuvem em larga escala por ser difícil de escalar. O *Google Distributed File System (GFS)* (GHEMAWAT *et al.*, 2003) e o *Hadoop Distributed File System (HDFS)* (SHVACHKO *et al.*, 2010) são exemplos de serviços de sistemas de arquivos distribuídos utilizados no ambiente de nuvem.

Serviços de armazenamento em nuvem baseado em blocos é um caminho direto para manipular unidades de armazenamento com um granulo menor do que um arquivo. Um volume pode ser pensado como um dispositivo de armazenamento virtual, independente e com tamanho de armazenamento total fixo, composto por pequenos blocos de unidades de armazenamento de mesmo tamanho. A Figura 1b ilustra um conjunto de blocos de armazenamento de um certo volume. Logo, um arquivo ou banco de dados pode ser armazenado em partes usando armazenamento baseado em blocos. Aplicações conseguem recuperar blocos pela rede usando seus endereços através de interfaces como *Internet Small Computer System Interface (iSCSI)* e *Fibre channel* (SHU *et al.*, 2005), comumente utilizadas em arquiteturas *Storage Area Network (SAN)* (KHATTAR *et al.*, 1999), onde é possível conseguir alto desempenho através da consolidação de dispositivos de armazenamento. Em armazenamento baseado em blocos, não faz sentido a existência de metadados associado a blocos, uma vez que as partes de dados isoladas só possuem algum significado quando agrupadas e gerenciadas por aplicações como sistemas operacionais e sistema de arquivos. O *OpenStack Cinder* (ROSADO; BERNARDINO, 2014) e o *Amazon Elastic Block Store (EBS)* (AMAZON, 2017a) são exemplos de serviços em nuvem que fornecem armazenamento em blocos.

Esta dissertação, foca em serviços baseados em objetos ou *object storages* como são regularmente chamados. *Object storages* são tecnologias emergentes que usam a abstração de objetos para facilitar a manipulação dos dados. Um objeto é uma abstração para uma unidade de armazenamento conceitualmente de tamanho ilimitado, isto é, não há um limite de tamanho

máximo de um objeto, pois o objeto pode ser armazenado em partes entre vários dispositivos de armazenamento físicos. A Figura 1c ilustra o armazenamento de objetos em um dispositivo de armazenamento. Um objeto pode ser usado para armazenar qualquer tipo de dado como arquivos, registros de banco de dados, imagens ou arquivos multimídia. Além disso, um objeto é dinâmico e autocontido, ou seja, o metadado associado a um determinado dado está contido no próprio objeto (MESNIER *et al.*, 2003). Os usuários podem alterar os dados e metadados de um objeto sem restrições de tamanho. Objetos podem ser isolados em contêineres como meio de organizar os objetos em conjuntos semânticos do ponto de vista das aplicações. Um contêiner é um conjunto lógico que serve para agrupar objetos, análogo a uma pasta em um sistema de arquivos. Um objeto é localizado por um identificador único (ID) que pode ser gerado a partir do conteúdo do objeto (dado e metadado) e o seu caminho lógico. O ID de um objeto é útil para recuperar um objeto de forma rápida por meio de uma rede de longo alcance como a *internet*. Como os componentes de armazenamento em uma nuvem podem estar separados geograficamente, o sistema pode criar e recuperar um objeto usando o seu ID através de interfaces de aplicação bem definidas sobre o protocolo HTTP. Exemplos de *object storages* incluem *OpenStack Swift* (SWIFT, 2017a), *Ceph* (WEIL *et al.*, 2006a) e *Dynamo* (DECANDIA *et al.*, 2007). Detalhes sobre o *OpenStack Swift* serão descritos nas seções 2.3 e 3.4.2, pois o mesmo é usado para a validação da contribuição proposta neste trabalho.

Os *object storages* gerenciam dispositivos de armazenamento chamados de *object-based storage devices (OSDs)* capazes de armazenar e recuperar dados no formato de objetos. Um OSD pode ser um simples dispositivo de armazenamento como um *HDD* ou *SSD* independente, assim como, um dispositivo de armazenamento customizado ou um agrupamento de dispositivos de armazenamento de baixo custo. Um OSD não é limitado ao acesso aleatório ou sequencial. A principal diferença entre um OSD e um dispositivo de armazenamento em blocos é a interface e não o tipo de mídia física (MESNIER *et al.*, 2005). O efeito imediato do armazenamento baseado em objetos é a transferência de responsabilidade do gerenciamento do espaço de armazenamento para aplicações especializadas, como banco de dados e sistema de arquivos locais. A Figura 2 ilustra a transferência de responsabilidade de uma sistema de arquivos tradicional para gerenciar o armazenamento em um OSD.

O modelo tradicional de um sistema de arquivos baseado em blocos é composto por duas interfaces: componentes do usuário e de armazenamento. O componente do usuário é responsável por representar estruturas de dados lógicas como arquivos e diretórios, enquanto que

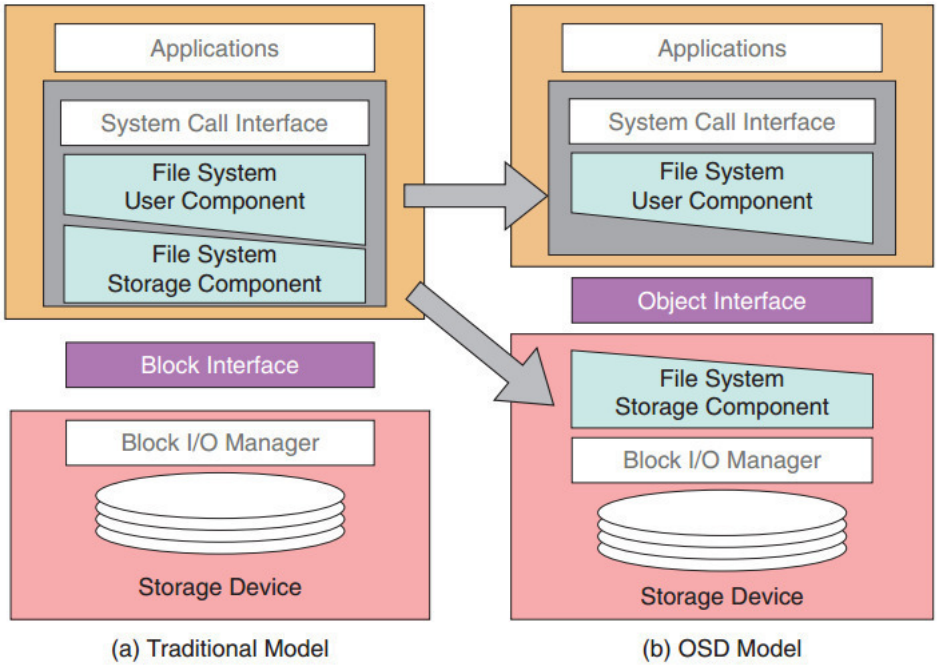


Figura 2 – Transferência de responsabilidade de uma sistema de arquivos para gerenciar o armazenamento em um *object-based storage device* (OSD) (Fonte:(MESNIER *et al.*, 2005)).

o componente do armazenamento mapeia as estruturas de dados para o dispositivo de armazenamento mediante uma interface de armazenamento em blocos. Nesse modelo, o componente de armazenamento está localizado na camada da aplicação como mostrado na Figura 2a.

O modelo baseado em OSDs consegue obter vantagem dessas interfaces, pois o OSD adiciona uma interface de objetos leve, capaz de diminuir a complexidade do gerenciamento dos dados ao delegar a responsabilidades para o sistema de arquivos localizado no OSD. O componente do usuário permanece inalterado, porém o componente de armazenamento é transferido para o dispositivo de armazenamento de modo que a interface de acesso do dispositivo é alterada de bloco para objeto. A adição da interface de objetos e a transferência de responsabilidades para o componente de armazenamento do modelo baseado em OSDs é mostrado na Figura 2b.

Sistemas de armazenamento baseados em objetos são sistemas de armazenamento com interfaces de alto nível de abstração, similares a sistemas de arquivos. Porém, diferente de sistemas baseado em arquivos ou em blocos, *object storages* apresentam-se como soluções de armazenamento altamente escaláveis devido a flexibilidade dos objetos, a padronização das interfaces de comunicação, a facilidade de gerenciamento dos OSDs e de acesso aos dados e metadados. A arquitetura dos sistemas de armazenamento de objetos, utilizada por provedores de serviços de armazenamento em nuvem, fornecem serviços para aplicações com diferentes benefícios (FACTOR *et al.*, 2005) (AZAGURY *et al.*, 2003). Por exemplo, os *object storages*

trazem benefícios para provedores de serviços e aplicações como:

- Eles facilitam o gerenciamento do armazenamento, porém com desempenho levemente inferior comparado com armazenamento baseado em blocos e sistema de arquivos.
- Os objetos são abstrações de dados customizáveis e com tamanho teoricamente infinito, manipuláveis pelas aplicações.
- Infraestrutura de armazenamento pode ser construída com dispositivos de baixo custo e fácil de escalar.
- Armazenamento massivo de dados não estruturados como textos, vídeos, áudios e imagens.
- Compartilhamento multiplataforma de dados baseados entre aplicações de forma simples e rápida.

2.3 Alocação de dados

Serviços de armazenamento em nuvem implementam estratégias e soluções particulares de como manipular e organizar os dados entre dispositivos de armazenamento com o objetivo de oferecer o melhor serviço possível para seus usuários. Eles enfrentam o desafio de otimizar a distribuição dos dados entre dispositivos de armazenamento conhecido como alocação de dados. A alocação de dados está relacionada à maneira de como atribuir unidades de dados (objetos, arquivos ou blocos) a nós de armazenamento (servidores e dispositivos de armazenamento) em um sistema distribuído para otimizar um ou vários critérios de desempenho, tais como reduzir o congestionamento de rede, utilizar menos espaço de armazenamento, melhorar o balanceamento de carga, dentre outros (PAIVA; RODRIGUES, 2015). Existe um custo-benefício (*tradeoff*) vinculado a cada tipo de critério das soluções de alocação de dados para otimizar o desempenho do sistema. Será discutido brevemente as principais estratégias de distribuição de dados ao longo desta seção.

Estratégias de alocação de dados baseadas em diretórios distribuídos são soluções que possuem a flexibilidade de definir alguma relação entre o dado e o nó de armazenamento em uma estrutura de dados (PAIVA; RODRIGUES, 2015). Sistemas de arquivos distribuídos como HDFS e GFS utilizam uma tabela de armazenamento customizadas para gerenciar a localização dos dados armazenados no sistema. Quando um arquivo é acessado, o sistema faz uma busca em uma grande tabela para retornar a localização do nó de armazenamento que guarda a informação consultada, análogo a um diretório remoto. Esse tipo de estratégia tem a vantagem de ser customizável, porém pode apresentar problemas como a degradação do desempenho das

consultas. Em larga escala, por conta da inserção e atualização da localização dos arquivos na estrutura de dados, o tamanho da estrutura pode crescer rapidamente a ponto de se tornar um gargalo de desempenho e armazenamento em memória principal. Como um subconjunto de nós é responsável por manter atualizada a estrutura de dados, os nós de armazenamento também podem ser tornar um gargalo visto que eles necessitam consultar os nós de gerenciamento para localizar os dados.

Distributed Hash Tables (DHTs) são sistemas distribuídos que utilizam funções de *hash* (*hash functions*) para distribuir dados através do mapeamento de chaves para nós de armazenamento. Este mapeamento é composto por um espaço de chaves com os possíveis valores de *hash* dos identificadores dos dados. Desta forma, o *hash* do identificador de um dado é utilizado como uma chave para acessar o dado armazenado em um determinado nó. Um DHT possui a característica de ser um sistema descentralizado, escalável e fácil de implementar. Os DHTs permitem a localização dos dados de forma bastante rápida uma vez que a chave gerada pelo hash do identificador do dado é mapeada de forma direta ao nó de armazenamento em que o dado se encontra (FELBER *et al.*, 2014). Dessa forma, um DHT pode ser usado como um método eficiente de distribuição e de acesso dos dados. Para mais detalhes sobre as abordagens clássicas de sistemas e protocolos baseados em DHTs como o *Pastry, Chord, Kademlia* e etc, é recomendada a leitura da análise comparativa dessas abordagens realizada em (LUA *et al.*, 2005) e (FELBER *et al.*, 2014).

Tabela 1 – Exemplo de mapeamento de objetos em um sistema de armazenamento que implementa um DHT com uma função de *hash* convencional.

Número total de *drives*	**h(x,n)**	**Mapeado para**
5	1	drive 1
6	3	drive 3
7	6	drive 6
8	7	drive 7

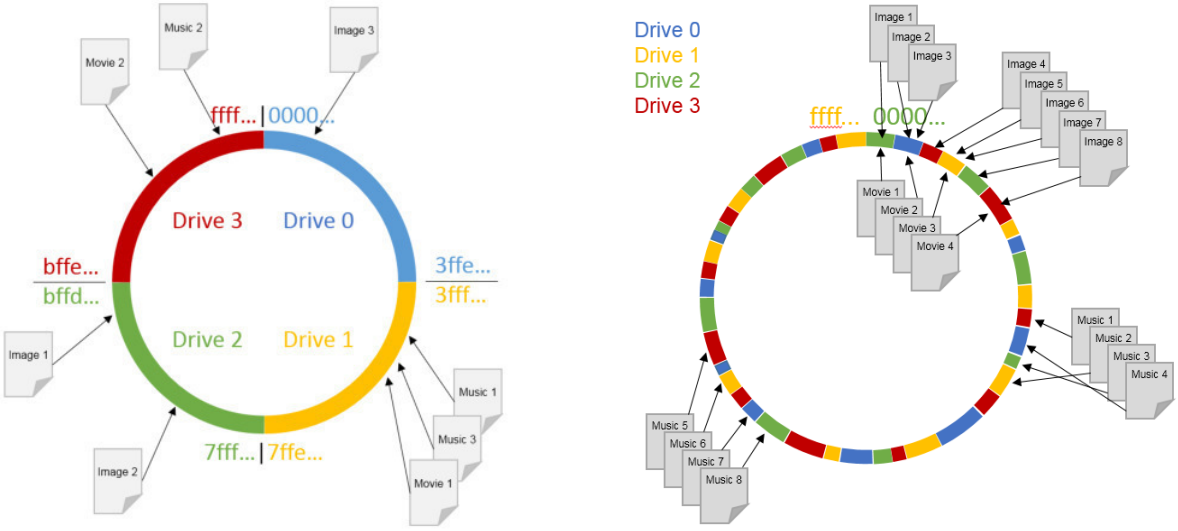
Um dos grandes desafios atribuídos aos DHTs em sistemas de armazenamento está relacionado a redistribuição das chaves entre os nós de armazenamento devido a adição e a remoção dos mesmos. O espaço de chaves gerenciado por um nó de armazenamento pode ser alterado, isto é, uma chave pode apontar para um novo nó de armazenamento. Dessa forma, o remapeamento de chaves do DHT implica que um processo de migração de dados será necessário para realocação dos dados previamente armazenados de forma a atender o novo mapeamento do espaço de chaves. Algumas abordagens baseadas em *hash* não conseguem lidar

bem com o dinamismo do espaço de chaves e podem causar sobrecarga na rede pelo excesso de movimentação de dados entre os nós de armazenamento.

Para esclarecer esse problema, considere um sistema de armazenamento composto por um conjunto de 5 dispositivos de armazenamento que são chamados de forma abreviada de *drives*. Cada objeto inserido nesse sistema recebe um ID único que é incrementado por 1 a partir do valor inteiro 0. Considere ainda que o sistema de armazenamento implementa um DHT com uma função *hash* $h : \mathbb{Z}+ \rightarrow \mathbb{Z}+$ que recebe como entrada x e n, onde x é o ID de um objeto e n é o número total de *drives* no sistema de armazenamento. A função h gera como saída uma chave $h(x,n)$ que é o resto da divisão do ID do objeto pelo número total de *drives*. Logo, um certo objeto com o ID igual a 111 é mapeado para o *drive* de número 1, pois o resto da divisão de 111 por 5 é igual a 1. O problema dessa função é que o calculo depende de n. Sempre que um *drive* é adicionado ou removido do sistema de armazenamento, a função h tende a gerar uma nova saída e mapear o objeto para diferentes *drives* a partir do mesmo ID do objeto. Para demostrar esse fato, a Tabela 1 exibe para qual *drive* o objeto com ID igual a 111 deve ser mapeado quando novos *drives* são adicionados. Perceba que o objeto deve ser movido para um *drive* diferente sempre que um *drive* é adicionado ou removido. Além do mais, esse recálculo deve ser feito para cada objeto no sistema de armazenamento fazendo-se necessário que cada objeto seja migrado para atender o novo remapeamento do espaço de chaves. Esse processo de remapeamento resulta em uma grande movimentação de dados entre diferentes *drives* após uma mudança do número total de *drives*.

Desse modo, uma classe de DHT denominado de *HASH* consistente (KARGER *et al.*, 1997) foi projetada com o intuito de distribuir os dados entre nós de armazenamento de forma uniforme e evitar redistribuir grande parte dos dados armazenados quando adicionados ou removidos nós de um sistema de armazenamento. O espaço de chaves (conjunto contendo todos os valores de *hash*) em um *hash* consistente simples é representado no formato de um anel (*ring*) de modo que cada *drive* é responsável por gerenciar um intervalo contínuo de chaves com tamanho médio igual a $1/N$, onde N é o número total de chaves. Cada *drive* individual utiliza a mesma função de *hash* para localizar dados remotos e seus próprios dados.

A Figura 3a ilustra um *hash* consistente simples utilizado por um sistema de armazenamento, onde cada *drive* físico gerencia um intervalo contínuo de chaves. Note que o espaço total de chaves é dividido por 4 *drives*. O *drive* 0 gerencia o intervalo de chaves do valor 0000... ao $3ffe$..., o *drive* 1 gerencia o intervalo de chaves do valor $3fff$... ao $7ffe$... e assim por



(a) *Hash* consistente simples com *drives* físicos. (b) *Hash* consistente com *drives* virtuais.

Figura 3 – Exemplos de *hashs* consistentes (Fonte: (HONG, 2017)).

diante. Ainda na Figura 3a, o mapeamento de um objeto para um *drive* é indicado por uma seta que aponta para o intervalo de chave que ele pertence. Independente da função *hash* utilizada para calcular o valor *hash* de um objeto (ex: imagem, filme ou arquivo de música), é gerada uma chave (uma função MD5 pode gerar um valor *hash* do nome de um objeto no formato *9793b...*) utilizada para localizar em qual intervalo de chaves ela pertence e assim saber qual o *drive* físico específico o objeto deve ser armazenado ou recuperado. No caso de um novo *drive* físico ser inserido no DHT, a localização da grande maioria dos objetos armazenados nos *drives* físicos de um sistema de armazenamento não é alterada. Apenas uma pequena quantidade de objetos são movidos entre os *drives* físicos após a restruturação dos intervalos de chaves dos *drives* físicos no DHT, pois as chaves de alguns objetos previamente armazenados no sistema podem fazer parte do intervalo de chaves controlado pelo novo *drive* físico.

Uma versão mais sofisticada do *hash* consistente simples é o *hash* consistente com *drives* virtuais. Ao invés de utilizar um grande intervalo contínuo de chaves por *drive* físico, um *drive* físico agrupa um conjunto de *drives* virtuais que representam o particionamento do intervalo de chaves de um *drive* físico em intervalos disjuntos com tamanhos diferentes. Em outras palavras, os *drives* virtuais podem ser vistos como partições do espaço de chaves de um determinado *drive* físico. A Figura 3b ilustra um DHT consistente com *drives* virtuais, veja que o *drive* físico 2 (cor verde) utiliza vários *drives* virtuais para particionar o seu espaço de chaves. Por mais que possam existir muitos *drives* virtuais no *ring*, o tamanho do intervalo de chaves de um *drive* virtual é pequeno em relação ao número total de chaves do *ring*. Portanto,

a adição ou remoção de um *drive* físico do *ring* implica na movimentação de uma pequena quantidade de objetos. O DHT consistente com *drives* virtuais proporciona que alguns *drives* físicos gerenciem mais objetos que outros devido ao tamanho distinto do espaço de chaves de um *driver* virtual. Logo, a distribuição dos dados pode se tornar a mais justa possível para os critérios de desempenho e/ou capacidade considerados por soluções de armazenamento que utilizam DHTs consistentes com *drives* virtuais.

Algoritmos de alocação de dados baseados em *hash* consistentes são altamente empregados em sistemas de armazenamento com componentes homogêneos em virtude do balanceamento uniforme dos dados fornecido pelas funções de *hash*. Sistemas de armazenamento em nuvem populares que utilizam *hashs* consistentes incluem o *Ceph*, *Dynamo*, *OpenStack Swift* e *Cassandra* (LAKSHMAN; MALIK, 2010). Algoritmos de alocação de dados como o CRUSH (WEIL *et al.*, 2006a) e o *ring* DHT do *OpenStack Swift* fornecem certas flexibilidades para manipulação dos dados através de pesos para *drives* físicos. O conjunto de pesos define o grau de importância dos dispositivos no sistema de armazenamento, e são utilizados por administradores de sistemas ou estratégias de terceiros para atender critérios de desempenho e armazenamento dos usuários.

O algoritmo de alocação do *OpenStack swift* utiliza um *DHT* consistente com *drives* virtuais, suporte a replicação de objetos e controle parcial da quantidade de chaves para um determinado *drive* físico. Diferente da Figura 3b, o valor *hash* de um objeto não identifica diretamente para qual dispositivo objeto é atribuído, pois o *DHT* consistente do *OpenStack Swift* considera uma dimensão a mais por conta de uma política de redundância. A política de redundância padrão do *OpenStack Swift* replica um determinado objeto no contexto dos *drives* virtuais, ou seja, todos os objetos possuem o mesmo número de réplicas e cada *drive* virtual tenta manter apenas uma réplica de um objeto dado seu valor *hash*. O *DHT* consistente do *OpenStack Swift* é comumente chamado apenas de *ring* e sua construção exige alguns parâmetros como:

- O fator de replicação, que define o número de réplicas dos objetos.
- O *partition power*, que reflete o número total de *drives* virtuais divididos entre os dispositivos de armazenamento
- Um conjunto de *drives* físicos e pesos associados a eles.
- Unidades administrativas lógicas como regiões e zonas para considerar possíveis domínios de falha.

O *ring* é um DHT consistente adaptado para replicação com informações autocon-

tidas, de modo que o mapeamento de todas as réplicas de um objeto é feito diretamente para os *drives* físicos do *OpenStack Swift*. Por esse motivo, cada servidor mantém uma cópia do *ring*, pois ele provê o mecanismo de localização completo para as réplicas de um objeto. O *ring* localiza as réplicas de objetos entre dispositivos de armazenamento por meio de uma tabela contendo chaves (os valores de *hash* dos objetos) e identificadores de réplicas de objetos para dispositivos de armazenamento (*drives* físicos). A Tabela 2 é uma representação do *ring* sob a perspectiva de uma estrutura de dados que mapeia chaves e réplicas para os *drives* virtuais, assumindo que cada objeto é configurado com três réplicas (configuração padrão utilizado pelo *OpenStack swift*) nessa representação. Observe que um objeto cuja chave é 29*a*8 tem suas réplicas alocadas aos *drives* 0, 2 e 1.

Tabela 2 – Representação da estrutura de dados do DHT com suporte a replicação do sistema de armazenamento *OpenStack Swift*.

		Chaves					
		29a8	*bffe*	*ec25*	*3a03*	*b670*	*5g70*
Réplicas	1	drive 0	drive 4	drive 3	drive 0	drive 2	drive 0
	2	drive 2	drive 5	drive 1	drive 2	drive 0	drive 1
	3	drive 1	drive 0	drive 0	drive 4	drive 5	drive 3

O *ring* também fornece um mecanismo para gerenciar a quantidade de dados que cada *drive* físico gerencia a fim de otimizar o espaço de armazenamento ou desempenho do sistema. Esse mecanismo é conhecido por possibilitar uma ponderação baseada em pesos para especificar quantos *drives* virtuais cada *drive* físico irá gerenciar. Dessa forma, considere que o *ring* DHT do *OpenStack Swift* foi construído com apenas 4 *drives* físicos como mostra a Figura 4. O *drive* físico d_1 que possui o maior peso, irá gerenciar mais dados por estar responsável por mais *drives* virtuais. Enquanto que o *drive* físico d_4 que possui o menor peso, irá armazenar e gerenciar menos dados. A manipulação do pesos implica na alteração da distribuição da proporção de dados por dispositivo de armazenamento, ou seja, dados existentes no sistema de armazenamento necessitam ser migrados entre os *drives* físicos para atender a nova configuração dos pesos. Além da migração dos dados existentes, novos dados enviados ao sistema de armazenamento são inseridos já atendendo a essa nova configuração de pesos. Dessa forma, devido uma nova configuração dos pesos, os *drives* físicos com maiores pesos tanto receberiam os dados previamente armazenados entre os outros dispositivos de armazenamento

por meio da migração de dados quanto receberiam novos dados por meio de novas requisições feitas por clientes do sistema de armazenamento. Assim, esse mecanismo de peso permite controlar tanto a distribuição da proporção dos dados entre os dispositivos e armazenamento quanto a proporção das requisições destinadas a eles.

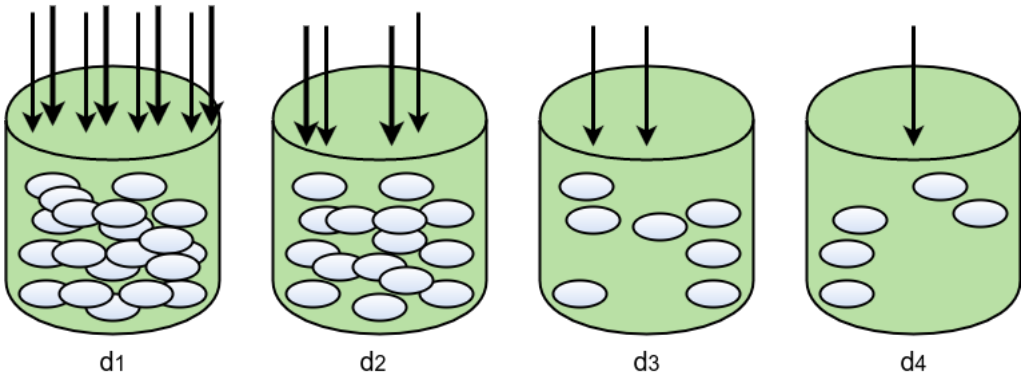


Figura 4 – Configuração dos pesos dos dispositivos de armazenamento.

Apesar do mecanismo de pesos fornecido pelo *ring* DHT do *OpenStack Swift* possibilitar o gerenciamento da proporção de dados em cada *drive* físico, ele pode conflitar com a quantidade de réplicas entre os *drives* que foi previamente definida. Dessa forma, o *ring* impede a alocação de réplicas redundantes, isto é, *drive* virtuais de mesma chave a um mesmo *drive* físico. Por exemplo, na estrutura de dados do *ring* mostrado na representação da Tabela 2, o *drive 0* foi configurado com um peso maior em relação aos demais, por isso 6 réplicas foram alocadas para ele enquanto que uma quantidade menor de réplicas são alocadas para outros *drives*. O algoritmo de distribuição dos *drives* virtuais entre chaves e réplicas ainda é responsável por garantir a dispersão dos dados, ou seja, um certo *drive* físico não deve armazenar réplicas redundantes de um mesmo objeto e as réplicas devem estar o mais distante possível umas das outras entre regiões, zonas e dispositivos de armazenamento. Fazendo uma analogia do *ring* do *Swift*, representado pela Tabela 2, e o *ring* da Figura 3b, o *ring* da Figura 3b é um caso especial do *ring* do *Swift* quando o número de réplicas dos objetos é igual a 1.

O algoritmo CRUSH, implementado pelo *Ceph*, utiliza uma abordagem pseudo aleatória para distribuição dos objetos do sistema com suporte à replicação em larga escala. O CRUSH utiliza diferentes tipos de funções de *hash* e *buckets* para mapear hierarquicamente objetos para componentes lógicos e dispositivos de armazenamento em um sistema de armazenamento. Uma implantação hierárquica em larga escala do CRUSH pode ser pensado em termos de *buckets* para representar componentes lógicos organizacionais como fileiras (*rows*) de gabinetes de servidores (*cabinets*), onde cada gabinete contem várias prateleiras (*shelves*) de dispositivos

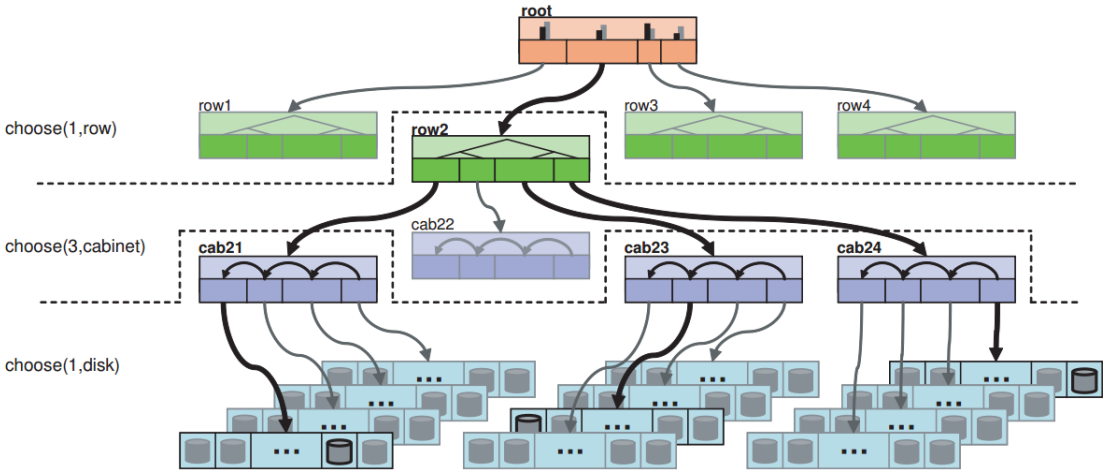


Figura 5 – Mapeamento hierárquico das réplicas de um determinado objeto proposto pelo algoritmo de distribuição de dados CRUSH (Fonte: (WEIL *et al.*, 2006a)).

de armazenamento (*disks*). Basicamente, o mapeamento hierárquico do CRUSH é composto de dispositivos de armazenamento e diferentes tipos de *buckets* com pesos associados a eles. Os pesos ponderam a alocação de dados em cada dispositivo de armazenamento. Um dado é inserido na hierarquia organizacional de forma recursiva, por meio de uma busca em *buckets* aninhados, via funções *hash* pseudo aleatórias. Os diferentes tipos de *buckets* (*uniform, list, tree e straw*) apresentam diferentes algoritmos de seleção com diferentes *tradeoffs* associados à complexidade computacional e movimentação de dados quando dispositivos são adicionados ou removidos (WEIL *et al.*, 2006a). O CRUSH consegue suportar diferentes estratégias de replicação de modo que as réplicas estejam localizadas em domínios de falha diferentes, garantindo a disponibilidade e o desempenho no acesso as réplicas.

Para elucidar o mapeamento hierárquico com uma política de replicação de objetos do CRUSH, considere uma política de replicação onde são alocadas 3 réplicas por objeto. Ainda, a regra de alocação base do CRUSH define que elas sejam distribuídas entre 3 gabinetes (*cabinets*) na mesma fileira (*row*). A Figura 5 ilustra o mapeamento hierárquico das réplicas de um determinado dado, em quatro níveis hierárquicos (*rows, cabinets* e *shelves* de dispositivos de armazenamento). Veja que é realizado um mapeamento inicial a partir do identificador de um objeto para um *bucket* do tipo *row* como a raiz (*root*) do algoritmo CRUSH. As linhas em negrito representam a seleção de itens em cada operação de seleção. Nesse caso foi selecionado a fileira *row2*. A seleção subsequente seleciona 3 *buckets* do tipo *cabinet* (*cab21,cab23,cab24*) aninhado abaixo de *row2*. Logo em seguida, é feito um mapeamento iterativo entre os gabinetes em (*cab21,cab23,cab24*) e suas prateleiras *shelves* de discos *disks*, de modo que os 3 gabinetes

resultem em um único disco aninhado a cada um dos gabinetes. Como resultado final, o CRUSH retorna 3 dispositivos de armazenamento distribuídos entre diferentes gabinetes, mas todos na mesma fileira. Por meio de regras de mapeamento de réplicas dos objetos para dispositivos de armazenamento fornecidas pelo CRUSH, as réplicas podem ser armazenadas e localizadas a um *pool* de recursos de armazenamento.

2.4 Desafios de balanceamento de carga em nuvem

A implantação de serviços de computação em nuvem enfrentam desafios relacionados ao método de fornecer o melhor serviço com o menor esforço possível com os recursos existentes disponíveis. Esses e outros desafios são vinculados ao balanceamento de carga. O balanceamento de carga em nuvem é compreendido como um mecanismo capaz de distribuir de forma eficiente uma determinada carga de trabalho entre múltiplos nós para melhorar o desempenho geral do sistema, utilização dos recursos, tempo de resposta, estabilidade, disponibilidade e várias outras características atribuídas às necessidades das aplicações dos serviços de computação em nuvem (DESAI; PRAJAPATI, 2013). A natureza dinâmica das operações em computação em nuvem pode provocar sobrecarga ou subutilização dos recursos em uma infraestrutura de nuvem, tornando ainda mais difícil balancear a carga de trabalho.

Nesta seção, são discutidos os principais desafios encontrados ao projetar soluções para o problema de balanceamento de carga em nuvem. Esses desafios também estão presentes em diferentes camadas da pilha de serviços da computação em nuvem, porém este trabalho foca em sistemas de armazenamento baseados em objetos. Trabalhos no estado da arte de balanceamento de carga em nuvem como (NUAIMI *et al.*, 2012) e (MESBAHI; RAHMANI, 2016), consolidam os principais desafios enfrentados para mitigar o problema de balanceamento de carga em nuvem. Entre tais desafios, pode-se destacar:

- **Heterogeneidade dos dispositivos de armazenamento**. Apesar de existirem diversas abordagens na literatura que conseguem distribuir bem a carga de forma uniforme entre nós de armazenamento, como as baseadas em DHTs (FELBER *et al.*, 2014), pouco tem se trabalhado em técnicas para sistemas de armazenamento heterogêneos em nuvem. Estes ambientes, frequentemente surgem pela necessidade dos provedores de armazenamento em nuvem terem que atualizar ou estender sua infraestrutura com armazenamento modernos. Algoritmos de balanceamento de carga precisam lidar com a vazão distinta dos dispositivos de armazenamento, pois determinados dispositivos podem processar operações de entrada

e saída com muito mais rapidez do que outros. Portanto, a má utilização dos recursos heterogêneos pode causar o retardamento das requisições dos clientes e o desperdício de recursos de um sistema de armazenamento. Algoritmos de balanceamento que buscam ajustar dinamicamente a carga de trabalho com as capacidades distintas dos dispositivos tendem a diminuir o tempo de resposta das requisições. Por conseguinte, eles otimizam o desempenho geral do sistema de armazenamento tirando vantagem da heterogeneidade do ambiente.

- **A distribuição espacial dos servidores**. Alguns algoritmos são projetados de forma eficiente para funcionar com servidores locais, ou seja, fisicamente próximos uns dos outros. Entretanto, é um grande desafio projetar um algoritmo eficiente que considere a distribuição espacial dos servidores de armazenamento. Esse desafio é motivado pelo fato de poder existir um grande atraso da conexão de rede entre os elementos participantes do processo de balanceamento. As requisições dos clientes podem sofrer com atraso do tempo de resposta devido a distância física dos servidores responsáveis pelo processamento das requisições. Assim, se faz necessário desenvolver estratégias de balanceamento que controlem os servidores distribuídos espacialmente ao longo do sistema e que sejam eficientes ao considerarem atrasos de rede.
- **Replicação**. Um dos grandes desafios ao projetar os algoritmos de balanceamento de carga é levar em consideração a camada de armazenamento do sistema com uma determinada politica de redundância. Assim, algoritmos de balanceamento de carga que utilizam replicação total propõem manter alta disponibilidade entre os servidores e dispositivos de armazenamento. Porém, esses algoritmos não conseguem ser tão eficientes pois eles exigem maiores custos de transmissão na rede, processamento e armazenamento na sincronização necessária para manter as réplicas atualizadas em todos os dispositivos de armazenamento. Para contornar esse tipo de problema, a política de replicação parcial é utilizado por algoritmos de balanceamento para economizar os recursos do sistema de armazenamento. Apesar desse tipo de política direcionar em uma melhor utilização dos recursos na camada de persistência de dados, existe a desvantagem de aumentar a complexidade do algoritmos que adotam esse tipo de política. Isso acontece pelo fato dos algoritmos atribuírem em seus modelos apenas algumas réplicas que estão distribuídas entre diferentes dispositivos de armazenamento ao longo da infraestrutura da nuvem. O desempenho das soluções que utilizam alguma política de redundância ainda pode ser

melhorado quando considerada a heterogeneidade da capacidade de armazenamento e a vazão dos dispositivos de armazenamento do sistema de armazenamento em nuvem, seja através da escrita ou seleção de réplicas.

- **Ponto central de falha**. O controle do estado de balanceamento de carga e a coleta de informações de diferentes componentes de um sistema de armazenamento em nuvem devem ser projetados de forma a evitar um único ponto de falha no algoritmo. Algoritmos de balanceamento de carga cujo controle é centralizado normalmente possuem baixa complexidade e são mais fáceis de ser implementados devido ao conhecimento global do estado do sistema. Porém, um único controlador torna arriscado o funcionamento constante do sistema. Em alguns casos, se o controlador falhar, todo o sistema irá falhar junto. Dessa forma, os algoritmos de balanceamento de carga devem ser projetados para superar esse desafio. Algoritmos de balanceamento com controle distribuído parecem ser a melhor abordagem. Entretanto, eles possuem a desvantagem de serem mais complexos e exigem maior coordenação e controle dos servidores para que o algoritmo de balanceamento funcione corretamente, sem gerar sobrecargas.

2.5 Taxonomia dos algoritmos de balanceamento de carga em nuvem

Os algoritmos de balanceamento de carga para sistemas de armazenamento em nuvem podem utilizar estratégias diferentes, considerar características particulares do sistema e seguir uma arquitetura centralizada ou distribuída. Um conjunto de estratégias de balanceamento de carga pode conter características semelhantes. Dessa forma, em (NUAIMI *et al.*, 2012), (DESHMUKH; DESHMUKH, 2015) e (MEGHARAJ, 2016), os autores discutem como as estratégias de balanceamento de carga para sistema de armazenamento em nuvem podem ser classificadas seguindo os padrões de decisão de projetos mais frequentes na construção das soluções. A seguir, é discutido as principais vantagens e desvantagens de cada tipo de abordagem.

Segundo os autores, existem duas abordagens diferentes que representam a taxonomia dos algoritmos de balanceamento de carga: *Static Load Balancing (SLB)* e *Dynamic Load Balancing (DLB)*. A Figura 6 ilustra a taxonomia dos algoritmos de balanceamento de carga em sistemas de armazenamento distribuídos.

A abordagem SLB possui como principal característica distribuir a carga de trabalho na fase de inicialização da computação. Em outras palavras, a decisão do balanceamento é baseada no conhecimento prévio das propriedades e capacidades dos recursos do sistema, porém

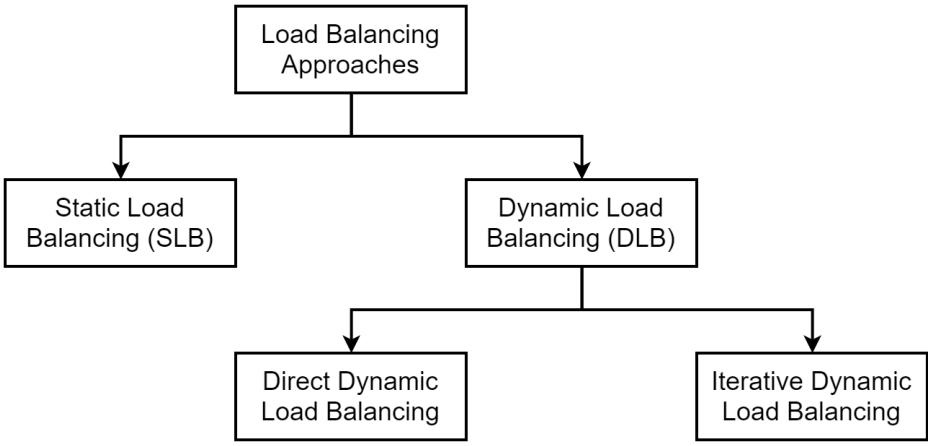


Figura 6 – Taxonomia dos algoritmos de balanceamento de carga em sistemas de armazenamento distribuídos (Fonte: (NUAIMI *et al.*, 2012), (DESHMUKH; DESHMUKH, 2015) e (MEGHARAJ, 2016)).

não consideram as mudanças dinâmicas e são incapazes de adaptar a carga durante a execução do sistema. Essa abordagem geralmente é utilizada para resolver problemas em ambientes homogêneos com padrão de computação uniforme e é mais fácil de ser implementado. Porém, para sistemas não uniformes e heterogêneos, o desempenho não é tão satisfatório quanto esperado. Para melhorar o desempenho do sistema de armazenamento, a abordagem SLB necessita de conhecimento global do sistema.

A abordagem DLB é caracterizada pelo fato do balanceamento de carga levar em conta as capacidades prévias e os atributos dinamicamente coletados dos servidores de armazenamento. Esse tipo de abordagem consegue dinamicamente reatribuir a carga já existente entre os componentes do sistema, tendo como base as informações mais recentes dos atributos considerados pela estratégia. O mecanismo base dessa abordagem envolve a migração de dados entre os servidores de armazenamento. Esse tipo de abordagem é crucial em ambientes heterogêneos. Entretanto, é preciso lidar com com alguns desafios particulares como a escolha do servidor que engatilha a decisão do balanceamento, verificar o melhor momento para engatilhar o processo de balanceamento de carga e fazer a escolha dos dados envolvidos durante a fase de migração.

A abordagem dinâmica ainda pode ser classificada em duas outras abordagens em relação ao mecanismo de migração: direto ou iterativo (DESHMUKH; DESHMUKH, 2015). No método iterativo, os algoritmos fazem o balanceamento do sistema passo a passo, normalmente, transferindo a carga em excesso dos recursos sobrecarregados para os subutilizados. No método direto, a execução do balanceamento de carga é feita em um único passo, após a verificação da carga em excesso de cada servidor de armazenamento. Como esse tipo de estratégia necessita conhecer o estado atual de todo o sistema em relação a capacidade e o excesso de carga em

cada servidor, protocolos *broadcasts* são comumente usados para disseminar tais informações. Embora o método direto realize o balanceamento em um único passo, o método leva mais tempo para convergir e consume mais largura de banda do sistema para obter o estado momentâneo de todos os atributos do sistema. Assim, esse *tradeoff* do método direto em relação ao estado da carga do sistema tem que ser considerado no projeto de algoritmo de balanceamento dinâmico.

Segundo o trabalho em (DONG *et al.*, 2012), os algoritmos de balanceamentos de carga podem ser classificados seguindo outro ponto de vista. Os autores consideram como os servidores tomadores de decisões estão organizados em relação ao modelo arquitetural do sistema de armazenamento. Em outras palavras, os algoritmos dinâmicos podem ser categorizados considerando a decisão dos servidores em relação ao gatilho de uma estratégia de balanceamento da carga de trabalho. A decisão pode seguir uma abordagem centralizada, baseada em grupos ou totalmente distribuída.

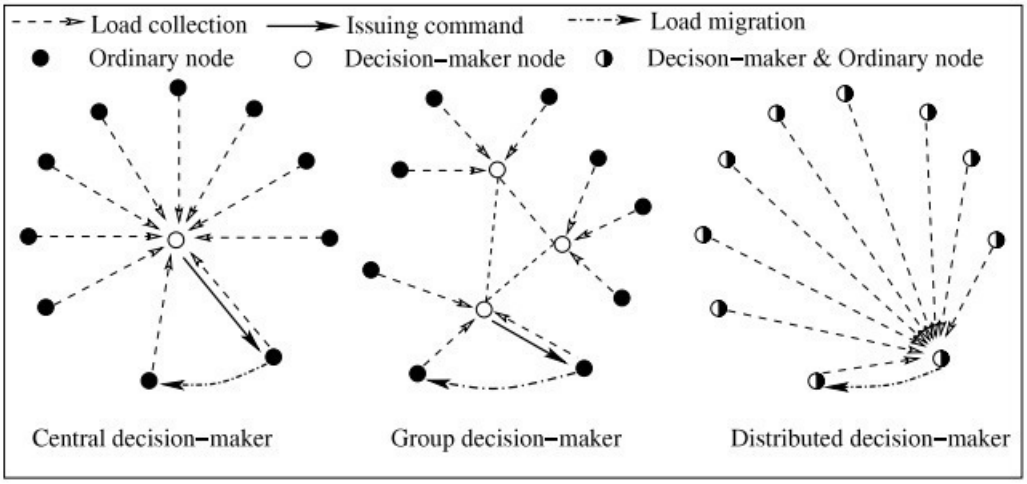


Figura 7 – Decisão centralizada, baseada em grupos e distribuída. (Fonte: (DONG *et al.*, 2012))

A Figura 7 mostra as três possíveis categorias que servidores tomadores de decisão (*decision-maker*) podem pertencer. Na decisão centralizada (*central decision-maker*), um nó central é responsável por coletar informações da carga de trabalho de todos os nós de armazenamento (*ordinary node*). Em seguida, a migração da carga é realizada entre nós de armazenamento para regular o estado global do sistema seguindo algum critério particular de uma estratégia de balanceamento de carga. A decisão baseada em grupos (*group decision-maker*) é uma extensão da decisão centralizada. Um conjunto de nós de decisão coletam a carga de determinados nós de armazenamento e trocam mensagens entre si para engatilhar a migração da carga de trabalho entre nós de armazenamento. Na decisão distribuída (*distributed decision-maker*), cada nó no

sistema de armazenamento é um nó de decisão e armazenamento. Portanto, cada nó coleta a carga dos demais nós e contribui em parte para a tomada de decisão e migração da carga de trabalho.

Outros autores como (RAJESHWARI; DAKSHAYINI, 2014) e (MESBAHI; RAHMANI, 2016) apresentam classificações levemente diferentes das discutidas anteriormente, uma vez que os autores generalizam as soluções de balanceamento de carga presentes em qualquer camada do modelo de computação em nuvem. No trabalho apresentado em (FELBER *et al.*, 2014), os autores focam em uma taxonomia para estratégias de balanceamento de carga em sistemas baseados em DHTs. Apesar de existirem essas outras classificações de algoritmos de balanceamento de carga, elas se distanciam um pouco mais do contexto desse trabalho.

2.6 Conclusão

Este capítulo apresentou um conjunto de conceitos e definições vinculados ao balanceamento de carga em sistemas de armazenamento em nuvem. Foram definidos os aspectos mais importantes que circundam o paradigma de computação em nuvem, bem como seus modelos de serviços e implantação. Também, foram apresentados os principais conceitos e tipos de sistemas de armazenamento baseados em: arquivos, blocos e objetos. Um maior enfoque foi dado ao controle da distribuição do armazenamento baseado em objeto (*objet storages*) por ser o foco desse trabalho. Além disso, foram apresentados os principais desafios e a taxonomia das soluções de balanceamento de carga para sistemas de armazenamento em nuvem.

3 TRABALHOS RELACIONADOS

Este capítulo apresenta os principais algoritmos de balanceamento de carga para sistemas de armazenamento distribuídos no contexto deste trabalho. Ao final desse capítulo, é realizada uma discussão sobre as principais similaridades e limitações dos trabalhos relacionados.

3.1 Estratégia proposta em (HSIAO *et al.*, 2013)

A estratégia proposta em (HSIAO *et al.*, 2013) apresenta um algoritmo de balanceamento de carga para sistema de arquivos distribuídos. A proposta dos autores se baseia em distribuir a carga de maneira uniforme entre os servidores de armazenamento enquanto tenta reduzir o custo de movimentação dos arquivos já armazenados. Assim, a estratégia reduz a carga de servidores sobrecarregados transferindo uma parte desta para servidores subutilizados por meio de um processo iterativo e de rápida convergência.

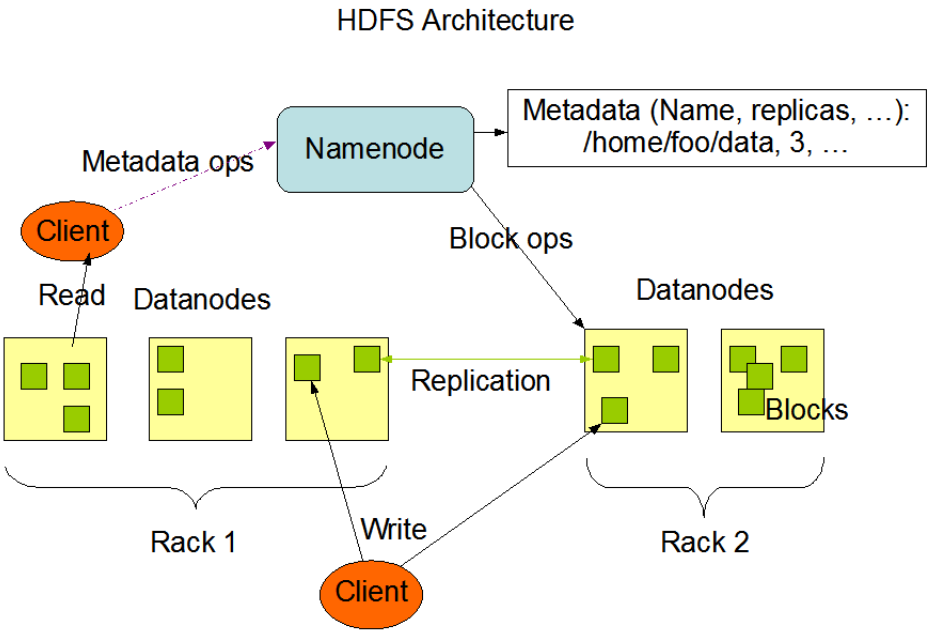


Figura 8 – Arquitetura do *Hadoop Distributed File System* (HDFS) (Fonte: (HDFS, 2017)).

Os autores consideram o *Hadoop Distributed File System* (HDFS) (HDFS, 2017) como estudo de caso para explorar o problema de balanceamento de carga em sistema de arquivos distribuídos. A Figura 8 mostra a arquitetura do HDFS. Nessa arquitetura, um determinado arquivo F é replicado e fragmentado em *chunks* de mesmo tamanho C_f, onde são armazenados em um conjunto V composto por n *datanodes*, ou seja, a cardinalidade de V é $|V| = n$. Observe que na arquitetura do HDFS exibida na Figura 8, é utilizada a nomenclatura de *datanode* para

um servidor de armazenamento e a carga destinada a um *datanode* é proporcional ao seu espaço de armazenamento disponível. O *namenode* é um servidor centralizado que recebe requisições dos clientes, gerencia os metadados e a localização dos arquivos armazenados no HDFS. Os arquivos dos clientes são armazenados em blocos chamados de *chunks*, onde os *chunks* podem ser replicados entre diferentes *datanodes* e *racks*. Nesse tipo de ambiente, falhas em componentes do sistema são comuns e o dinamismo de operações sobre os arquivos, tais como criar, atualizar e deletar, provoca uma distribuição não uniforme dos *chunks* armazenados entre os *datanodes*. Dado que a quantidade ideal A que cada *datanode* $i \in V$ deve manipular, definida por $A = \frac{\sum_{f \in F} |C_f|}{n}$, o algoritmo proposto tenta minimizar o fator de desbalanceamento em cada *datanode* i definido por $||L_i - A||$. Onde, L_i é definido como a carga do servidor de armazenamento i e $||.||$ representa a função de valor absoluto.

O algoritmo assume dois estados para os servidores de armazenamento: *light* e *heavy*. Um servidor *light* ou subutilizado é um servidor que possui o número de *chunks* menor que o limiar de $(1 - \Delta_L)A$, onde $0 \leq \Delta_L \leq 1$. Por outro lado, um servidor sobrecarregado ou *heavy* possui o número de *chunks* maior que $(1 + \Delta_U)A$, onde $0 \leq \Delta_U \leq 1$, sendo Δ_L e Δ_U parâmetros do sistema. O algoritmo implementa um protocolo no DHT de modo que se um servidor i sair e reingressar a rede como um sucessor de um servidor j, o servidor i será representado como servidor $j+1$, vizinho sucessor de j, e o sucessor original de j como $j+2$, o sucessor do sucessor de j como $j+3$ e assim por diante.

A versão centralizada do algoritmo de balanceamento de carga considera que cada servidor possui conhecimento global do estado do sistema. Assim, se um servidor i for o mais subutilizado entre os servidores *lights*, i imediatamente abandona o sistema e migra todos os seus *chunks* para o seu sucessor $i+1$. Logo em seguida, ele é reintegrado como sucessor do servidor mais sobrecarregado entre os servidores *heavy*. O servidor i reduz a carga do servidor j requisitando a carga em excesso, ou seja, é migrado $min\{L_j - A, A\}$ *chunks* do servidor j para o servidor i. Como o servidor j pode continuar sendo o servidor mais sobrecarregado, o mecanismo anterior é repetido iterativamente com o servidor mais subutilizado atual até que o servidor j, um servidor *heavy*, se torne um servidor $light$. Note que o algoritmo de balanceamento faz uma correspondência entre o servidor mais subutilizado com o servidor mais sobrecarregado para diminuir o custo de movimento e convergir mais rapidamente. Essas características são perceptíveis quando um servidor *light* migra poucos dados ao sair do sistema para que seu vizinho reduza grande parte da carga de um servidor *heavy* de forma a alcançar rapidamente

o estado de balanceamento global do sistema. Esse conceito é estendido com o objetivo de diminuir a complexidade do algoritmo e suportar operações paralelas, uma vez que os servidores *lights* tomam a iniciativa de balancear a carga de servidores *heavy*. Desse modo, o algoritmo possui uma lista U contendo os servidores subutilizados e outra lista O contendo os servidores sobrecarregados. A listas são ordenadas de tal modo que os servidores com cargas menores estejam pareados com os servidores com maiores cargas. Consequentemente, os servidores *light* conseguem solicitar *chunks* para seu respectivo servidor *heavy* de maneira paralela.

A versão final do algoritmo de balanceamento de carga não assume conhecimento global do sistema, ou seja, a decisão é distribuída. Para identificar a carga dos demais servidores de armazenamento, cada servidor *light* i mantém um vetor V que armazena o estado da carga e coleta amostras de s servidores escolhidos aleatoriamente. Na perspectiva do servidor i e com base nas informações de carga coletadas de s servidores, é calculada a carga média ou ideal A_i que cada servidor deve manter. O servidor i cria então uma lista para os servidores *lights* e outra para os *heavy* a partir do vetor V, incluindo ele mesmo. Caso o servidor i detecte que ele é um servidor *light*, dentro da lista de servidores subutilizados, ele irá começar a solicitar *chunks* de um servidor sobrecarregado com posição correspondente na lista de servidores sobrecarregados. Assim, de forma semelhante à versão centralizada do algoritmo, o servidor i, ao sair do sistema, migra sua carga para seu vizinho e em seguida retorna para reduzir a carga do servidor *heavy* pré definido.

A estratégia distribuída ainda consegue explorar a localização física dos servidores para aliviar o congestionamento da rede, pois a localização no DHT é construída sobre endereçamentos lógicos e a distância física influencia diretamente para a queda de desempenho. Para lidar com esse problema, ao invés de coletar um único vetor V por iteração do algoritmo, um servidor *light* coleta um conjunto n_V de vetores e seleciona os servidores *heavy* fisicamente mais próximos baseado em mensagens *round-trip delay (RTD)* (ALMES *et al.*, 1999). Além disso, o algoritmo ainda tira vantagem da heterogeneidade em relação à capacidade de armazenamento de um servidor, estabelecendo a quantidade de *chunks* ideal A_i que cada servidor i deve manipular em função da sua capacidade de armazenamento. Os resultados experimentais demonstram que a estratégia proposta superou o algoritmo de balanceamento implementado no HDFS.

O algoritmo de balanceamento de carga apresentado em (HSIAO *et al.*, 2013), considera a heterogeneidade da capacidade de armazenamento dos dispositivos de armazenamento, porém este não considera a heterogeneidade do desempenho desses dispositivos. O algoritmo

ainda possui a desvantagem de não garantir réplicas redundantes entre os *datanodes*. Mesmo com baixa probabilidade, os *datanodes* podem apresentar réplicas redundantes dos *chunks* de um arquivo devido ao processo de migração para balancear a capacidade de armazenamento entre eles.

3.2 Estratégia proposta em (WANG *et al.*, 2015)

Na maioria dos sistemas de armazenamento distribuídos, os dados são armazenados entre uma determinada quantidade de nós (servidores de armazenamento). Algoritmos de balanceamento de carga como descrito em (HSIAO *et al.*, 2013) utilizam técnicas como *hash* para distribuir uniformemente os dados entre os nós de armazenamento. Consequentemente, a carga de acesso também é distribuída uniformemente entre os nós dos sistema. Todavia, essas estratégias não são adequadas quando o acesso aos dados não é uniforme.

O trabalho apresentado em (WANG *et al.*, 2015) busca resolver o problema de acesso a dados populares (*hotspots*) propondo um *framework* para balanceamento dinâmico de carga com gerenciamento de recursos adaptativo para *object storages* em nuvem. Os autores utilizaram o *OpenStack Swift* como estudo de caso para implementação e análise experimental. A Figura 9 mostra o mapeamento dos recursos virtuais para os componentes físicos correspondentes esperado pela solução proposta com o intuito de balancear a carga de acesso aos *hotspots*. O algoritmo de balanceamento de carga proposto foca em regularizar os recursos utilizados pelas máquinas virtuais (VMs) e máquinas físicas (PMs). Perceba que a mudança dinâmica do mapeamento das VMs em relação PMs, provocada pelo acesso aos dados, faz com que o balanceamento da carga seja realizado na camada das máquinas físicas através da migração de VMs entre PMs.

Para atingir o objetivo de balanceamento de carga e prover um gerenciamento adaptativo dos recursos do sistema de armazenamento, os autores propõem uma arquitetura para seu *framwork* como mostrado na Figura 10. A arquitetura é organizada de forma que possa fornecer a capacidade de monitorar os recursos virtuais e físicos utilizados pelo algoritmo de balanceamento para fins de decisão.

A estratégia de balanceamento proposta é composta por quatro algoritmos com responsabilidades distintas. O primeiro algoritmo chamado de *Resource Reallocate Algorithm* (RRA) procura regularizar os recursos computacionais alocados na camada de virtualização de acordo com a carga de trabalho. Para saber qual intensidade da carga que cada nó no sistema está

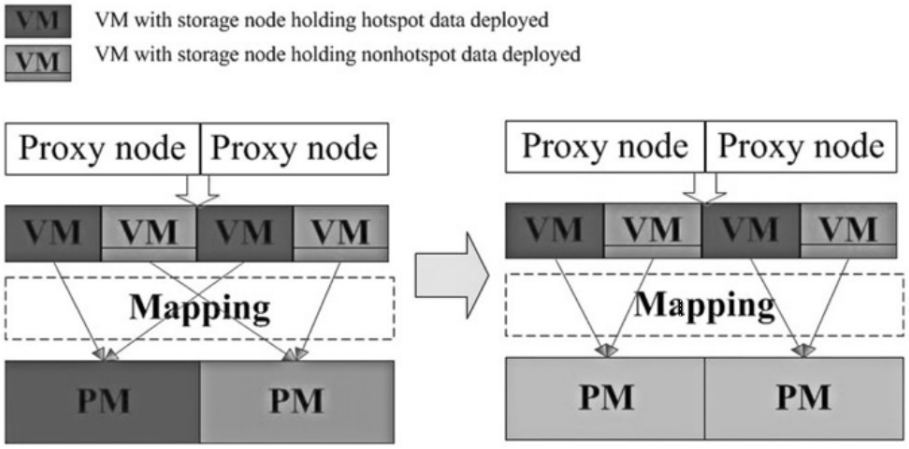


Figura 9 – Mapeamento entre *virtual machines* (VMs) e *physical machines* (PMs) esperado pela solução proposta em (WANG *et al.*, 2015) (Fonte: (WANG *et al.*, 2015)).

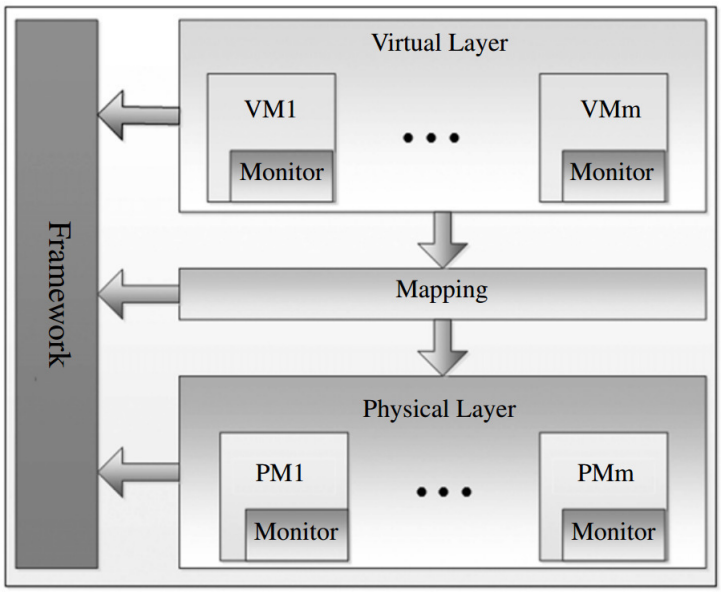


Figura 10 – Arquitetura do *Object Storage* utilizado em (WANG *et al.*, 2015) (Fonte: (WANG *et al.*, 2015)).

lidando, o *framework* define três tipos de nós em relação a carga de trabalho, seja ele uma VM ou PM: sobrecarregado, subutilizado e normal. Assim, o algoritmo RRA computa a quantidade de recursos que deve ser realocado em cada VM quando ela está sobrecarregada ou subutilizada. O segundo algoritmo, chamado de *Split Algorithm* (SA), e o terceiro algoritmo, denominado *Merge Algorithm (MA)* são utilizados para regularizar a carga na camada física, onde as PM residem. O SA é executado em uma PM quando ela está sobrecarregada com o intuito de aliviar a carga através da migração de VM's para outras PM's subutilizadas. O MA é um algoritmo semelhante ao SA em relação a migração de VMs, porém ele é executado entre PMs subutilizadas para consolidar o número de VM's em apenas uma só PM. Assim, as PMs não utilizadas podem ser temporariamente desligadas para economizar energia. Por último, o algoritmo denominado

Pair Algorithm (PA) é utilizado para dar suporte aos algoritmos MA e SA. O algoritmo PA otimiza a escolha das VMs que deverão ser regularizadas com as PMs antes do processo de migração. De forma objetiva, o algoritmo PA faz o melhor mapeamento de VMs para PMs baseado na utilização dos recursos, visando o estado de balanceamento futuro quando o processo de migração for concluído.

Apesar da estratégia de balanceamento de carga ser eficiente para mitigar o acesso dinâmico a dados populares em sistemas de armazenamento distribuídos, os autores não consideram características dos dispositivos de armazenamento para melhorar o desempenho do sistema de armazenamento. Além disso, na estratégia proposta, a caracterização da carga de trabalho não foi explorada e os experimentos publicados apresentam resultados obtidos a partir de componentes físicos e virtuais com características homogêneas.

3.3 ALDM: Adaptive Loading Data Migration in Distributed File Systems

Acesso a servidores com dados populares ou *hotspots* tornou-se um grande desafio para balanceamento de carga em sistemas de armazenamento distribuídos por causa da complexidade de ajustar a utilização dos recursos aos dados mais frequentes. Servidores contendo dados populares são sobrecarregados rapidamente devido à alta demanda da carga de trabalho e, como consequência direta, o tempo de resposta dos clientes aumenta e o desempenho geral do sistema tende a ser reduzido. Assim, o algoritmo denominado *Adaptative Loading Data Migration in Distributed File System (ALDM)* (TAN *et al.*, 2013) trata do problema de *hotspot* no balanceamento de carga em sistemas de armazenamento distribuídos.

O algoritmo ALDM é um algoritmo adaptativo com relação à carga de trabalho. Para isso, o ALDM utiliza a migração de dados como técnica principal para melhorar o desempenho de sistemas de arquivos distribuídos. O ALDM se baseia no trabalho de (KUNKEL; LUDWIG, 2008) para capturar a utilização dos recursos de acordo com a carga de trabalho. Em (KUNKEL; LUDWIG, 2008), os autores propõem um método para calcular a utilização de um determinado recurso baseado no conceito do preço. A intuição é que o preço de um recurso caia a medida que ele é utilizado ao longo do tempo. Assim, dada a utilização de um determinado recurso u_i, seu preço é dado por $P(u_i) = P_{max}\frac{K_i^{u_i}-1}{K_i-1}$. Onde K_i reflete a degradação do desempenho provocado pela utilização dos recursos e P_{max} é máxima precificação do recurso em destaque. Com o valor apropriado para K é possível descrever com melhor precisão o comportamento da utilização de um determinado recurso. A Figura 11 mostra diferentes curvas para o preço de um recurso

fixando um dado valor para K e variando a utilização de um recurso ao longo do tempo. Quando o valor K está no intervalo $(0,1)$ e a utilização do recurso é baixa, o preço do recurso é alto. Quando o valor de K está no intervalo $(1,\infty)$, a utilização do recurso é alta e o preço do recurso é baixo. O preço do recurso será o maior possível quando a utilização do recurso é 100%. O valor de K é empírico e dependente do tipo de recurso.

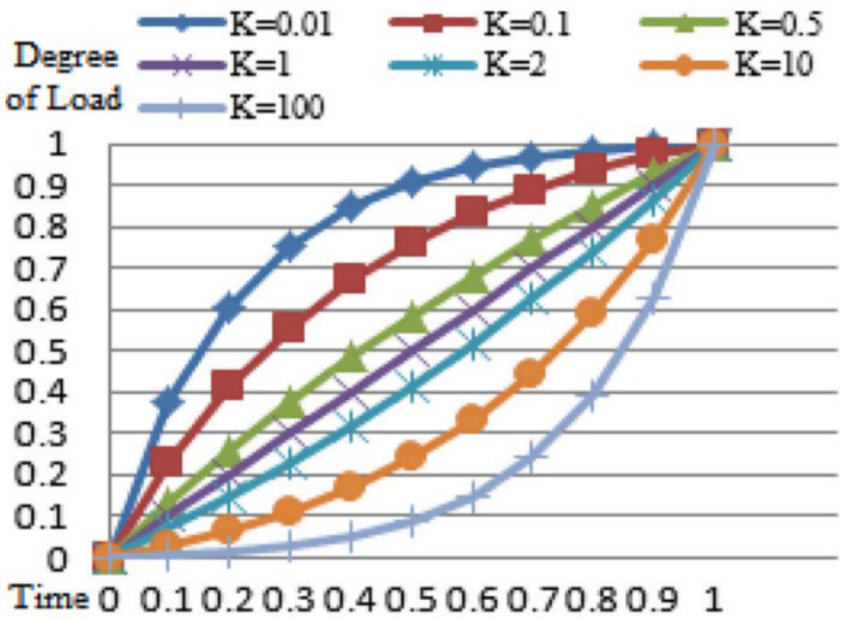


Figura 11 – Mudança do grau da carga de trabalho com a utilização dos recursos (Fonte: (TAN *et al.*, 2013)).

O ALDM utiliza o preço de um determinado recurso para modelar a carga de trabalho especifica para esse recurso. A carga de trabalho para o recurso de rede é definida como $N_i(n_i) = \frac{K_N^{n_i}-1}{K_N-1}$, onde n_i é a utilização dos recursos de rede, calculada como a média do número *Input/Output* (I/O) de rede dividido pela capacidade de rede total. De forma semelhante, a carga para I/O de disco no servidor de armazenamento é definida como $D_i(d_i) = \frac{K_D^{n_i}-1}{K_D-1}$, onde d_i é a utilização do I/O de de disco dos servidores de armazenamento calculada como a média da utilização em um dado período de tempo. A carga para a capacidade de armazenamento é definida como $C_i = \frac{C_u}{C_a}$, onde C_u é a capacidade utilizada e C_a é a capacidade total de armazenamento de um dispositivo de armazenamento.

A arquitetura do ALDM foi projetada para que um servidor de controle (CN) colete a carga dos servidores de armazenamento (DN) e consiga tomar uma decisão para balancear as requisições realizadas ao sistema, baseado na carga dos recursos de rede, I/O de disco e capacidade de armazenamento. A arquitetura do ALDM, como mostra a Figura 12, considera

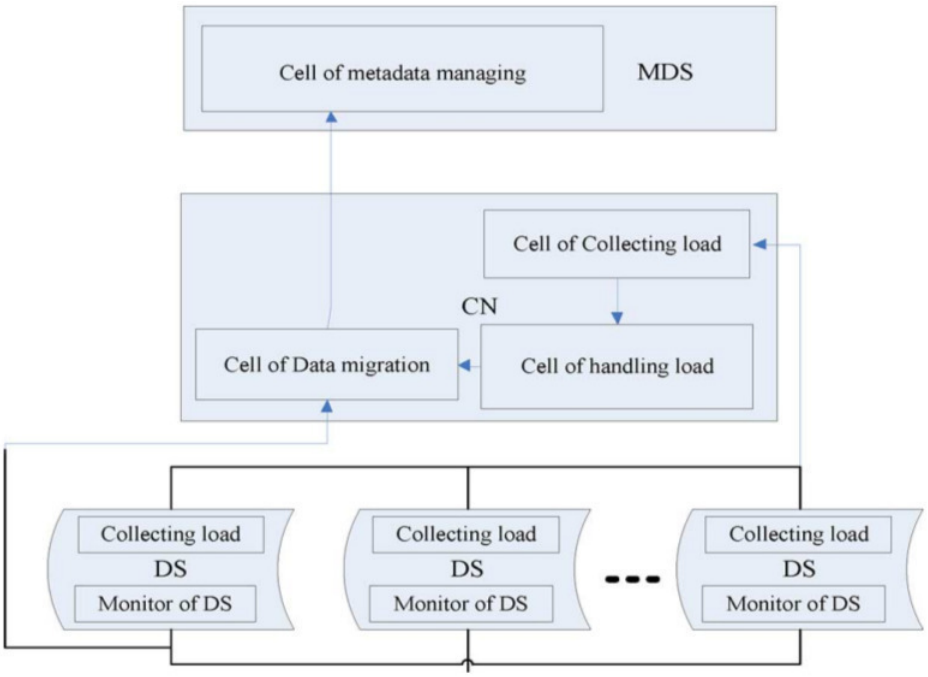


Figura 12 – Arquitetura do *Adaptative Loading Data Migration* (ALDM) (Fonte: (TAN *et al.*, 2013)).

que um CN é responsável por realizar o controle da migração de dados, assim como a escolha dos servidores participantes desse processo. O *CN* também é responsável por fazer a limpeza de possíveis dados de controle e solicitar a atualização de metadados após a migração de dados para um servidor de metadados (MDS).

O algoritmo ALDM realiza o balanceamento de carga por meio da coleta da utilização dos recursos dos servidores de armazenamento *DS*. Com a estimativa da utilização de um determinado tipo de recurso é possível calcular a sua carga, e para cada DS, o fator de desbalanceamento é analisado dinamicamente em relação ao desvio padrão dos diferentes tipos de carga que eles manipulam. No ALDM, os arquivos ainda são classificados em *hot*, *cold* ou *warm* com o objetivo de estimar o custo de desempenho ao migrar cada tipo de arquivo. O tipo de arquivo com menor custo será o escolhido para o processo de migração de dados.

Os experimentos realizados comprovam a eficiência da solução ao tratar o problema de *hotspots*, melhorando a largura de banda agregada do sistema com adaptatividade a diferentes tipos de cargas. Contudo, o cenário do experimento não varia parâmetros da carga de trabalho para analisar seu comportamento em relação ao aumento dinâmico da concorrência das requisições feitas ao sistema. Os experimentos foram realizados apenas com 4 clientes configurados de forma estática. Por fim, o ALDM funciona apenas com componentes homogêneos.

3.4 QoSC: A QoS-Aware Storage Cloud based on HDFS

Sistema de armazenamento distribuídos como HDFS são construídos para tirar proveito de *hardware* comuns de baixo custo. Contudo, o projeto desses sistemas assume que eles serão executados em ambientes estáveis, onde recursos de rede não representam um gargalo para sistema. Esse tipo de sistema, quando submetido a instabilidades na rede, compromete a utilização dos recursos e a qualidade de serviço pode não ser atendida. O trabalho *A QoS-Aware Storage Cloud Based on HDFS* (QoSC) (YANG *et al.*, 2015) aborda o problema de balanceamento de carga pelo ponto de vista da instabilidade do ambiente de rede dos sistemas de armazenamento distribuídos.

A arquitetura do QoSC encapsula uma política de redundância para recuperação de dados e uma estratégia de armazenamento levando em conta o QoS dos servidores de armazenamento. Como QoSC é baseado no HDFS, alguns elementos pertencentes à arquitetura do HDFS também estão presentes na arquitetura do QoSC, apresentada na Figura 13. QoSC é um *framework* hierárquico composto por um conjunto de servidores autônomos chamados de *datanodes*, responsáveis por prover espaço livre para armazenamento em seus dispositivos de armazenamento. Os *datanodes* são organizados em grupos gerenciados por um *namenode*, logo um *datanode* j pertencente a um *namenode* N_i é identificado por D_{ij}. Um *namenode* N_i gerencia metadados e monitora o QoS do grupo de *datanodes* por meio de uma rede heterogênea, em relação à largura de banda e a latência para decisão de balanceamento de carga.

A arquitetura do QoSC apresenta um *web service* com o intuito de intermediar o acesso dos serviços de gerenciamento por parte dos administradores e os serviços de armazenamento para usuários finais que desejam armazenar ou recuperar arquivos do sistema. As interfaces exportadas pelos *web services* para gerência do sistema permitem que estratégias autônomas tenham o controle sobre as requisições que estão chegando ao sistema por meio de configuração de pesos para os *datanodes*. QoSC utiliza um *namenode* para ajustar os pesos de um grupo de *datanodes* baseado no QoS dos *datanodes*, incorporado na estratégia de armazenamento dos dados. Quanto maior a disponibilidade, largura de banda de rede, baixa latência para transferência de dados e recursos disponíveis um *datanode* possuir, maior será o peso atribuído a ele. Consequentemente, mais dados e requisições serão destinados aos *datanodes* com maiores pesos.

Assim como no HDFS, no QoSC, os arquivos são armazenados e recuperados pelos clientes em pedaços e replicados ao longo dos *datanodes* de um determinado grupo. O HDFS

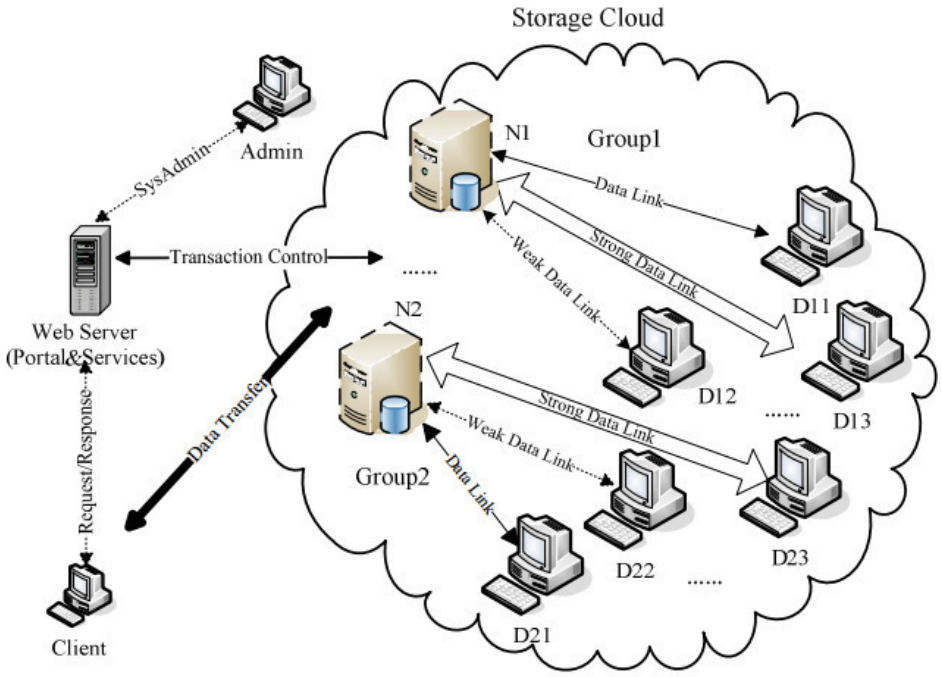


Figura 13 – Arquitetura do QoSC (Fonte: (YANG *et al.*, 2015)).

possui uma política de replicação para manter a disponibilidade das informações. Essa política estabelece que cada pedaço de um arquivo seja replicado em um conjunto de *datanodes*. Porém, esse tipo de política demanda maior espaço de armazenamento. O custo do espaço ocupado é proporcional ao tamanho de um arquivo e ao seu fator de replicação. Normalmente, o espaço ocupado é três vezes o tamanho de um arquivo. Desse modo, a política de redundância do QoSC propõe manter a disponibilidade dos dados, mas com menor custo de espaço de armazenamento. Para tal finalidade, QoSC mantém pedaços adicionais dos arquivos, especialmente utilizados para recuperar um arquivo como um todo em caso de falha. Assim, para um arquivo com N pedaços, M são pedaços especiais para recuperação do arquivo em caso de falhas, onde $M < N$ e os M pedaços são calculados em função dos N originais.

O balanceamento de carga do QoSC consiste em estimar os pesos W_{ij} para um *datanode* D_{ij}, uma vez que o peso define o número de pedaços de um arquivo que um *datanode* deve armazenar. O peso ótimo de um *datanode* é calculado em função da largura de banda ou taxa de transferência T_{ij}^{p}, disponibilidade do serviço A_{ij}^{p}, a utilização da CPU U_{ij} e a utilização do espaço de armazenamento δ_{ij}. Esses parâmetros são computados durante um determinado período de tempo p. Porém, antes de computar o peso ótimo, é calculado o peso inicial da seguinte forma: $WD_{ij}^{p*} = \frac{A_{ij}^{p}}{T_{ij}^{p}} * C * \delta_{ij}$, onde C é uma constante utilizada para normalizar o valor inicial de WD_{ij}^{p*}.

Devido a flutuação dinâmica do ambiente de rede, o peso WD_{ij}^{p*} é reajustado para

melhorar a estabilidade no desempenho do *datanode* e computado como $WD_{ij}^{p} = \frac{WD_{ij}^{p*} - WD_{ij}^{p-1}}{K} + WD_{ij}^{p-1}$, onde K é um fator que tem o efeito de suavizar as eventuais oscilações de WD_{ij}^{p*}. Finalmente, considerando a utilização da CPU U_{ij}, o QoSC calcula o peso final de um *datanode* D_{ij} como $W_{ij} = WD_{ij}^{p} * (1 - (U_{ij})^2)^{\frac{1}{2}}$. Assim, dado o peso final W_{ij} de um *datanode* D_{ij} e considerando um arquivo com $m = N + M$ pedaços, o algoritmo denominado *Weight-based data placement algorithm* (WBD) estima o número teórico de pedaços de um certo arquivo TB_{ij}^{p} que será armazenado pelo D_{ij} da seguinte forma: $TB_{ij}^{p} = \lceil \frac{W_{ij}^{p}}{\sum_{i=1}^{M} \sum_{j=1}^{ND_i} W_{ij}^{p}} \rceil * m$.

Embora o QoSC utilize o algoritmo WBD para dar suporte ao balanceamento de carga, a heterogeneidade considerada pela abordagem é referente à largura de banda de rede. Apesar da modelagem da estratégia computar os pesos dos *datanodes* com base em diferentes tipos de recursos, o QoSC não considera a vazão ou latência dos dispositivos de armazenamento como parte da solução. A concorrência das operações e a variação do padrão da carga de trabalho não são exploradas para comprovar a eficiência da estratégia sobre essas condições.

3.4.1 Ceph

Ceph é um sistema de armazenamento distribuído confiável, escalável de fácil gerenciamento. O *Ceph* exporta diferentes interfaces que fornecem serviços de armazenamento de objetos, arquivos e blocos (CEPH, 2017). O *Ceph* é composto por três componentes principais: interface de acesso ao sistema de armazenamento, um sistema de armazenamento para armazenar dados e metadados e um *cluster* de servidores de metadados (WEIL *et al.*, 2006b).

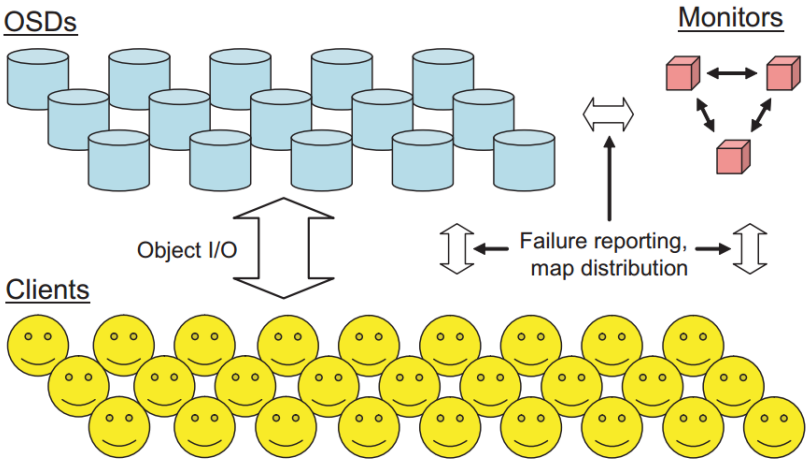


Figura 14 – Arquitetura do sistema de armazenamento baseado em objetos RADOS (Fonte: (WEIL *et al.*, 2007)).

O *Ceph* consegue prover alta escalabilidade, confiabilidade e autonomia através do sistema de armazenamento distribuído baseado em objetos chamado RADOS (WEIL *et al.*, 2007). A Figura 14 ilustra a arquitetura do sistema de armazenamento de objetos RADOS. O RADOS consiste de uma grande coleção de *Object Storage Devices (OSDs)* e um pequeno grupo de processos monitores responsáveis por gerenciar a adição e remoção de novos elementos no *cluster* de OSDs. Um OSD na arquitetura RADOS consiste de um dispositivo com CPU, memória RAM e uma interface de rede com um dispositivo de armazenamento anexado. OSDs e monitores executam processos autônomos que servem para gerenciar a distribuição dos dados e lidar contra falhas de componentes do sistema de armazenamento. O controle de quais OSDs estão em operação no sistema de armazenamento, assim como a distribuição dos dados entre os OSDs, são feitos através de uma estrutura de dados contendo o mapeamento do *cluster* (*cluster map*). O *cluster map* é gerenciado pelos monitores de sorte que a estrutura é replicada entres os OSDs e clientes que utilizam o serviço do sistema de armazenamento. No RADOS, os clientes expõem interfaces de armazenamento simples para aplicações, dessa forma, clientes podem realizar operações de I/O ao escrever ou ler objetos dos OSDs como mostra a Figura 14. Além disso, como todos os elementos da arquitetura possuem uma cópia do *cluster map*, qualquer mudança de estado no mapeamento da distribuição (p. ex. falha nos OSDs) ou evento que afete a distribuição dos dados, notificações são geradas para reportar mudanças no estado do *cluster*.

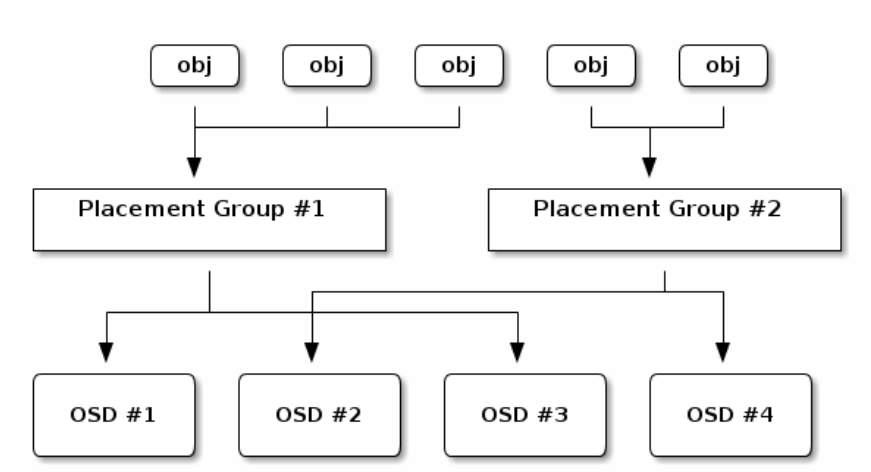


Figura 15 – Mapeamento de um conjunto de objetos para *placement groups* e *placement groups* para OSDs. (Fonte: (WEIL *et al.*, 2007)).

A estratégia de balanceamento de carga das requisições de leitura e escrita do sistema de armazenamento *RADOS* agrega as estratégias de alocação e replicação de dados do algoritmo CRUSH. Cada objeto armazenado no RADOS inicialmente é mapeado para a uma camada

intermediária denominada de *placement group* (PG). Um PG agrupa uma coleção lógica de objetos, onde as réplicas de um objeto em um PGs são mapeadas para uma coleção de OSDs, como mostra a Figura 15. Cada objeto é mapeado a um certo PG determinado pelo *hash* do objeto, o fator de replicação e uma máscara de bits que controla o número total de *PGs* no sistema de armazenamento. Para mapear réplicas de um objeto em um PG para um conjunto de OSDs, o RADOS delega ao algoritmo de alocação de dados CRUSH para distribuir os dados de forma pseudo aleatória entre dispositivos de armazenamento de um OSD, discutido anteriormente na seção 2.3. A camada de indireção composta pelos PGs permite ao *RADOS* rebalancear dinamicamente novos OSDs quando ocorrem falhas, são removidos ou adicionados. Em alto nível, o CRUSH funciona como uma função de *hash* consistente que ajusta dinamicamente apenas os dados necessários para manter uma distribuição balanceada.

Lembre que o algoritmo CRUSH consegue controlar o montante de dados relativo entre os dispositivos de armazenamento com o uso de pesos baseado em capacidade ou desempenho do dispositivo. Os pesos podem ser configurados externamente por um administrador do sistema de armazenamento em que o CRUSH está executando. Com a cópia do *cluster map* e o algoritmo CRUSH, um cliente do *Ceph* consegue realizar operações de leitura/escrita (através da interface LIBRADOS do RADOS) sobre um determinado objeto, localizando os devidos OSDs que armazenam as réplicas de um objeto no *cluster* de OSDs do RADOS. Independente do tipo de política de replicação, RADOS se baseia em cópia primária, onde clientes realizam as operações de leitura/escrita em um único OSD e depois a atualização é propagada para outros OSDs. Qualquer esquema de replicação configurado no RADOS garante que após as réplicas serem seguramente atualizas, a consistência das informações é preservada (WEIL *et al.*, 2007). Uma vez que todas as réplicas são atualizadas com sucesso, uma confirmação é retornada para o cliente.

3.4.2 OpenStack Swift Object Storage

O *OpenStack Swift Object Storage* ou simplesmente *OpenStack Swift* é um sistema de armazenamento baseado em objetos que tem o objetivo principal de fornecer alta disponibilidade, redundância, *throughput* e capacidade para armazenar grandes quantidades de dados. O *OpenStack Swift* é um sistema distribuído tolerante a falhas que foca na disponibilidade ao invés da forte consistência. Além disso, o *OpenStack Swift* foi projetado para escalar incontáveis servidores de armazenamento (SWIFT, 2017a). Conforme o teorema CAP (ABADI, 2012), um

sistema distribuído só consegue garantir duas dentre as três propriedades: consistência, disponibilidade e tolerância a partições. Dessa forma, a arquitetura do *OpenStack Swift* utiliza uma política de consistência eventual sem nenhum tipo de atraso de mecanismos de trava ou suporte a transações do tipo ACID. Assim, muitas operações concorrentes de leitura podem ser executadas rapidamente. Mesmo sobre alta demanda de operações de escrita simultâneas o sistema consegue completá-las rapidamente. A consistência eventual do *OpenStack Swift* garante que possíveis conflitos são resolvidos, porém com um certo atraso. O *OpenStack Swift* é um projeto de código aberto em constante evolução e possui fácil implantação em uma ambiente de nuvem (SWIFT, 2017b) sendo utilizado por várias empresas como Rackspace (RACKSPACE, 2017) e SwiftStack (SWIFTSTACK, 2017). Uma instalação do *OpenStack Swift* pode ser formada por componentes de servidores de baixo custo (*commodity hardwares*) e fornece a flexibilidade e facilidade do gerenciamento do armazenamento via software quando comparado com soluções proprietárias especializadas em *hardwares*. O *OpenStack Swift* é ideal para armazenar dados como *backups*, *logs*, imagens de *VMs*, dados de dispositivos móveis e outros tipos de dados não estruturados que crescem indefinidamente.

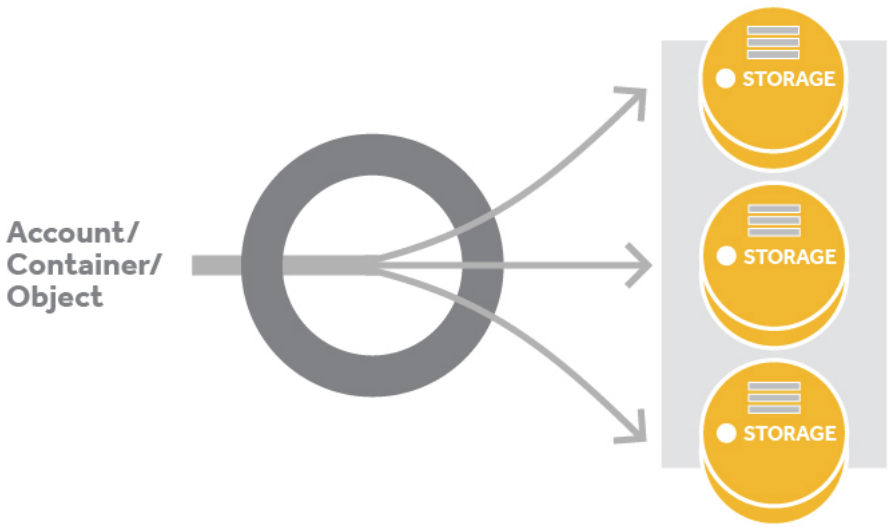


Figura 16 – Modelo organizacional de armazenamento do objetos no sistema de armazenamento *OpenStack Swift* (Fonte: (SWIFTSTACK, 2017)).

O *OpenStack Swift* possui um modelo de armazenamento de dados não estruturados, onde um objeto é organizado hierarquicamente em três camadas: *account, container* e *object*. O *account* é um nome único que representa uma área de armazenamento, não necessariamente identifica um usuário do sistema. O *container* é uma abstração de localização para facilitar na manipulação de objetos por meio de endereçamentos lógicos. Usuários podem definir

semânticas diferentes para *containers*. O *Object* é o nome do objeto propriamente dito que representa um objeto em particular (dado e metadado associados). A Figura 16 ilustra o modelo de armazenamento dos objetos no *OpenStack Swift*. A partir da combinação dessas três camada, o *OpenStack Swift* consegue gerar um identificador único para endereçar um objeto. Por exemplo, o identificador do $objeto_1$ é calculado a partir do *hash* do caminho do objeto: *conta1/conteiner1/objeto1*. Através das interfaces *Representational State Transfer* (REST) (RICHARDSON; RUBY, 2008) exportada pelo *OpenStack Swift* e o modelo organizacional de armazenamento de objetos, aplicações de usuários conseguem realizar operações de leitura, escrita e deleção pela internet através da URL do objeto.

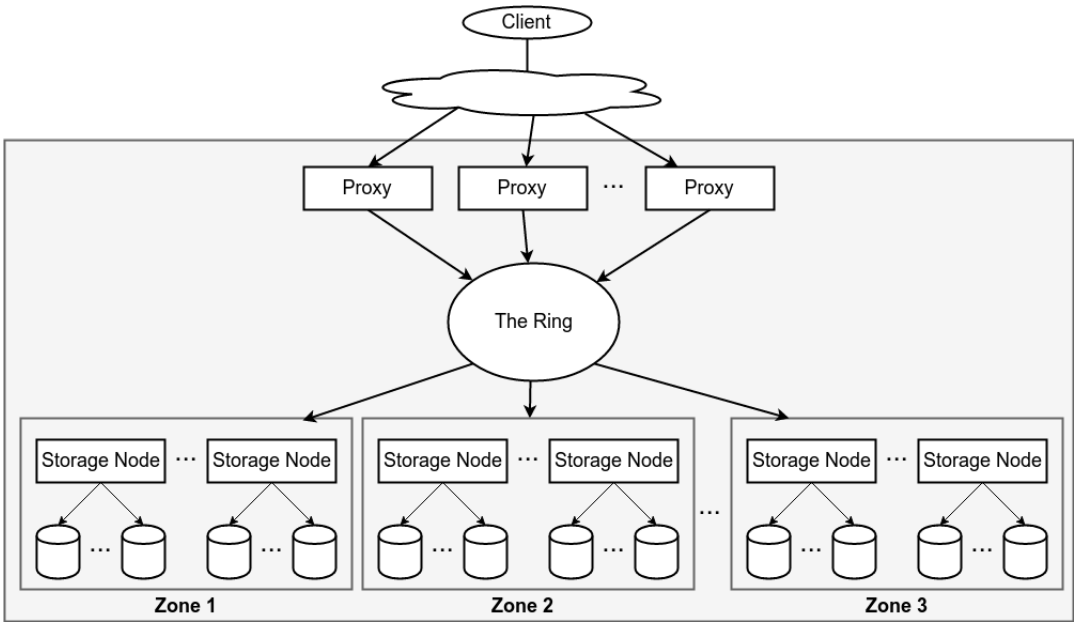


Figura 17 – Arquitetura do sistema de armazenamento de objetos *OpenStack Swift*.

A arquitetura do *OpenStack Swift* mostrada na Figura 17 é composta por um conjunto de servidores, dispositivos de armazenamento, e processos executados em segundo plano nos servidores (*proxy, account, container* e *object*). Servidores que executam processos do tipo *proxy* são chamados simplesmente de *proxys*. Basicamente, os *proxys* se comunicam com os clientes externos e coordenam as requisições de leitura e escrita destinadas aos dispositivos de armazenamento. Os servidores que executam processos do tipo *account, container* e *object* são chamados de *storage nodes* e são responsáveis por gerenciar um conjunto de dispositivos de armazenamento e objetos os quais ele armazena. Normalmente, *storage nodes* são gerenciados em zonas administrativas com o intuito de isolar possíveis falhas no sistema. Como o *OpenStack Swift* também suporta políticas de replicação de objetos para manter a durabilidade, disponi-

bilidade e alto *throughput* sobre concorrência de acesso, os *storage nodes* também executam processos responsáveis por manter a consistência e replicação dos objetos entre os dispositivos de armazenamento (ARNOLD, 2014). Os *storage nodes* e *proxys* utilizam uma versão adaptada do HASH consistente chamado de *ring* que possui suporte a replicação, similar ao discutido na seção 2.3. Os processos de *account*, *container* e *object* utilizam uma cópia do *ring* para prover acesso rápido às respectivas informações de interesse entre os dispositivos de armazenamento.

O usuário consegue controlar parte da alocação de dados do sistema como uma caixa preta através do mecanismo de alto nível de abstração fornecido pelo *ring* do *OpenStack Swift*. O usuário define o peso de um dispositivo de armazenamento e esse peso é comparado com os demais pesos registrados no *ring* para calcular o número total de partições que devem ser atribuídas a um dispositivo de armazenamento. Pesos com valores maiores refletem em maior número de partições, consequentemente, mais objetos (réplicas) e requisições são atribuídos ao dispositivo (ARNOLD, 2014). Dessa forma, a estratégia de balanceamento de carga padrão sugerida pela documentação do *OpenStack Swift* atribui pesos aos dispositivos de forma estática (SWIFT, 2017a) definida manualmente pelo administrador do sistema. Resumidamente, a estratégia do *Openstack Swift* consiste em atribuir um peso a um dispositivo de armazenamento proporcional a sua capacidade de armazenamento. Por exemplo, um dispositivo de armazenamento com espaço de armazenamento igual a 1TB e outro com 2TB devem ter pesos similares a 1000 e 2000, respectivamente. A estratégia de balanceamento de carga padrão do *OpenStack Swift* consegue distribuir o volume de dados proporcionalmente à capacidade de armazenamento dos dispositivos de armazenamento, mesmo que esses sejam heterogêneos em relação a essa capacidade. Entretanto, a estratégia padrão do *OpenStack Swift* atribui pesos aos dispositivos de armazenamento de forma estática e não considera nenhuma característica de desempenho.

Os servidores no *OpenStack Swift* gerenciam requisições de leitura e escrita de maneiras diferentes. Em uma requisição de leitura, uma réplica pode ser escolhida entre as disponíveis nos *storage nodes* seguindo algum critério de balanceamento (ALMEIDA *et al.*, 2016). A estratégia de seleção de réplicas padrão do *OpenStack Swift* consiste em construir uma lista contendo a localização das réplicas existentes de um determinado objeto e em seguida escolher aleatoriamente a réplica que se encontra no topo lista. Esse processo é repetido para cada requisição do tipo leitura. Nas requisições do tipo escrita, o *OpenStack Swift* utiliza um algoritmo baseado em quórum para confirmar a conclusão da operação. Quando um usuário faz uma requisição de escrita de um objeto, o *proxy* que atende a requisição utiliza o *ring* para

saber em quais *storage nodes* e dispositivos de armazenamento inserir as réplicas de um objeto. Logo em seguida, o *proxy* encaminha o objeto para os devidos *storage nodes*. O *proxy* precisa apenas da confirmação da metade do número total de réplicas mais uma por parte dos *storage nodes* para confirmar com sucesso a requisição de escrita de um objeto. Uma vez confirmado o sucesso da operação por parte dos *storage nodes*, o *proxy* envia o estado da operação para o usuário (SWIFT, 2017a).

3.5 Discussão

Este capítulo explorou o projeto arquitetural das principais estratégias de balanceamento de carga para sistemas de armazenamento distribuídos utilizados no ambiente de nuvem. Foram destacados os principais componentes da arquitetura dos sistemas de armazenamento e como eles interagem com os algoritmos de balanceamento. Note que as diferenças e semelhanças entre estratégias de balanceamento de carga acontecem por que elas consideram diferentes aspectos ao projetarem suas soluções. A seguir, será discutido algumas dessas características que aproximam ou distanciam as soluções quando focadas em algum critério específico.

Estratégias de balanceamento de carga como proposta em (WANG *et al.*, 2015) e ALDM não são intrusivas ao sistema de armazenamento, ou seja, exigem pouca ou nenhuma alteração do código fonte e interfaces do sistema de armazenamento para integrar a solução de balanceamento. Os algoritmos de alocação de dados *CRUSH* e o *DHT* consistente com suporte a replicação também não são intrusivos e estes são os componentes principais das estratégias de balanceamento de carga do *Ceph RADOS* e *OpenStack Swift*, respectivamente. Além disso, esses algoritmos fornecem a flexibilidade de controlar o montante de dados associados aos dispositivos de armazenamento através de pesos vinculados a eles de forma não intrusiva. Por outro lado, soluções como QoSC e (HSIAO *et al.*, 2013) são dependentes da plataforma e menos flexíveis em sua implementação e integração ao sistema que se deseja implantar.

Vale ressaltar que os algoritmos apresentados neste capítulo tem em comum o fato de utilizarem a migração de dados como abordagem principal para solucionar o problema de balanceamento de carga em sistemas de armazenamento distribuído em nuvem, com exceção da solução proposta em (WANG *et al.*, 2015) que explora a migração de VMs para regular o acesso aos dados. Entretanto, soluções que utilizam funções de *hash* consistente como as integradas ao sistemas de armazenamento de objetos *Ceph* e *OpenStack Swift* se destacam das demais em relação ao reajuste da distribuição dos dados, pois evitam a migração de grande volume de

dados quando modificado o estado dos dispositivos (falha, adição ou remoção) do sistema de armazenamento.

Em relação a replicação de dados, o *framework* QoSC é o único entre as estratégias aqui exploradas que propõe uma nova política de redundância em conjunto com o algoritmo de balanceamento. Dessa forma, o QoSC consegue fortalecer a disponibilidade dos dados e melhorar a utilização dos recursos através do acesso às réplicas. Entretanto, sistemas de armazenamento como *OpenStack Swift*, *Ceph* e *HDFS* já fornecem algum tipo de política de replicação de dados. A solução proposta em (HSIAO *et al.*, 2013) utiliza a replicação dos fragmentos de um arquivo como política de redundância de dados do sistema HDFS, mas a solução não trata a questão de réplicas redundantes entre os dispositivos de armazenamento.

É importante destacar que cada estratégia considera hipóteses sobre o padrão de acesso ou componentes da carga de trabalho. As estratégias ALDM, (WANG *et al.*, 2015) e *CRUSH* são as únicas que consideram aspectos de acesso aos dados, enquanto o restante das estratégias apresentadas neste capítulo assume algum tipo de acesso com padrão uniforme ou a composição da carga de trabalho não muda muito ao longo do tempo. Especificamente, as estratégia ALDM e (WANG *et al.*, 2015) exploram o padrão de acesso a dados populares com o objetivo de otimizar a intensidade da carga com a utilização dos recursos. Apesar do algoritmo *CRUSH*, utilizado pelo RADOS, não lidar diretamente com dados populares, o *CRUSH* tenta diversificar o acesso entre as réplicas de um objeto através de funções de *hash* pseudo aleatórias.

Além da carga de trabalho, a ineficiência em alguma camada específica do sistema pode impactar na degradação de seu desempenho. Assim, torna-se um grande desafio considerar o gerenciamento de um conjunto específico de recursos, componentes e elementos do sistema de armazenamento para que estes não se tornem gargalos de desempenho. Por exemplo, (WANG *et al.*, 2015) enfatiza a utilização dos recursos computacionais virtualizados como CPU e memória primária em um ambiente homogêneo, enquanto o QoSC foca principalmente na instabilidade e heterogeneidade do ambiente de rede. Já a estratégia ALDM atribui o custo do compartilhamento de rede, recursos de desempenho e armazenamento dos dispositivos para a modelagem de sua solução.

Embora estratégias como ALDM considerarem o desempenho de dispositivos de armazenamento em termos de operações de *Input/Output* (I/O), todas as estratégias discutidas neste capítulo não consideram um ambiente de armazenamento com dispositivos heterogêneos em relação ao desempenho e a mudança de certos componentes da carga de trabalho. Dessa

forma, a estratégia BACOS proposta nesse trabalho surge para atender tais necessidades. BACOS é uma estratégia de balanceamento de carga não intrusiva que explora a heterogeneidade dos dispositivos de armazenamento para melhorar o desempenho geral do sistema de armazenamento. BACOS é capaz de se adaptar a certos parâmetros da carga de trabalho como o número de clientes concorrentes e a taxa de requisições de leitura/escrita. Mais detalhes sobre a estratégia BACOS serão apresentados no capítulo 4.

A Tabela 3 apresenta um breve comparativo das principais características consideradas pelas estratégias de balanceamento de carga discutidas neste capítulo, incluindo a estratégia BACOS, proposta neste trabalho.

Tabela 3 – Análise comparativa do trabalhos relacionados.

Algoritmo	**Abordagem**	**Hotspot**	**Storage System**	**Análise da carga**	**Principal gargalo considerado pelo algoritmo**	**Recursos considerado pelo algoritmo**	**Heterogeneidade**
HSIAO 2013	Migração de dados	Não se aplica	File system	Homogênea	Capacidade de armazenamento	Capacidade dos dispositivos de armazenamento	Capacidade dos dispositivos de armazenamento
WANG 2015	Migração de VMs	Se aplica	Object Storage	Heterogênea	Utilização dos recursos computacionais (Camada de virtualização)	CPU, Memória e Rede	-
ALDM	Migração de dados	Se aplica	File system	Heterogênea	Utilização por tipo de recursos	Rede, I/O e capacidade dos dispositivos de armazenamento	-
QoSC	Migração e redundância de dados	Não se aplica	File system	Homogênea	Instabilidade de rede	CPU, Rede e capacidade dos dispositivos de armazenamento	Rede
Ceph CRUSH	Migração e redundância de dados	Parcialmente	Object storage	Heterogênea	falha em dispositivos de armazenamento	Armazenamento ou desempenho dos dispositivos	Armazenamento ou desempenho dos dispositivos
OpenStack Swift	Migração e redundância de dados	Não se aplica	Object storage	Homogênea	Capacidade de armazenamento	Capacidade dos dispositivos de armazenamento	Capacidade dos dispositivos de armazenamento
BACOS	Migração e redundância de dados	Não se aplica	Object storage	Heterogênea	Desempenho dos dispositivos de armazenamento	I/O e capacidade dos dispositivos de armazenamento	I/O dos dispositivos de armazenamento

3.6 Conclusão

Os principais trabalhos referentes ao estado da arte sobre balanceamento de carga em sistemas de armazenamento distribuídos e relacionados ao tema desta dissertação foram

apresentados nesse capítulo. Foram descritas as principais características consideradas no projeto arquitetural das soluções, assim como suas contribuições e limitações. Finalmente, foram comparados os trabalhos relacionados. A maioria das propostas apresentadas no capítulo não exploram a heterogeneidade do desempenho dos dispositivos de armazenamentos em conjunto com a análise da carga de trabalho.

4 UMA ESTRATÉGIA DE BALANCEAMENTO DE CARGA PARA SISTEMAS DE ARMAZENAMENTO DE OBJETOS EM NUVEM

4.1 Introdução

Neste trabalho, é proposto uma estratégia de balanceamento de carga para sistemas de armazenamento em nuvem. Em particular, é proposto uma estratégia de balanceamento de carga para sistemas de armazenamento de objetos (*object storages*) denominado *Dynamic Load Balancing Strategy for Cloud Object Storage* (BACOS) (PAULA *et al.*, 2017). BACOS tem o objetivo principal de balancear a carga, requisições de leitura e escrita de objetos geradas pelas aplicações dos usuários, por meio da distribuição dos dados no sistema proporcionalmente às requisições e ao desempenho heterogêneo dos dispositivos de armazenamento de um *object storage*. A tomada de decisão da estratégia BACOS é baseada no conhecimento prévio e parcial sobre o padrão de carga do sistema de armazenamento utilizado em produção. Assim, BACOS faz a redistribuição das requisições e dados dinamicamente de acordo com a mudança da carga de trabalho para assegurar que o sistema está operando com o melhor desempenho possível observado. BACOS gerencia os dados de forma eficiente buscando mitigar o impacto na migração de dados e no atendimento de requisições. Ademais, BACOS é uma estratégia não intrusiva, ou seja, não necessita modificação do código fonte interno do sistema de armazenamento dado que a estratégia se aproveita de interfaces de alto nível de abstração, expostas pelo sistema, para controlar a utilização dos recursos.

4.2 Arquitetura

A arquitetura dos *object storages* modernos é projetada com a intenção de fornecer segurança, compartilhamento de dados entre diferentes plataformas, alto desempenho e escalabilidade em relação ao número de clientes e dispositivos do sistema (XU *et al.*, 2016). Os projetistas de sistemas de armazenamento definem as características mais importantes da arquitetura considerando a relação custo-benefício em termos de escalabilidade, disponibilidade, desempenho, capacidade de armazenamento e assim por diante. Deste modo, para o bom funcionamento da estratégia de balanceamento de carga BACOS é considerado uma arquitetura de sistema de armazenamento escalável, com suporte à replicação parcial (NUAIMI *et al.*, 2013), protocolo de consistência eventual e com fácil acesso aos dados através de interfaces bem definidas, adequada para implantação do BACOS. Um exemplo desse tipo de arquitetura é ilustrado na Figura 18.

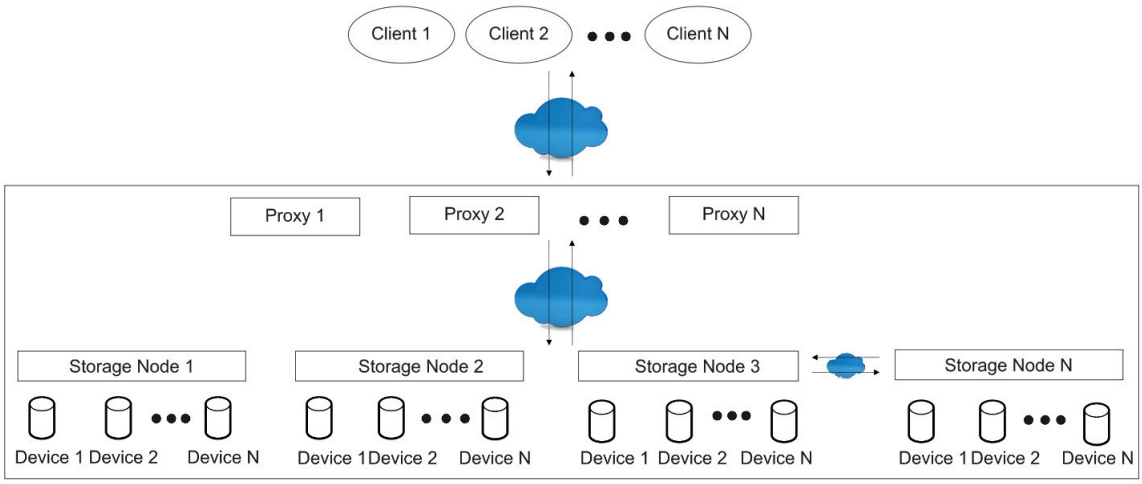


Figura 18 – Arquitetura de um *object storage* utilizada para a implantação da estratégia de balanceamento BACOS.

A estratégia BACOS considera uma arquitetura de *object storage* formada por três camadas: A primeira camada é composta por um conjunto de servidores controladores de requisições denominados de (*proxy nodes*). Um *proxy node* tem o papel de receber requisições dos clientes e direcioná-las para um ou mais servidores de armazenamento (*storage nodes*); a segunda camada é composta por servidores de armazenamento. Um servidor de armazenamento é responsável por gerenciar vários dispositivos de armazenamento como HDDs e SSDs; e a terceira camada é composta por dispositivos de armazenamento com interfaces de objetos, assim, compondo os OSDs do *object storage*. Recorde que o modelo de OSDs foi discutido na seção 2.2.1. Requisições recebidas por um servidor de armazenamento são repassadas para um determinado dispositivo de armazenamento (veja a Figura 18). Na arquitetura em questão, os clientes podem enviar requisições de leitura, escrita ou remoção. Porém, nesse trabalho, é considerado apenas operações de leitura e escrita. Como os *object storages* são sistemas escaláveis, eles são mais recomendados para aplicações que constantemente necessitam armazenar e recuperar uma grande quantidade de objetos, em consequência, realizam mais operações de leitura e escrita do que de deleção de objetos.

O projeto arquitetural do sistema de armazenamento que é apresentado na Figura 18, prioriza o desempenho geral do sistema e a disponibilidade dos dados. Assim, a arquitetura do sistema de armazenamento deve prover mecanismos de acesso rápido às informações entre servidores e dispositivos de armazenamento e suporte a alguma política de replicação de objetos. Por esse motivo, BACOS utiliza o DHT consistente com suporte a replicação como principal estrutura de dados distribuída para controlar o número de réplicas dos objetos, o acesso aos dados e o número de chaves para dispositivos de armazenamento como explicado nas seções 2.3

e 3.4.2. No DHT consistente com suporte a replicação, um dispositivo de armazenamento recebe um peso baseado em algum critério de armazenamento e ou desempenho. Esse critério faz com que o número de chaves destinado a um determinado dispositivo de armazenamento defina a quantidade de requisições e de réplicas de objetos que devem ser destinadas aos dispositivos de armazenamento. Conforme mencionado na seção 2.3, uma nova configuração de pesos ocasiona não só em migração dos dados existentes mas também em alteração na disposição dos dados de novas requisições. Por isso, BACOS considera o impacto da atualização da configuração dos pesos a fim de evitar tráfego demasiado de rede e indisponibilidade do serviço de armazenamento.

Pensando em facilitar o gerenciamento dos elementos da arquitetura de um sistema de armazenamento escalável, foi escolhido o modelo arquitetural de *webservice* REST (RICHARDSON; RUBY, 2008). Esse modelo permite que diferentes tipos de servidores e dispositivos de armazenamento com recursos heterogêneos possam ser facilmente integrados ao sistema de armazenamento através de uma API bem definida. Além disso, o modelo de *webservice* possibilita administradores de sistema ou estratégias de balanceamento de terceiros regular remotamente parâmetros e recursos do sistema quando necessário. Essas características, sobre a comunicação entre os servidores dos sistema de armazenamento são fundamentais para a estratégia BACOS, pois ela consome interfaces REST API para tomada de decisão sobre o balanceamento de carga do sistema. Assim, BACOS consegue manipular dinamicamente e de forma não intrusiva a distribuição das requisições para os dispositivos de armazenamento através de seus pesos a fim de manter o melhor desempenho do sistema, como será visto nas próximas seções.

4.3 Visão geral

Esta seção apresenta a visão geral da estratégia de balanceamento de carga BACOS. A tabela de símbolos desta dissertação sumariza as principais definições dos conceitos utilizados pela estratégia BACOS. Diante disso, a solução foi dividida em três etapas com responsabilidades diferentes como ilustrado na Figura 19. A primeira etapa, chamada de classificação dos dispositivos de armazenamento, tem como objetivo medir o desempenho de cada dispositivo de armazenamento e atribuir dispositivos com desempenhos similares a um mesmo grupo. A segunda etapa, chamada de medição do desempenho do sistema de armazenamento, busca medir e registrar o desempenho geral do sistema de armazenamento por meio da variação dos parâmetros da carga de trabalho e os parâmetros do sistema que definem a distribuição das

requisições e os dados armazenados pelos dispositivos de armazenamento utilizando a interface dos pesos dos dispositivos de armazenamento fornecido pelo DHT. Neste trabalho, o desempenho geral do sistema é definido pelo tempo de resposta e a taxa de sucesso/insucesso das requisições realizadas ao sistema de armazenamento. A terceira e última etapa, denominada de método de balanceamento de carga, consiste em encontrar a melhor distribuição dos dados entre os dispositivos de armazenamento e aplicá-la em produção com o intuito balancear as requisições de leitura e escrita para maximizar o desempenho geral do sistema. BACOS ainda evita sobrecarregar recursos e a indisponibilidade do serviço mediante à avaliação da migração de dados e a taxa de insucesso das requisições. Os detalhes de cada etapa serão discutidos nas próximas seções.

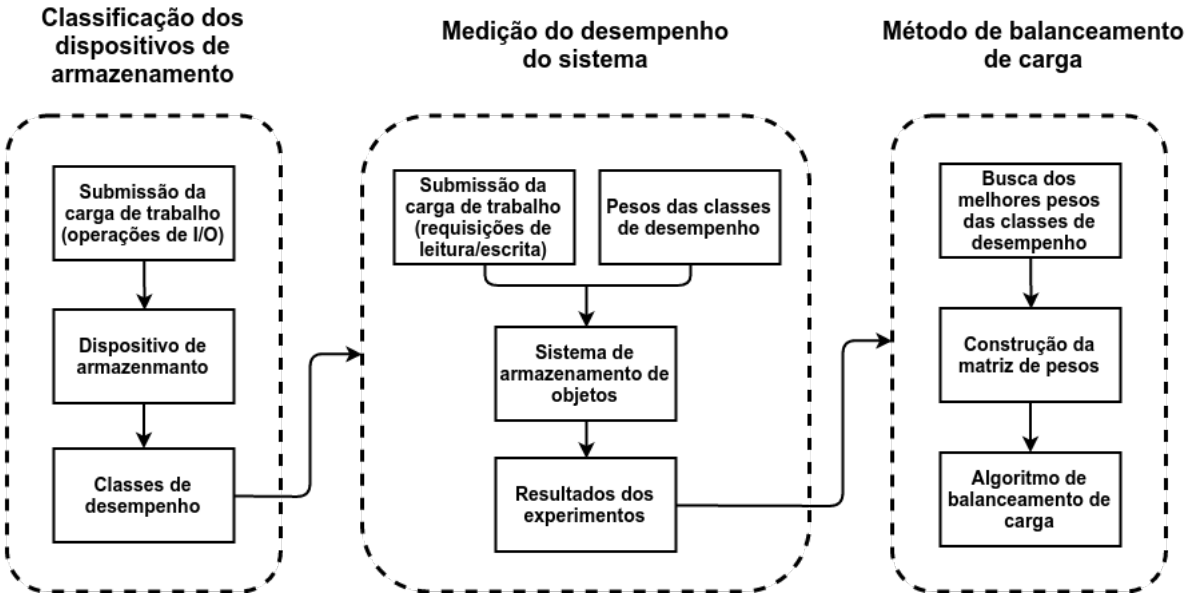


Figura 19 – Visão geral do fluxo de execução da estratégia BACOS.

4.4 Classificação dos dispositivos de armazenamento

Na primeira etapa, dispositivos de armazenamento são classificados com base no seu desempenho, pois é interessante utilizar os dispositivos de armazenamento mais rápidos o máximo possível, mas sem sobrecarregá-los para não sobrecarregar o sistema de armazenamento como um todo e evitar comportamentos inesperados. BACOS busca equilibrar a carga do sistema atribuindo mais requisições a dispositivos com maior desempenho e atribuir menos requisições a dispositivos mais lentos. Entretanto, dispositivos de armazenamento populares como HDD, SSD e PCM (KIM *et al.*, 2014) possuem componentes e interfaces distintas que podem afetar o

desempenho final do dispositivo. O tipo de material, densidade de armazenamento e a tecnologia de persistência dos dados são algumas características que podem determinar o desempenho final do dispositivo. Logo, as diferentes interfaces de transferência de dados, matéria prima e tecnologias dos componentes dos dispositivos de armazenamento torna a tarefa de definir com precisão o desempenho desses dispositivos bastante difícil. Para facilitar a descoberta do desempenho de um dispositivo de armazenamento ao processar operações de entrada ou saída, foi decidido utilizar como métrica de desempenho para dispositivos de armazenamento o *Input/Output Operations Per Second (IOPS)* (BOSE *et al.*, 2017). O IOPS é uma métrica bastante utilizada na indústria de dispositivos de armazenamento e consegue passar a intuição do quão veloz um dispositivo consegue processar operações de entrada e saída. O IOPS é uma métrica que permite generalizar as características de um determinado dispositivo de armazenamento que estão correlacionadas ao desempenho assim como o tipo de requisição que o dispositivo processa, seja ela de leitura, escrita ou exclusão de dados.

Antes de um dispositivo de armazenamento ser inserido em um sistema de armazenamento, o administrador deve medir o IOPS máximo do dispositivo para descobrir o número de operações de entrada e saída que o dispositivo consegue suportar, caso o fabricante não tenha disponibilizado essa informação. Portanto, considere um dispositivo de armazenamento $d_k \in \{d_1, ..., d_Z\}$, onde Z é o número total de dispositivos existentes no sistema de armazenamento. Foi utilizado uma ferramenta de *benchmark* específica, como FIO (AXBOE, 2017), para estressar o dispositivo e verificar o seu IOPS máximo. É interessante que os parâmetros de carga utilizados para medir o IOPS dos dispositivos reflitam no padrão de carga comumente utilizado pelas aplicações consumidoras do sistema de armazenamento. Se a carga de trabalho não é conhecida, é recomendado utilizar uma carga padrão indicado pela documentação da ferramenta FIO como um meio de descobrir o IOPS máximo do dispositivo. A carga possui um tamanho total de $\geq 1GB$ e é realizado operações de leitura/escrita com acesso aleatório apenas para o dispositivo de armazenamento com uma taxa de leitura/escrita igual a $50\%/50\%$, respectivamente, onde o tamanho dos blocos requisitados é igual a $4KB$. Ao utilizar esse tipo de carga, assume-se que operações de leitura e escrita possuem o mesmo impacto no desempenho final do dispositivo. Porém, para uma medição mais precisa do IOPS de um dispositivo de armazenamento é recomendado a utilização e comparação com outras ferramentas como o *Buttress* (ANDERSON *et al.*, 2004). A descoberta do IOPS máximo de um dispositivo de armazenamento é realizado apenas uma vez, no momento em que o dispositivo é inserido no sistema.

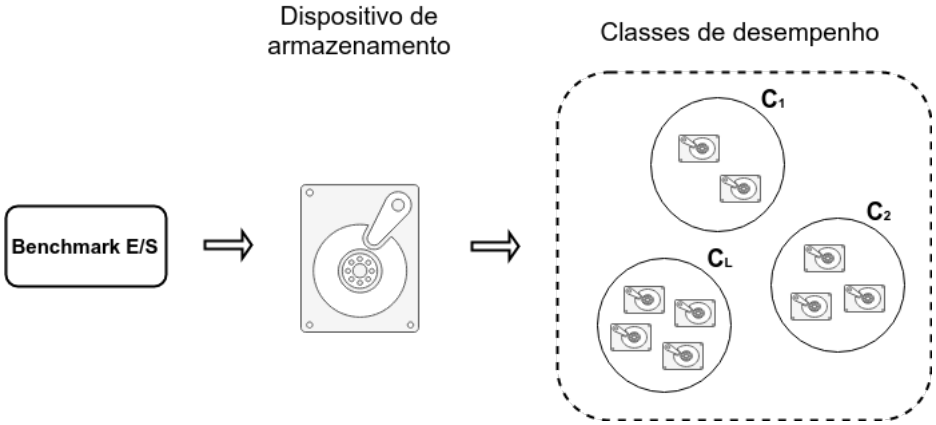


Figura 20 – Classificação dos dispositivos de armazenamento em classes de desempenho.

Uma vez que foi medido o IOPS máximo de cada dispositivo de armazenamento por meio da submissão da carga de trabalho para um dispositivo e coleta dos valores de IOPS, agrupam-se os dispositivos em classes de desempenhos C_j, para $1 \leq j \leq L$, onde L é o número total de classes. Uma classe de desempenho é um conjunto de dispositivos de armazenamento com desempenhos similares. As classes de desempenho são construídas definindo-se limiares de valores para IOPS T_{C_j} por classe C_j. Assume-se, sem perda de generalização, que $T_{C_1} < T_{C_2} < ... < T_{C_L}$. Como o sistema de armazenamento lida com dispositivos heterogêneos em relação ao desempenho, considera-se que existem pelo menos duas classes de desempenho distintas no sistema de armazenamento. Assim, um dispositivo d_k é atribuído a uma classe de desempenho C_j se o valor de IOPS do dispositivo d_k for maior que o valor limiar T_{C_j} da classe C_j, onde j é o maior $j \in \{1,2,...,L\}$. A Figura 20 ilustra o processo de classificação dos dispositivos em classes de desempenho. Adota-se por convenção, ao construir as classes de desempenho, que dispositivos de armazenamento da classe C_i são mais rápidos que os dispositivos da classe C_j caso $1 \leq j \leq i \leq L$.

A medição do IOPS de um dispositivo de armazenamento é um processo individual que pode apresentar imprecisões dos valores obtidos por conta da escolha específica dos parâmetros da carga de trabalho e características particulares dos dispositivos de armazenamento. Entretanto, o mais importante nesse processo é encontrar a vazão máxima suportada dos dispositivos sobre as mesmas condições para transparecer o quão rápido um dispositivo consegue processar operações em certos níveis de granularidade, rotulados pelas classes de desempenho. Na próxima etapa do BACOS, o desempenho dos dispositivos de armazenamento, por meio das classes de desempenho, será explorado em conjunto à caracterização da carga de trabalho para melhorar o desempenho geral do sistema de armazenamento. Essa etapa serve apenas para

agrupar dispositivos com desempenhos similares em classes de desempenho. Ela não inclui a análise do custo-benefício do número de dispositivos de alto desempenho que deve existir no sistema de armazenamento em relação aos dispositivos de baixo custo. Contudo, essa etapa pode ser complementada com soluções como a apresentada em (XU *et al.*, 2015) que exploram a proporção de dispositivos de baixo custo e alto desempenho no sistema de armazenamento como uma maneira de minimizar os custos de aquisição e maximizar o desempenho geral do sistema. O número de classes de desempenho não é definido pela estratégia BACOS, pois o mesmo pode ser definido por um administrador de sistema através do limiar T_C para uma classe de desempenho de acordo com a heterogeneidade do ambiente.

BACOS tira proveito do desempenho dos dispositivos atualmente configurados no sistema. Logo, as classes de desempenho facilitam o gerenciamento de *pools* de dispositivos de armazenamento, reduz a complexidade da variação dos parâmetros utilizados pela estratégia de balanceamento BACOS e ajuda a escalar a solução.

4.5 Medição do desempenho do sistema

Na segunda etapa, a estratégia BACOS tem o propósito de gerar uma base de treino contendo informações sobre características do padrão de acesso das aplicações que estão consumindo o serviço de armazenamento, informações sobre a distribuição das requisições destinadas aos dispositivos de armazenamento configuradas como parâmetro do sistema e o comportamento do desempenho geral do sistema de armazenamento. Para tal propósito, considere dois sistemas de armazenamento, um ambiente de produção e um ambiente simulado de produção. Conceitualmente, o ambiente de produção é o sistema principal que é utilizado por um provedor de serviço para oferecer serviços de armazenamento para os seus clientes com máxima disponibilidade. O ambiente simulado é uma cópia isolada do sistema de produção, utilizado apenas pelos administradores do sistema para executar experimentos exploratórios baseados no padrão de carga do sistema de produção. Na prática, o ambiente de produção simulado é o próprio ambiente de produção, onde experimentos são executados e métricas sobre a carga e componentes do sistema são coletadas enquanto o sistema de armazenamento fornece o serviço de armazenamento para seus usuários. Porém, nesse tipo de ambiente, o usuários podem perceber uma queda no desempenho devido a coleta das métricas e mudanças de configuração dos parâmetros do sistema.

Nesta etapa, o objetivo principal é gerar uma base de treino contendo as configura-

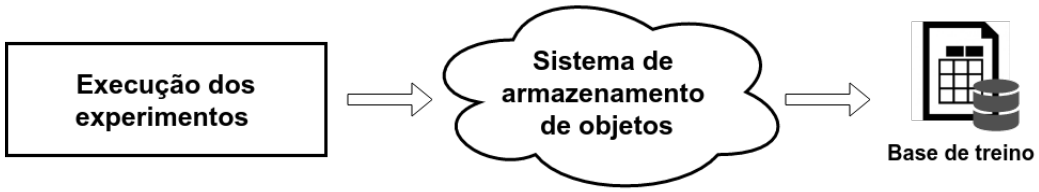


Figura 21 – Criação da base de treino da estratégia de balanceamento de carga BACOS.

ções dos experimentos e os resultados relacionados ao desempenho do sistema. Posteriormente, na próxima etapa da estratégia BACOS, esses dados serão utilizados para descobrir qual configuração dos parâmetros do sistema de armazenamento que melhoram o seu desempenho geral, dado um padrão de carga de trabalho. A Figura 21 mostra o fluxo de execução dos experimentos para a criação da base de treino do BACOS. De um modo geral, é executado um conjunto de experimentos variando parâmetros relacionados à carga de trabalho e parâmetros vinculados ao sistema de armazenamento de objetos. Como resultado, é registrado o desempenho observado do sistema (tempo de resposta, taxa de sucesso e insucesso das requisições) em cada experimento. Os parâmetros do sistema de armazenamento são referenciados como os parâmetros dos pesos responsáveis por definir a distribuição das requisições e dados entre os servidores de armazenamento. Esses experimentos são gerados utilizando um ambiente simulado de produção para evitar qualquer queda da qualidade do serviço, caso seja executado diretamente no sistema em produção. Uma vez que a base de treino é gerada, ela representa o conhecimento prévio sobre como a carga afeta o desempenho do sistema de armazenamento, visto que a carga de teste, provinda do ambiente simulado de produção, é uma cópia fiel da carga real utilizada no sistema em produção.

Como discutido na seção 4.2, BACOS utiliza uma arquitetura que usufrui de interfaces de alto nível de abstração que são exportadas pelo sistema de armazenamento para configurar os pesos que refletem as requisições e os dados destinados aos dispositivos de armazenamento. Os pesos individuais dos dispositivos são configurados por meio do DHT consistente com suporte à replicação. O conceito dos pesos dos dispositivos de armazenamento pode ser estendido para as classes de desempenho apresentadas na seção anterior. Logo, $W = (w_1, w_2, ..., w_L)$ é definido como o conjunto dos pesos das classes de desempenho de modo que $w_1 + w_2 + ... + w_L = 1$, onde o peso $w_j \in W$ representa a proporção de dados e requisições que uma classe de desempenho C_j controla. Consequentemente, w_j também define a proporção de dados e requisições de um dispositivo de armazenamento d_k, uma vez que cada dispositivo recebe o mesmo peso de sua classe de desempenho.

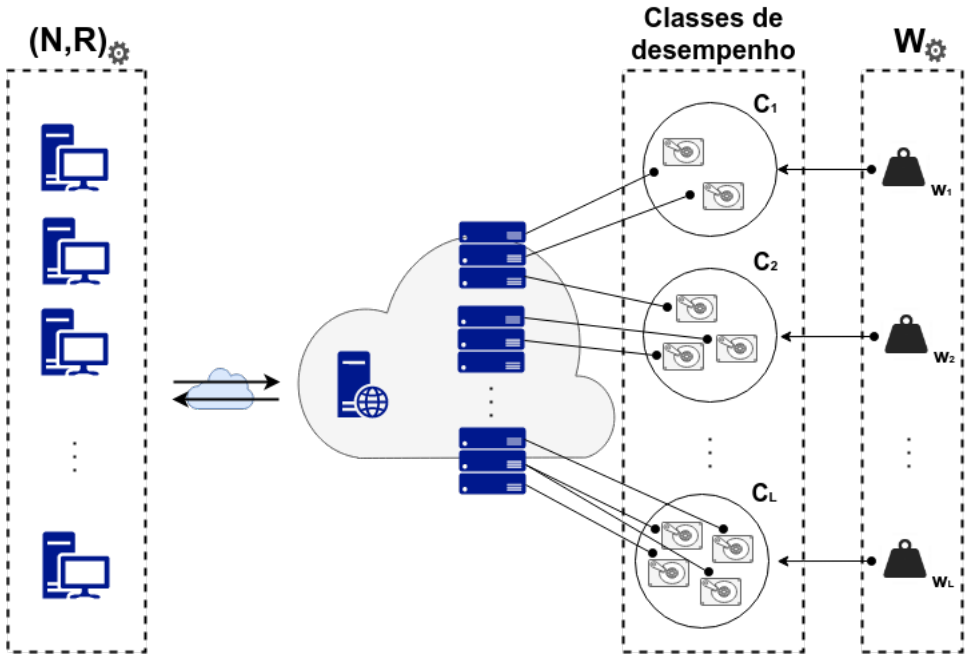


Figura 22 – Configuração dos parâmetros (N,R,W) de um experimento no ambiente simulado de produção.

Em cada experimento, são considerados o número de clientes concorrentes N e a taxa de leitura R como parâmetros da carga de trabalho, representados pela configuração (N,R). Como o parâmetro R representa a taxa de leitura em porcentagem, ele apresenta valores entre 0 e 100. Para diminuir a quantidade de parâmetros gerenciados pela estratégia BACOS, assume-se apenas a R, N e a taxa de escrita como os parâmetros da carga de trabalho, de modo que a taxa de escrita pode ser derivada de R calculando $100-R$. Apesar de ser utilizado N e R como os parâmetros da carga de trabalho em cada experimento, a estratégia BACOS não está limitada a esses parâmetros. O conjunto dos pesos das classes de desempenho W é considerado o único parâmetro configurável do sistema utilizado para gerar a base de treino do BACOS. Dados os parâmetros da carga de trabalho (N,R) quanto ao parâmetro do sistema W apresentado anteriormente, a entrada de um experimento é dada pela combinação dos parâmetros (N,R,W). A cada configuração específica da carga de trabalho (N,R) é variado os pesos das classes de desempenho pertencentes ao conjunto W. A Figura 22 ilustra a configuração dos parâmetros (N,R,W) para um certo experimento no ambiente simulado de produção. Note que em um experimento com parâmetros (N,R,W), os parâmetros da carga de trabalho (N,R) são configurados externos ao sistema de armazenamento, enquanto que o parâmetro W é configurado diretamente no sistema. Cada experimento tem duração total de 5 minutos. Devido aos possíveis ruídos, foi desconsiderado $30s$ iniciais e finais do tempo total do experimento. Os valores

específicos da variação dos parâmetros (N,R,W) em um experimento serão apresentados no capítulo 5 que abordará a análise de desempenho da estratégia BACOS.

Para cada experimento realizado, configurados com valores de (N,R,W), foi coletado o tempo médio de resposta RT das requisições realizadas ao sistema, o número total de requisições REQ_{total} e o número total de requisições com erros REQ_{error} reportado pelo sistema de armazenamento quando requisições não são atendidas durante o tempo de duração do experimento. A taxa de sucesso e insucesso são conceitos complementares, visto que a taxa de sucesso é compreendida como a proporção de requisições de leitura e escrita que o sistema de armazenamento conseguiu atender como o esperado, realizando a operação solicitada e sem reporta nenhum tipo de problema na requisição. Logo, a partir do número total de requisições REQ_{total} e o número total de requisições com erros REQ_{error}, a taxa de insucesso é definida e calculada como $I = \frac{REQ_{error}}{REQ_{total}} \in [0,1]$. Como a taxa de sucesso é a taxa complementar da taxa de insucesso e vice-versa, por consequência, a taxa de sucesso pode ser obtido a partir de I como $|1-I|$.

A taxa de insucesso I foi observada como uma maneira de entender a relação das requisições não atendidas com o gerenciamento dos recursos e o desempenho do sistema de armazenamento. Uma grande quantidade de requisições com erros em relação a quantidade total de requisições reflete em grande parte, mau gerenciamento dos recursos do sistema de armazenamento, visto que as requisições podem estar desbalanceadas em relação à carga que os componentes do sistema conseguem responder. Consequentemente, o sistema acaba recusando algumas requisições e postergando outras que estão aguardando na fila de espera do sistema. A falta de recursos generalizada do sistema também provoca um número elevado de erros. Quando o sistema não possui recursos suficientes para processar mais requisições, dificilmente um algoritmo de balanceamento conseguirá melhorar o desempenho geral do sistema apenas com os recursos existentes. Nesse caso, uma estratégia dinâmica de provisionamento e alocação de recursos, como apresentado no trabalho (FARIAS, 2016), coexistindo com a estratégia de balanceamento de carga BACOS, seria o mais adequada para atender a demanda de recursos com a máxima eficiência em sua utilização.

Assim, a saída dessa etapa consiste em uma base de treino contendo um conjunto de informações sobre os experimentos no formato (N,R,W,RT,I). Perceba que em um experimento, a cada variação de (N,R) existe uma configuração de W associada que produz como saída valores de RT e I. Na próxima etapa, será explorado o custo-benefício dos resultados RT e I de cada

experimento, quando configurados diferentes valores dos pesos das classes de desempenho W e parâmetros da carga de trabalho (N,R).

4.6 Método de balanceamento de carga

Na terceira e última etapa, o BACOS tem por objetivo construir uma matriz de consulta para armazenar os pesos ideais das classes de desempenho W que balanceiam a carga de trabalho e maximizam o desempenho geral do sistema. Essa matriz é construída a partir da base de treino dos experimentos gerados no ambiente simulado de produção. A matriz de consulta rápida contem os pesos das classes de desempenho ótimos que são utilizados no ambiente de produção quando os parâmetros da carga de trabalho (N,R) são observados. Dessa forma, define-se W_{opt} como os pesos das classes de desempenho ótimos que maximizam o desempenho geral do sistema quando o sistema é submetido a uma carga de trabalho com padrão (N,R). Assuma que o conceito de ótimo é visto como ótimo local, ou seja, considerado apenas entre o espaço de informações registrados na base de treino do BACOS. O parâmetro W_{opt} determina a distribuição dos dados e requisições dos dispositivos de armazenamento com o melhor equilíbrio entre o tempo médio de resposta RT e a taxa de insucesso das requisições feitas ao sistema, representado por I. Em outras palavras, W_{opt} passa a intuição de que dispositivos de armazenamento em classes com melhor desempenho recebem mais requisições do que em classes com pior desempenho, de forma que dispositivos mais rápidos consigam suportar a maior parte da demanda de requisições para atender o máximo de requisições com sucesso.

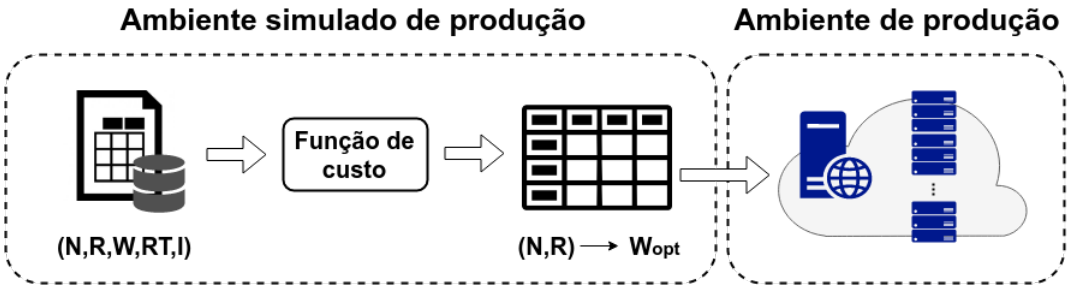


Figura 23 – Fluxo de execução do método de balanceamento.

Esta etapa foi dividida em três seções para facilitar o entendimento do fluxo de execução do BACOS. A Figura 23 mostra o fluxo de execução geral do método de balanceamento de carga. Note que no ambiente simulado de produção, a partir da base de treino do BACOS contendo os parâmetros de configuração e a saída de um experimento (N,R,W,RT,I), uma

função de custo é aplicada para descobrir os pesos ótimos W_{opt} quando configurado uma carga (N,R). O esperado é que os pesos ótimos W_{opt} descobertos no ambiente simulado de produção das classes de desempenho maximizem o desempenho geral do sistema quando aplicados no ambiente de produção. Assim, na seção 4.6.1, será apresentado uma função de custo como principal critério para escolha dos pesos ótimos W_{opt} para cada variação dos parâmetros da carga de trabalho. Na seção 4.6.2, será explicado como a matriz contendo os pesos ótimos é construída e na seção 4.6.3 será discutido como essa matriz é incluída no algoritmo de balanceamento de carga do BACOS no ambiente de produção.

4.6.1 Critério para os pesos ótimos

Esta seção apresenta a função de custo F utilizada como critério para construir a matriz de consulta contendo os pesos ótimos W_{opt}. Recorde que na etapa anterior da estratégia BACOS, descrita na seção 4.5, foi gerado um conjunto de experimentos sobre a variação dos parâmetros (N,R,W), onde (N,R) são os parâmetros de configuração da carga de trabalho e W o parâmetro de configuração do sistema. Em cada experimento, foi coletado o tempo médio de resposta RT e a taxa de insucesso das requisições I resultando em um registro no formato (N,R,W,RT,I).

Para uma determinada variação da carga de trabalho (N,R), existem vários candidatos para os pesos das classes de desempenho W que podem melhorar o desempenho geral do sistema como possíveis soluções. Porém W_{opt} é a melhor solução entre elas. Assim, a partir dos registros dos experimentos, cujo formato do registro de um experimento é (N,R,W,RT,I), define-se L_W como a lista dos conjuntos dos pesos das classes de desempenho relacionada a uma carga de trabalho com parâmetros (N,R). A lista L_W agrupa todos os experimentos configurados com o parâmetro do sistema W relacionados a uma determinada carga (N,R). Desta forma, para cada W contido na lista L_W de uma carga (N,R), existem seus respectivos valores de RT e I associados. Dito isto, a taxa de insucesso médio I_{mean}, o maior tempo de resposta RT_{max} e o menor tempo de resposta RT_{min} dos experimentos configurados com W podem ser calculado com base nos valores RT e I em L_W.

$$F(RT,I) = \frac{RT + (RT_{min} \times P(I))}{RT_{max} + RT_{min}} \tag{4.1}$$

Considerando os valores estatísticos pré-calculados em L_W associados a uma carga

(N,R), a função F é definido na equação 4.1 como uma função de custo para um determinado experimento (N,R,W,RT,I) que tem parâmetros de entrada (N,R,W) e saídas RT e I. A função de custo F tem como entrada o RT e I de um possível candidato W para ser o W_{opt}. A função F gera um valor de saída entre 0 e 1, esse valor reflete o desempenho geral de um experimento em termos do tempo de resposta e a taxa de insucesso das requisições.

$$P(I) = \frac{I}{I + I_{mean}} \tag{4.2}$$

Perceba que na função de custo F, quanto menor o RT gerado por um determinado experimento configurado com uma dada distribuição dos pesos em W, menor é o valor resultante da função. Entretanto, o valor de RT é penalizado com uma fração do RT_{min} definido por P na equação 4.2 para evitar que soluções com baixo tempo de resposta, mas com uma grande quantidade de requisições não atendidas, sejam favorecidas. O valor retornado por P alivia o efeito colateral de considerar somente o tempo de resposta das requisições concluídas com sucesso. No pior caso, quando P for igual a 1, o RT será penalizado com o menor tempo de resposta RT_{min}. O valor de RT é normalizado pelo RT_{max} e RT_{min} para que a imagem da função assuma valores entre 0 e 1. Portanto, quanto mais próximo o valor de F de um experimento se aproxima de 0, melhor é o candidato W para o W_{opt} que otimiza o desempenho geral do sistema de armazenamento.

4.6.2 Construção da matriz de pesos

Nesta seção, será visto a construção de uma matriz de consulta rápida M contendo os pesos ótimos das classes de desempenhos W_{opt} para cada configuração dos parâmetros da carga de trabalho utilizando a função F, apresentada anteriormente. Uma vez encontrados os pesos ótimos e inseridos em uma matriz de consulta rápida, evita-se recalculá-los a todo momento que houver mudança na carga de trabalho percebida por um sistema de armazenamento em produção.

Na segunda etapa da estratégia BACOS, descrita na seção 4.5, é gerado um conjunto de experimentos com as possíveis soluções para os pesos das classes de desempenho W, onde o desempenho geral do sistema resultante (RT e I) foi obtido sobre uma variação específica da carga de trabalho (N,R). Então, para a escolha do conjunto de pesos ótimos das classes de desempenho W_{opt} de uma carga de trabalho (N,R), o desafio é encontrar o conjunto de pesos das classes de desempenho W que maximiza o desempenho geral do sistema, através do

melhor equilíbrio entre tempo de resposta e taxa de insucesso das requisições não atendidas, encapsuladas pela função F.

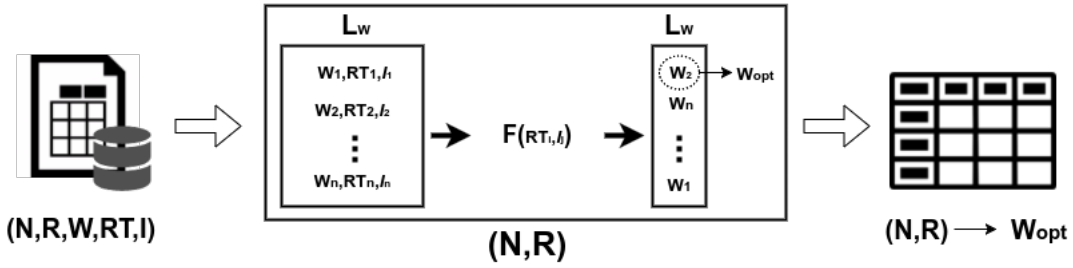


Figura 24 – Construção da matriz M.

A Figura 24 ilustra a criação da matriz de consulta M contendo os W_{opt} para cada carga (N,R). Lembre que o conjunto das possíveis soluções W foram organizadas em um lista L_W para cada configuração de carga de trabalho (N,R) para facilitar a computação do custo da solução, discutida anteriormente na seção 4.6.1 quando apresentado a função de custo F. Com o intuito de encontrar o W_{opt} em cada L_w, a função de custo F é aplicada em cada $W \in L_W$, passando como entrada o RT e I resultante da configuração de um determinado W como mostra a Figura 24. O resultado desse processo é um valor de custo associado a cada W. Em seguida, ordena-se de forma crescente os valores resultantes da função de custo F das possíveis soluções W em L_w. Logo, o W_{opt} de cada padrão de carga (N,R) é o W que está no topo da lista L_w resultante, pois o W_{opt} é o conjunto de pesos das classes de desempenho que minimiza o valor da função F e representa o melhor desempenho que o sistema consegue fornecer com uma determinada carga (N,R). Seguindo o mesmo procedimento para cada variação (N,R), é construído a matriz M que mapeia uma tupla (N,R) para seu peso ótimo W_{opt} escolhidos a partir da função de custo F.

Após a construção da matriz M, os pesos ótimos W_{opt} serão configurados pelo sistema em produção quando houver mudança na carga de trabalho. Ou seja, quando houver um mudança nos parâmetros da carga (N,R) haverá uma mudança do conjunto de pesos atuais em W para os pesos em W_{opt} com o intuito de regular o pesos ótimos aos novos parâmetros da carga. Entretanto, ao mudar de configuração do peso atual para o desejado, um montante de objetos será migrado pela rede para ajustar a proporção de dados dos dispositivo de armazenamento especificada pelo novo conjunto dos pesos das classes de desempenho. Se o volume de dados transferido pela rede for grande, ele irá sobrecarregar a banda de rede compartilhada do sistema que deveria ser aproveitado para atender às requisições feitas por clientes. Essa sobrecarga

durante a migração de dados pode causar um colapso no sistema de armazenamento, e por consequência gerar indisponibilidade do serviço de armazenamento. Por esse motivo, é proposto um parâmetro de configuração para a estratégia BACOS chamado de fator de migração α. O parâmetro α mitiga o impacto de migração de dados causados por uma nova configuração dos pesos das classe de desempenho. Dessa forma, esse parâmetro possibilita que o impacto para satisfazer a nova configuração seja dividido em iterações. Cada iteração irá movimentar uma quantidade de dados proporcional ao total necessário para atender a nova configuração de pesos de acordo com α assumindo valores entre 0 e 1.

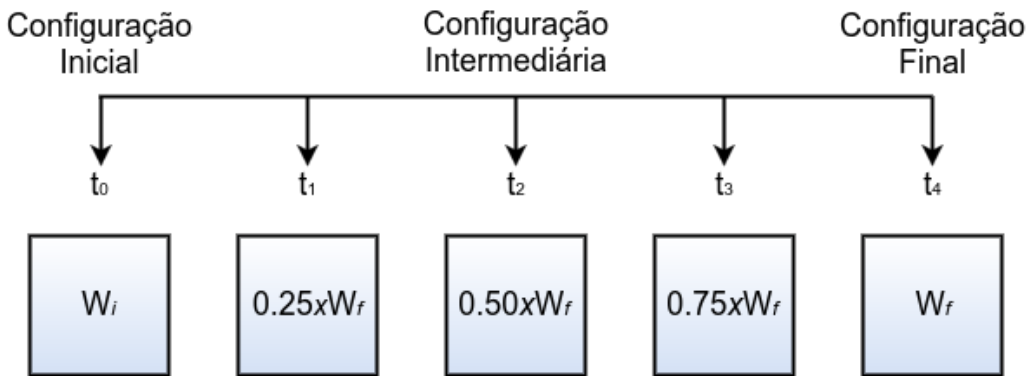


Figura 25 – Exemplo de configuração do parâmetro α configurado com valor igual a 0.25.

A escolha do parâmetro α estabelece um conflito entre o custo de migração e o tempo de convergência ao mudar a distribuição dos dados entre os dispositivos de armazenamento. Por exemplo, considere um cenário em que a atual distribuição dos dados de um sistema de armazenamento, definido pelo parâmetro dos pesos das classes de desempenho, esteja inicialmente configurado com W_i. Em seguida, BACOS determina uma a configuração alvo com estado final W_f. A Figura 25 ilustra como BACOS implementa a mudança de configuração dos pesos W_i para o W_f a cada instante de tempo t_i utilizando o parâmetro α com o valor igual a 0.25. Em uma iteração, será migrado apenas 25% do total dos dados destinados para regularizar a nova distribuição dos dados previamente armazenados no sistema. A cada iteração no instante de tempo t_i, W_i passa por valores intermediários até atingir a nova configuração de pesos definida em W_f. Ou seja, a distribuição dos dados entre os dispositivos de armazenamento atual passa por estados intermediários até convergir para a distribuição de dados desejada no instante de tempo t_4. Nesse exemplo, assume-se que o valor de α por iteração é fixo.

Note que a estratégia BACOS define a distribuição dos dados e requisições para cada dispositivo de armazenamento por meio dos pesos em W do DHT consistente com suporte a replicação. Contudo, a escolha dos objetos a serem migrados e os possíveis dispositivos

de armazenamento destino não fazem parte direta da estratégia BACOS. Essas tarefas são delegadas ao DHT. O remapeamento do espaço de chaves do DHT determina a localização e quais objetos precisam ser migrados para refletir a configuração dos pesos em W. Ainda, o sistema de armazenamento é responsável por manter a consistência das réplicas dos objetos durante e após o processo de migração de dados.

4.6.3 Pseudocódigo

Esta seção apresenta um pseudocódigo do algoritmo de balanceamento BACOS proposto neste trabalho, aplicado em um sistema de armazenamento de objetos em produção. O Algoritmo 1 proposto, assume que a solução ótima W_{opt} escolhida para cada variação de (N,R) já foi consolidada na matriz M durante a sua construção no ambiente simulado de produção, descrito na seção anterior (seção 4.6.2). Assim, o algoritmo recebe como entrada a matriz de consulta M contendo os pesos ótimos W_{opt}, uma estrutura de dados Dev representando os dispositivos agrupados pelas classes de desempenhos C_i e o fator de migração α para evitar sobrecarregar o sistema durante a fase de migração de dados, também discutido na seção anterior. Considerando ainda que o algoritmo é executado a cada instante t, três estados possíveis são apresentados como saída do algoritmo após um ciclo de execução: 0) não há necessidade de fazer o balanceamento de carga, 1) balanceamento de carga executado com sucesso e -1) sem recurso necessário para realizar o balanceamento de carga. Mais adiante, serão discutidas as principais situações que geram as saídas do algoritmo.

O algoritmo BACOS realiza o monitoramento da capacidade e IOPS utilizado dos dispositivos de armazenamento e coleta métricas sobre a carga de trabalho para tomada de decisão, como pode ser visto nas linhas 1 a 5, onde N^t é o número total de clientes e R^t a taxa de leitura e escrita no instante de tempo t. Essas métricas coletadas correspondem aos parâmetros da carga de trabalho corrente e coletadas da perspectiva do sistema de armazenamento. Além disso, são coletadas métricas sobre os parâmetros do sistema como os pesos das classes de desempenho W, o IOPS e a capacidade de armazenamento corrente dos dispositivos de armazenamento. A distribuição dos pesos das classes de desempenho é representada por W^t. Os parâmetros $d_{1...Z}.IOPS^t$ e $d_{1...Z}.Cap^t$ representam, respectivamente, o IOPS e a capacidade de armazenamento corrente coletados de cada dispositivo de armazenamento.

Na linha 6 do algoritmo, a estratégia BACOS consulta a matriz M para obter W_{opt} para classes de desempenho a partir do número de clientes N^t e taxa de leitura/escrita R^t no

momento t atual. Na linha 7 é feita uma estimativa de configuração dos pesos da classe de desempenho que deve ser aplicada para atingir a configuração esperada W_{opt}. É possível que esse processo leve algumas iterações para convergir devido a configuração do parâmetro α.

Algoritmo 1: BACOS load Balance Algorithm

input : $T \leftarrow \{(N_1,R_1) : W_{opt_1}, ..., (N_i,R_j) : W_{opt_k}\}$
$Dev \leftarrow \{C_1 : [d_1, ..., d_Z], ..., C_L : [d_1, ..., d_Z]\}$
α

output : 0: no need for load balancing
1: load balancing success
-1: no more resource to load rebalance

```
1  N^t ← collectCurrentNumberOfClients();
2  R^t ← collectCurrentReadRatio();
3  W^t ← collectCurrentClassWeight();
4  d_1...Z.IOPS^t ← collectCurrentIOPS();
5  d_1...Z.Cap^t ← collectCurrentCapacity();
6  W_opt ← T[(N^t, R^t)];
7  W^t ← W^t + α × (W^t − W_opt);
8  if d_1...Z.IOPS^t / ∑d_1...Z.IOPS_total ≥ 1 and d_1...z.Cap^t / ∑d_1...z.Cap_total ≥ 1 then
9      return −1;
10 else if dataMigration() or W_t ≈ W_opt then
11     return 0;
12 else
13     while W^t ≠ W_opt do
14         for l ← 1 to L do
15             deviceList ← Dev[C_l];
16             foreach dev in deviceList do
17                 w_dev ← getDeviceWeight(W^t, len(deviceList));
18                 apply w_dev to dev;
19             end
20         end
21         W^t ← W^t + α × (W^t − W_opt);
22     end
23     return 1
24 end
```

É importante destacar que os valores de N e R armazenados em M são considerados valores discretos em seus respectivos domínios. Consequentemente, é um desafio encontrar valores de W_{opt} em produção para todas as combinações possíveis de (N,R). É computacionalmente custoso armazenar todos os valores de W_{opt} na matriz M, pois existem infinitas combinações de experimentos (N,R,W) e os computadores atuais possuem limitações de espaço na memória principal. Além disso, o tempo para gerar um conjunto de experimentos pode ser inviável. Perceba que há um *tradeoff* entre a descoberta precisa dos valores de W_{opt} e o esforço computacional para gerar e armazenar a matriz M em memória. Quanto mais informação sobre

a variação de (W,N,R) executadas no ambiente simulado de produção, melhor é a precisão da escolha do valor de W_{opt} para a construção da matriz M, discutido na seção 4.6.2. Em contrapartida, o tempo e outros custos diretamente envolvidos do sistema de armazenamento para gerar os experimentos, podem ser elevados. Além disso, o espaço em memória para a matriz M é limitado. Dito isto, quando BACOS vai consultar a matriz M através dos parâmetros de carga (N,R) desconhecidos, existe a possibilidade da matriz não conter o W_{opt} desejado visto que essa configuração específica não foi explorada no ambiente simulado de produção. Por esses motivos, o método de interpolação baseado no *nearest neighbor* (LEHMANN *et al.*, 1999) foi utilizado para encontrar o valor W_{opt} mais próximo encontrado na matriz M ao interpolar os parâmetros da carga de trabalho (N,R). Por exemplo, considere a matriz de pesos da Tabela 4 construído na segunda etapa da estratégia BACOS, caso os valores dos parâmetros da carga (N,R) forem iguais a $(70,140)$, eles serão interpolado para os valores vizinhos mais próximos $(80,150)$. Portanto, o W_{opt} selecionado será o W_{opt_5}.

Tabela 4 – Interpolação *nearest neighbor* utilizando a matriz de pesos.

		N					
		10	**50**	**100**	**150**	**200**	**250**
R	**50**	W_{opt_2}	W_{opt_3}	W_{opt_4}	W_{opt_4}	W_{opt_6}	W_{opt_8}
	80	W_{opt_1}	W_{opt_2}	W_{opt_3}	W_{opt_5}	W_{opt_7}	W_{opt_9}

Na linha 8, é verificado se existem recursos suficientes nos dispositivos de armazenamento para aplicar o valor de W_{opt} desejado, fornecido pela matriz M. Caso não exista capacidade de armazenamento ou IOPS disponível nos dispositivos de armazenamento, a estratégia deixa de sugerir novas configurações. Essa decisão foi adotada para evitar mais sobrecarga para o sistema, pois qualquer nova mudança não vai surtir efeito eficiente para melhoria do desempenho geral do sistema, uma vez que os componentes do sistema já se encontram sobrecarregados. Apenas a inserção de novos recursos mitigaria esse problema. BACOS não explora o gerenciamento da falta de recursos dos dispositivos de armazenamento. Porém, como mostrado na linha 9, o algoritmo sinaliza para o administrador do sistema ou uma estratégia de terceiros para provisionar novos recursos.

Nas linhas 10 e 11 são verificadas situações onde não é necessário alterar a configuração dos pesos das classes de desempenho. Duas situações são possíveis. A primeira situação acontece quando já existe um processo de migração em execução, derivado de uma

decisão do BACOS para mudar os pesos das classes de desempenho corrente W^t para o peso alvo W_{opt}. Essa medida evita instabilidades e sobrecarga do sistema, pois mudanças frequentes nos pesos compromete a convergência do algoritmo, causam constante migração de dados entres os dispositivos de armazenamento e sobrecarregam os recursos do sistema. A segunda situação é quando o peso W^t atual é muito próximo do ótimo W_{opt}, consultado na matriz M, para os novos valores dos parâmetros da carga de trabalho (N,R). Em outras palavras, o padrão de carga de trabalho no instante t não mudou o suficiente em relação ao instante $t-1$. Além disso, um mesmo W_{opt} pode ser similar a diferentes tipos de padrões de carga (N,R).

A partir da linha 13 até a linha 23, são aplicados os pesos das classes de desempenho que engatilham a migração de dados entre os dispositivos de armazenamento para ajustar a proporção de dados e requisições que cada um deve manipular. Em outras palavras, alguns objetos, definidos pelo novo mapeamento de chaves do DHT, são migrados entre os dispositivos de armazenamento a cada iteração, até que a distribuição dos dados representados pela configuração W^t se torne a distribuição esperada W_{opt}. Considerando que o controle da proporção de dados é feito por dispositivo de armazenamento d_l, o peso da classe de desempenho C_l também define o peso dos dispositivos pertencente a essa classe como mostrado nas linhas 17 e 18. Na linha 21, a nova configuração de peso W^t mantém a distribuição da proporção dos dados que serão migrados por iteração, W^t é recalculado em função do parâmetro α e os pesos ótimos para classes de desempenho W_{opt}.

4.7 Conclusão

Nesse capítulo, foi apresentada uma estratégia dinâmica para balanceamento de carga para sistemas de armazenamento baseado em objetos ou *object storages* chamada BACOS. Inicialmente a arquitetura e a função dos componentes do sistema de armazenamento ideal para o bom funcionamento do BACOS foram detalhadas. Em seguida, foi discutido como BACOS tira vantagem da heterogeneidade do desempenho dos dispositivos de armazenamento. Foi visto como o BACOS manipula dinamicamente a distribuição dos dados e requisições dos dispositivos de armazenamento dada uma mudança na carga de trabalho observada. Também foi proposta uma solução para mitigar o problema de migração de grandes volumes de dados entre dispositivos de armazenamento para ajustar a distribuição do dados já alocados no sistema enquanto o sistema lida com novas requisições.

5 AVALIAÇÃO EXPERIMENTAL

Este capítulo apresenta a avaliação experimental visando mostrar a efetividade da estratégia BACOS ao melhorar o desempenho geral de um sistema de armazenamento de objetos (*object storage*). Os resultados experimentais da estratégia BACOS foram gerados utilizando um *object storage* comercialmente popular e comparados com a estratégia padrão do *object storage* em que BACOS foi implantado. Também são apresentados detalhes da implementação e analise do comportamento da estratégia BACOS ao lidar com diferentes tipos de cargas de trabalho.

5.1 Introdução

A análise experimental proposta nesse trabalho tem como principal objetivo mostrar que a estratégia BACOS consegue melhorar o desempenho geral do sistema de armazenamento em nuvem, onde a estratégia foi implantada, sujeito a cargas de trabalho dinâmicas. A análise experimental ainda pretende mostrar que a estratégia BACOS consegue melhorar o desempenho geral do sistema mesmo sobre padrões de cargas de trabalho não registrados na base de treino da estratégia BACOS. Isso é explicado pelo fato de que a estratégia BACOS consegue se adaptar bem aos padrões de carga previamente observados dentro de um intervalo de valores para cada componente da carga de trabalho devido ao método de interpolação baseado no *nearest neighbor* como explicado no capítulo 4. Além do mais, foi discutido como BACOS gera uma base de treino a partir de vários experimentos variando parâmetros da carga de trabalho e do sistema para construção da matriz M. Devido as infinitas combinações dos parâmetros, também foi utilizado o método de interpolação baseado no *nearest neighbor* para simplificar a escolha dos pesos das classes de desempenho e economizar espaço na matriz M armazenando apenas os pesos das classes de desempenho com as cargas de trabalho observadas.

Em vista disso, foram conduzidos alguns experimentos utilizando o sistema de armazenamento de objetos *Openstack Swift* em um ambiente de nuvem privada composta por dispositivos de armazenamento heterogêneos quanto ao desempenho e a capacidade de armazenamento. A análise experimental foi feita sobre a perspectiva do cliente, ou seja, o sistema de armazenamento foi tratado como caixa preta e as aplicações não foram cientes do gerenciamento dos recursos do sistema. Dessa forma a estratégia BACOS implantada no *OpenStack Swift* foi comparada com a estratégia de balanceamento de carga padrão sugerida pela documentação do *Openstack Swift* discutida na seção 3.4.2. A implantação da estratégia BACOS

não necessita alterar o código fonte do *OpenStack Swift*, uma vez que é adicionado apenas uma aplicação em um dos servidores distribuídos que consome as interfaces de configuração dos parâmetros do sistema de armazenamento. Portanto, por meio das interfaces expostas pelo sistema de armazenamento, a aplicação da estratégia BACOS fica responsável pela decisão do balanceamento de carga ao verificar a carga de trabalho atual e configurar os pesos das classes de desempenho no *object storage* para melhorar o desempenho geral do sistema de acordo com a carga. Ao longo deste capítulo, a estratégia padrão do *Openstack Swift* é referenciada como *baseline*.

Nos experimentos realizados, o desempenho do sistema de armazenamento foi observado pelos usuários ao submeter um determinado padrão de carga de trabalho, composto pelo número de clientes e taxas de leitura e escrita das requisições. Como uma forma de medir o desempenho por meio dos resultados observados, foram consideradas as seguintes métricas:

- O tempo médio de resposta, medido em milissegundos.
- Número de requisições feitas por segundo (*troughput*).
- Taxa de sucesso das requisições feitas ao sistema de armazenamento.

Vale destacar que as requisições de leitura e escrita no *OpenStack Swift* são concluídas com sucesso somente se o sistema de armazenamento conseguir obter o objeto solicitado e confirmar a escrita das réplicas de um objeto de acordo com a estratégia de seleção de réplicas e a estratégia de quórum, respectivamente. Para mais detalhes sobre as operações de leitura e escrita do *OpenStack Swift*, consulte a seção 3.4.2.

5.2 Infraestrutura

Foi implementado um protótipo da estratégia BACOS para análise de desempenho usando a linguagem de programação Python versão 2.7. Para o método de interpolação multivariável *nearest neighbor*, foi utilizada a implementação disponível na biblioteca *SciPy* versão 0.18.1 (SCIPY, 2017). Como ferramenta de *benchmark*, foi utilizado o *Cloud Object Storage Benchmark* (COSBench) (ZHENG *et al.*, 2013), versão 0.4.2*rc*2. O *COSBench* é uma ferramenta de *benchmark* com arquitetura escalável capaz de simular diferentes tipos de aplicações mediante da configuração customizada da carga de trabalho. No ambiente de experimentação, o *COSBench* foi empregado para submeter a carga de trabalho para o sistema de armazenamento e coletar métricas de desempenho a partir da perspectiva das aplicações.

O protótipo da estratégia BACOS foi testado no ambiente de virtualização *OpenStack*

na nuvem privada do Laboratório de Sistemas e Banco de Dados (LSBD) [1]. Como caso de teste para sistema de armazenamento de objetos em nuvem, foi utilizado o *OpenStack Swift* configurado com a política de replicação parcial de objetos. O fator de replicação na construção do DHT foi configurado com valor igual 2, indicando que foram armazenados 2 réplicas por objeto dispersos entre os dispositivos de armazenamento do *OpenStack Swift*. O fator de replicação foi configurado apenas com 2 réplicas em razão da limitação dos recursos no ambiente de experimentação. Um fator de replicação maior implicaria em mais troca de mensagens pela rede para manter a consistência por meio da sincronização entre servidores de armazenamento. Para compor a arquitetura do *OpenStack Swift*, foi utilizado um conjunto de máquinas virtuais provindas pelo serviço de provisionamento de recursos computacionais *Nova*. Já os dispositivos de armazenamento foram criados como volumes, fornecidos pelo serviço de provisionamento de armazenamento em blocos *Cinder*. Tanto o *Nova* quanto o *Cinder* fazem parte da arquitetura de serviços do *OpenStack* (ROSADO; BERNARDINO, 2014).

Tabela 5 – Configuração dos nós computacionais no ambiente de experimentação.

Nome da instância (VM)	**Processamento (vCPUs)**	**Memória principal (GB)**	**Memória secundária (GB)**
COSBench	4	8	80
Swift proxy	16	16	160
SN_1	4	8	80
SN_2	4	8	80

A Tabela 5 resume os parâmetros dos recursos utilizados para os nós computacionais no ambiente de experimentação. Particularmente, foi necessário configurar apenas uma máquina virtual destinada ao *COSBench* para sobrecarregar o sistema de armazenamento. Para a arquitetura do *Openstack Swift*, também foi configurado apenas um *swift proxy node*, porém com maior poder de processamento para suportar a demanda das requisições. O *proxy node* também foi utilizado para tomar a decisão de balanceamento por meio da coleta do número de clientes, taxa de leitura/escrita, IOPS e espaço de armazenamento corrente. Foram criadas 2 máquinas virtuais para comporem os servidores de armazenamento (*storage nodes*) do sistema *OpenStack Swift*, chamados de SN_1 e SN_2. Cada nó computacional no sistema possui um espaço de armazenamento secundário acoplado para armazenar o sistema operacional e programas internos. Porém, os *storage nodes* foram criados para cada um gerenciar 3 dispositivos de

[1] Laboratório de Sistemas e Banco de Dados (LSBD) - <http://www.lsbd.ufc.br/>

armazenamento, reservados para armazenar dados dos clientes.

Tabela 6 – Configuração dos dispositivos de armazenamento no ambiente de experimentação.

storage node	**Dispositivo de armazenamento**	**Capacidade de armazenamento em Gigabyte (GB)**	**Número de operações de entrada e saída por segundo (IOPS)**	**Classe do dispositivo de armazenamento**
SN_1	d_1	10	150	C_3
	d_2	20	500	C_2
	d_3	40	1000	C_1
SN_2	d_1	40	150	C_3
	d_2	40	150	C_3
	d_3	40	150	C_3

A Tabela 6 resume a configuração dos recursos dos dispositivos de armazenamento assim como o resultado da classificação dos dispositivos em classes de desempenho, discutido na seção 4.4. Foram definidas 3 classes de desempenho com o objetivo de agrupar dispositivos de armazenamento com desempenho alto (C_1), médio (C_2) e baixo (C_3). O servidor de armazenamento SN_1 foi configurado com dispositivos heterogêneos em relação a capacidade e IOPS suportados. Geralmente, dispositivos de armazenamento de alto desempenho atingem elevadas taxas de IOPS, mas são mais caros e possuem baixa capacidade de armazenamento (ZHOU *et al.*, 2016). O servidor de armazenamento SN_2 foi configurado apenas com dispositivos de armazenamento homogêneos em relação ao desempenho e a capacidade de armazenamento. Dispositivos de armazenamento de baixo custo podem apresentar alta capacidade de armazenamento, mas oferecem baixas taxas de IOPS (ZHOU *et al.*, 2016). Esse arranjo experimental foi criado para compor um sistema de armazenamento heterogêneo contendo uma proporção maior de dispositivos de baixo custo em relação à quantidade de dispositivos de armazenamento de alto desempenho. Os sistemas de armazenamento heterogêneos conseguem prover baixo tempo de resposta para as aplicações, equiparáveis aos sistemas de armazenamento compostos apenas por dispositivos de alto desempenho (XU *et al.*, 2015). Dessa forma, os sistemas de armazenamento heterogêneos podem baratear os custos de aquisição e manutenção de grandes quantidades de dispositivos de armazenamento.

5.3 Cenários de avaliação

Sistemas de armazenamento em nuvem buscam manter a qualidade do serviço das aplicações mesmo sobre crescente demanda do volume de requisições. O paralelismo das

aplicações pode gerar grandes volumes de requisições para um sistema de armazenamento, de modo que o acesso concorrente aos dispositivos de armazenamento possa degradar a vazão e a latência do sistema de armazenamento (HOU *et al.*, 2016). Esses fatores são refletidos nos experimentos como o número de clientes e o padrão de acesso das requisições de leitura/escrita.

Isto posto, foram propostos dois cenários com o intuito de explorar como a estratégia BACOS, implementada no sistema de armazenamento *Openstack Swift* com dispositivos de armazenamento heterogêneos, se adapta à natureza do padrão de acesso das aplicações. Isto é feito considerando o aumento da carga de trabalho em função da variação do número de clientes e fixação de uma determinada taxa de leitura e escrita, dado que este padrão é conhecido ou não pela de balanceamento BACOS. A Tabela 7 resume os cenários propostos e os detalhes de cada cenário são discutidos abaixo:

1. O padrão de acesso das requisições de leitura e escrita feitas ao sistema de armazenamento e previamente conhecido pela estratégia BACOS, ou seja, BACOS possui o registro de parte dos parâmetros da carga de trabalho em sua base de treino. Assim, foram realizados dois experimentos com padrões de leitura/escrita diferentes. No primeiro experimento, foi configurado o padrão de acesso das requisições de leitura/escrita com taxa igual a 80/20 para representar aplicações com leitura intensa. Enquanto no segundo experimento, foi configurado o padrão de acesso das requisições de leitura/escrita com taxa igual a 50/50 para representar aplicações com acesso de leitura similar ao de escrita.
2. O padrão de acesso das requisições de leitura e escrita feitos ao sistema de armazenamento é desconhecido pelo BACOS. Dessa forma, foi realizado apenas um experimento configurando uma carga de trabalho com taxa de leitura/escrita igual a 60/40, simulando um padrão de carga com leitura moderada. Esse experimento visa explorar o comportamento da estratégia BACOS sem informações na base de treino de como a carga e a distribuição de dados influenciam no desempenho do sistema de armazenamento.

Tabela 7 – Resumo dos cenários propostos para a avaliação experimental da estratégia BACOS.

	Cenário 1	**Cenário 2**
Taxa de leitura/escrita	80/20	60/40
	50/50	

O método experimental a seguir foi aplicado de forma similar para ambos os cenários. Foi gerada uma carga de trabalho com tamanho fixo de 4KB e padrão de requisições de leitura e

escrita específica, respectivamente igual a R e $100 - R$ para cada cenário descrito anteriormente utilizando a ferramenta de *benchmark* COSBench. Em seguida, a carga de trabalho foi disparada pela rede em direção ao sistema de armazenamento, aumentando suavemente o número de clientes N conforme os valores descritos na lista $[1, 10, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250]$. O resultado das métricas de desempenho observadas foram coletadas do lado cliente pela ferramenta COSBench. As métricas foram coletadas a cada segundo durante um tempo total de 5 minutos para uma determinada variação de N descrito anteriormente, descartando apenas os valores das métricas coletadas dos 30 segundos iniciais e finais para evitar ruídos durante a coleta. Os valores de uma determinada métrica de desempenho coletados para um dado N foram agregados em uma média e o conjunto dessas médias ao longo da variação de N compõem uma distribuição que foi utilizada para a análise de desempenho. Foram criados 32 contêineres com 500 objetos em cada um, resultando em um total de 160000 objetos em um experimento. Os objetos utilizados entre requisições de leitura foram diferentes dos objetos utilizados para requisições de escrita para isolar o desempenho final de cada tipo de requisição. Onde, 250 objetos por contêiner foram utilizados para leitura e outros 250 objetos por contêiner foram utilizados para requisições de escrita. Como as requisições de leitura e escrita são realizadas de forma concorrentes, metade do total dos objetos (80000 objetos) utilizados em um experimento foram previamente inseridos no sistema como pré-requisito para iniciar o experimento com requisições de leitura. Um objeto escolhido em uma determinada requisição entre os objetos reservados para aquele tipo de requisição segue uma distribuição uniforme.

Os valores dos parâmetros da carga de trabalho utilizados durante a geração da base de treino da estratégia BACOS (consultar seção 4.5) e os valores utilizados nos experimentos discutidas anteriormente foram similares. Entretanto, BACOS utiliza uma base de treino parcialmente ciente da carga de trabalho, pois os experimentos feitos com o número de clientes mudaram de maneira suave, mas com grânulo menor comparado aos descritos anteriormente. O número de clientes foi aumentado linearmente entre os valores descritos na lista $[10, 50, 100, 150, 200, 250]$. Ainda, o padrão de leitura/escrita foi conhecido apenas para o cenário 1, ou seja, igual a $80/20$ e $50/50$. Adicionalmente, os experimentos da base de treino do BACOS foram feitos com o valor do parâmetro *alpha* (seção 4.6.2) definido empiricamente igual a 1. Esse valor indica que a migração dos objetos entre dispositivos de armazenamento foi realizada em apenas uma iteração, suficiente para não sobrecarregar totalmente os componentes do sistema de armazenamento com um custo de migração e convergência razoável no ambiente de experimentação. Como

existem 3 classes de desempenho derivadas da configuração dos dispositivos de armazenamento no *OpenStack Swift*, é assumido que os valores dos pesos das classes de desempenho como $W = (w_1, w_2, w_3)$ (seção 4.5). Assim, $w_i \in W$ foi variado entre valores de 0 a 1 com o passo de 0.05, onde $w_1 + w_2 + w_3 = 1$ e $w_1 \geq w_2 \geq w_3$. Foi excluído o subconjunto de experimentos com a variação de pesos $w_1 < w_2 < w_3$, pois os pesos maiores foram atribuídos às classes de desempenho com dispositivos com baixa vazão, por consequência, gerou-se os piores resultados em relação ao desempenho geral do sistema. Variando os parâmetros (N, R, W) como descrito anteriormente, foram gerados um total de 396 experimentos para formar a base de treino da estratégia BACOS.

Não foram apresentados outros tipos de cenários explorando o tamanho dos objetos ou outros padrões de carga com escrita intensiva, devido a limitação dos recursos dos dispositivos de armazenamento atuais dentro da infraestrutura de experimentação. Dessa forma, não foi possível extrapolar os parâmetros ou número de componentes da arquitetura sem que os recursos do sistema sobrecarregassem rapidamente a ponto deles se tornarem um gargalo. Certamente, esse problema comprometeria os resultados finais dos experimentos.

5.4 Método de avaliação

O método de avaliação escolhido para análise de desempenho das estratégias é um tipo de teste de hipótese baseado no intervalo de confiança para a média da diferença entre duas distribuições (JAIN, 1991). Considere duas distribuições representadas por D_1 e D_2. Para construir o intervalo de confiança para a média da diferença entre D_1 e D_2, as amostras das duas distribuições podem ser pareadas como uma só amostra, caso sejam geradas sobre mesmas condições. Dados os pares de amostras, a diferença entre elas é calculada para gerar uma distribuição resultante D_{dif}. Logo, calcula-se o intervalo de confiança para a média $\bar{x}$ da distribuição resultante D_{dif} assumindo-se uma certa distribuição de probabilidade. Dessa forma, considere, respectivamente, a seguinte hipótese nula e alternativa:

- H_0: $\bar{x} \neq 0$, caso o intervalo de confiança não incluir o valor zero, conclui-se que ambas distribuições D_1 e D_2 são significantemente diferentes.
- H_1: $\bar{x} = 0$, caso o intervalo de confiança incluir o valor zero, conclui-se que ambas distribuições D_1 e D_2 não são significantemente diferentes.

Além disso, analisando o sinal e o valor da média de D_{dif} e a largura do intervalo de confiança é possível definir se uma distribuição foi melhor que outra com mais precisão.

Por exemplo, se a média de $D_{dif} = D_1 - D_2$ for positiva e o seu intervalo de confiança não incluir o zero, indica que D_1 é em média maior que D_2. Além disso, a margem de erro de estimativa presente no intervalo de confiança também apresenta uma certa precisão do quanto uma distribuição é maior que outra.

O método descrito foi aplicado como o critério de comparação para as distribuições resultantes das métricas de desempenho (tempo de resposta e *throughput*) coletadas a partir dos experimentos propostos envolvendo a estratégia BACOS e a *baseline*. O método foi aplicado para as métricas de tempo médio de resposta e *throughput* distintamente para requisições de leitura e escrita. Por representar um número pequeno de amostras, assumimos que a distribuição resultante para o cálculo do intervalo de confiança segue uma distribuição de probabilidade *t* de *Student* com nível de confiança de 95% e graus de liberdade igual a $n-1$ (WINER *et al.*, 1971), onde n é o número de amostras da distribuição. Também, foi avaliado o quanto a estratégia BACOS conseguiu melhorar as métricas de desempenho em relação ao *baseline*. Como complemento do comportamento das métricas de desempenho, foi verificado o quanto as estratégias conseguiram manter a taxa de sucesso sobre alta demanda de requisições feitas ao sistema de armazenamento.

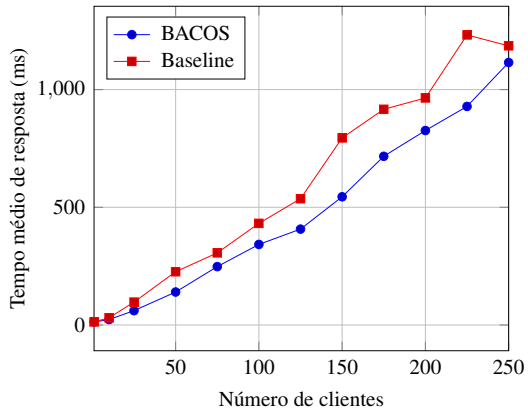
5.5 Resultados experimentais

5.5.1 Cenário 1 - experimento 1: carga de trabalho com taxa de leitura/escrita igual a 80%/20%

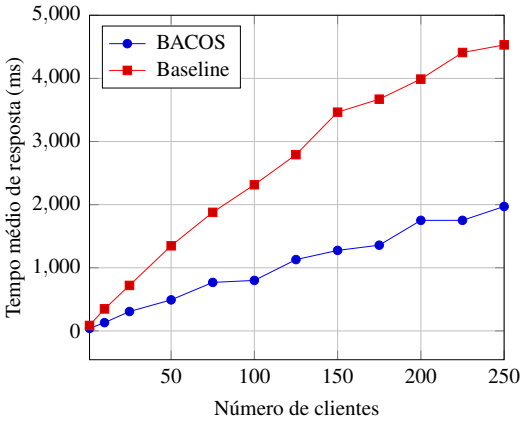
5.5.1.1 Tempo médio de resposta

Os principais resultados obtidos do tempo médio de resposta provindo do primeiro experimento do cenário 1 podem ser visualizados na Figura 26. As Figuras 26a e 26b mostram, respectivamente, as distribuições do tempo médio de resposta para requisições de leitura e escrita das estratégias: *baseline* e BACOS.

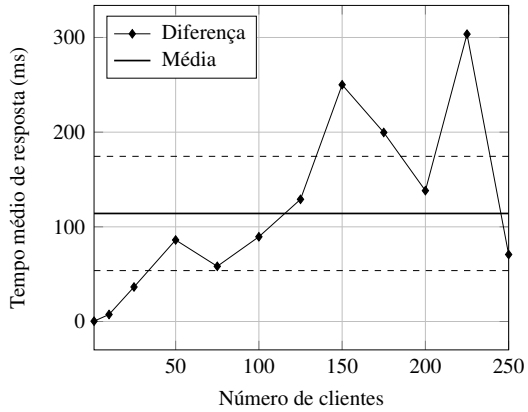
Calculando-se a diferença das distribuições do tempo médio de resposta entre a estratégia *baseline* e BACOS para requisições de leitura e escrita, obtém-se o intervalo de confiança para a média da diferença das distribuições de tempo médio de resposta entre a estratégia *baseline* e BACOS mostrados nas Figuras 26c e 26d. Houve uma diferença média de 114.12 milissegundos e erro máximo de estimativa igual a 60.35 entre a estratégia *baseline*



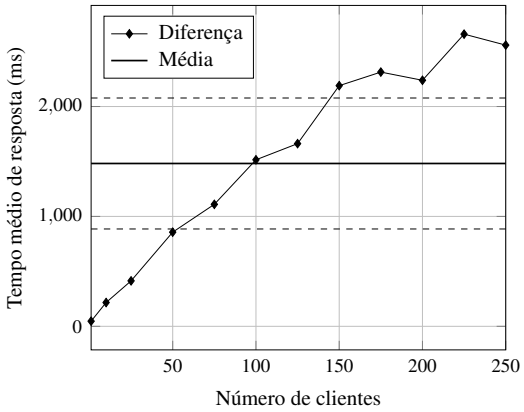
(a) Distribuições do tempo médio de resposta para requisições de leitura.



(b) Distribuições do tempo médio de resposta para requisições de escrita.



(c) Intervalo de confiança para a média da diferença das distribuições do tempo médio de resposta entre o *baseline* e BACOS para requisições de leitura com 95% de confiança.



(d) Intervalo de confiança para a média da diferença das distribuições do tempo médio de resposta entre o *baseline* e BACOS para requisições de escrita com 95% de confiança.

Figura 26 – Comparação do tempo médio de resposta entre o *baseline* e BACOS com padrão de requisições 80% leitura e 20% escrita.

e BACOS para requisições de leitura e uma diferença média de 1481.60 milissegundos e erro máximo de estimativa igual a 595.85 para requisições de escrita. Note que o intervalo de confiança não incluiu o valor zero e a média da diferença entre as distribuições do *baseline* e BACOS foi positiva em ambos tipos de requisições. Consequentemente, aceita-se a hipótese nula H_0 e pode-se afirmar com 95% de confiança que a estratégia BACOS foi melhor que o *baseline* em relação ao tempo médio de resposta para requisições com 80% de leitura e 20% de escrita. BACOS também conseguiu melhorar em média 22.05% o tempo médio de resposta de leitura e em média 60.04% o tempo médio de resposta de escrita em relação ao *baseline*.

Note que a medida que aumenta o número de clientes, o tempo médio de resposta das estratégias também aumenta e a diferença entre elas vai ficando cada vez mais evidente como mostrado nas Figuras 26c e 26d, principalmente para escrita. Perceba que essa diferença é muito

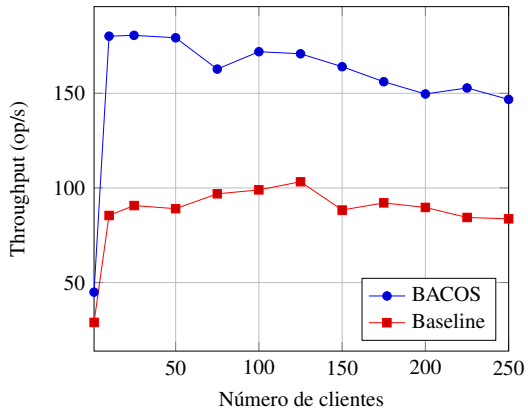
maior para escrita do que para leitura em virtude do quórum implementado pelo *OpenStack Swift*. Como foi configurado 2 réplicas por objeto em todos os experimentos, é necessário apenas 1 dispositivo de armazenamento para confirmar o sucesso da operação de escrita. Assim, utilizando a estratégia BACOS, existe uma grande chance de que pelo menos uma réplica seja atribuída a um dispositivo com alta vazão e retornar com mais rapidez a confirmação de sucesso da operação de escrita por parte dos *storage nodes*.

5.5.1.2 Throughput e taxa de sucesso

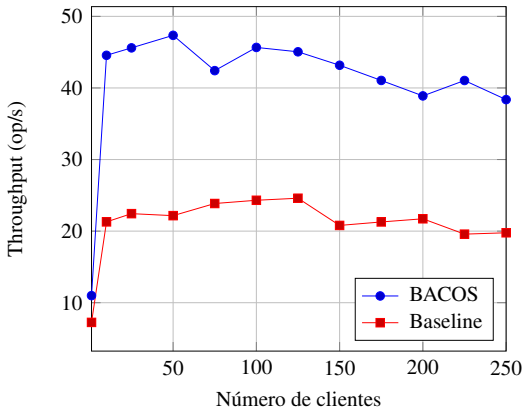
Os principais resultados obtidos do experimento podem ser visualizados na Figura 27. As Figuras 27a e 27b mostram, respectivamente, as distribuições do *throughput* para requisições de leitura e escrita das estratégias: *baseline* e BACOS. Para análise do *throughput*, foi considerado apenas o *throughput* das requisições realizadas com sucesso, computados a partir da taxa de sucesso de leitura e escrita mostradas nas Figuras 27e e 27f.

Calculando-se a diferença do *throughput* entre a estratégia BACOS e o *baseline* para requisições de leitura e escrita, obtém-se o intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* mostrados nas Figuras 27c e 27d. Houve uma diferença média de 69.06 operações por segundo entre o *baseline* e o BACOS e erro máximo de estimativa igual a 12.96 para requisições de leitura. Ainda, houve uma diferença média de 19.59 operações por segundo e erro máximo de estimativa igual a 3.49 para requisições de escrita. O intervalo de confiança não incluiu o valor zero e a média da diferença entre as distribuições do BACOS e o *baseline* foi positiva em ambos tipos de requisições. Dessa forma, aceita-se a hipótese nula H_0 e pode-se concluir com 95% de confiança que a estratégia BACOS foi melhor que o *baseline* em relação ao *throughput* para requisições com 80% de leitura e 20% de escrita. BACOS também melhorou em média 43.82% o *throughput* de leitura e em média 47.26% o *throughput* de escrita em relação a estratégia *baseline*.

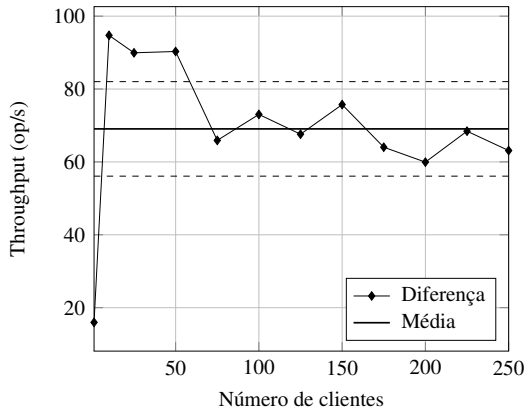
Tanto requisições de leitura quanto de escrita, a baixa e moderada concorrência das requisições derivada do número de clientes entre 1 e 150 maximizam o *throughput* por meio da abordagem de balanceamento de carga das estratégias. Enquanto sobre alta concorrência das requisições, acima de 150 clientes, o *throughput* apresenta uma tendência de queda devido ao grande volume de requisições. Novas requisições são postas em espera e demoram muito mais tempo para serem processadas devido as limitações de recursos dos dispositivos de armazenamento. Como BACOS consegue obter um baixo tempo médio de resposta para requisições



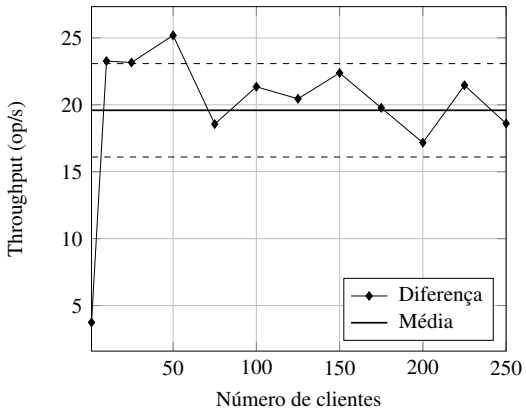
(a) Distribuições do *throughput* médio para operações de leitura.



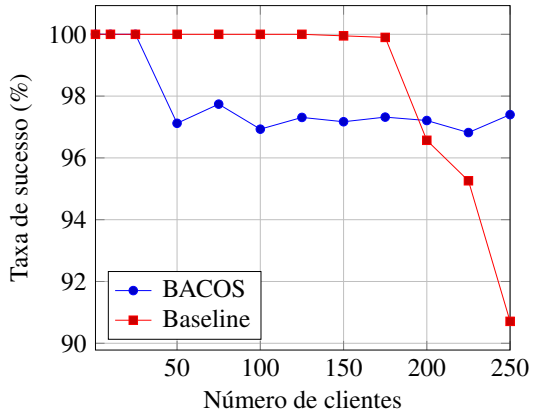
(b) Distribuições do *throughput* médio para operações de escrita.



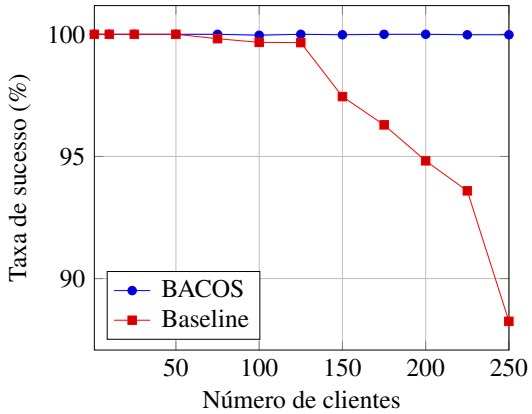
(c) Intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* para requisições de leitura com 95% de confiança.



(d) Intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* para requisições de escrita com 95% de confiança.



(e) Taxa de sucesso para requisições de leitura.



(f) Taxa de sucesso para requisições de escrita.

Figura 27 – Comparação do *throughput* entre BACOS e o *baseline* com padrão de requisições 80% leitura e 20% escrita.

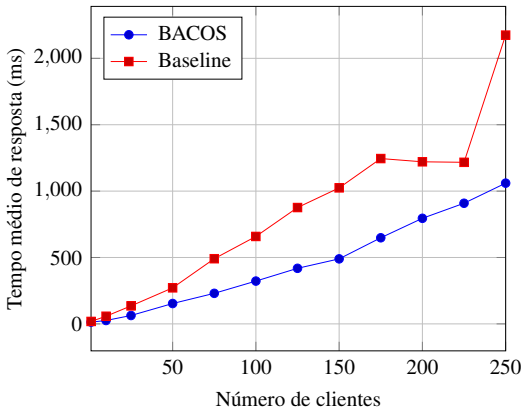
de leitura e escrita, torna-se possível gerar muito mais requisições em menos tempo. BACOS gerou taxas de *throughput* superiores ao do *baseline* mesmo com uma leve degradação da taxa de sucesso em relação as requisições de leitura. Ainda, BACOS conseguiu garantir um taxa de

sucesso igual a 100% para escrita e uma taxa de sucesso razoável para leitura mesmo sobre alta concorrência do volume de requisições.

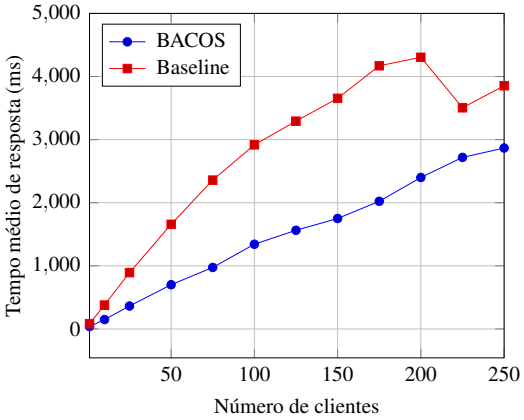
5.5.2 Cenário 1 - experimento 2: carga de trabalho com taxa de leitura/escrita igual a 50%/50%

5.5.2.1 Tempo médio de resposta

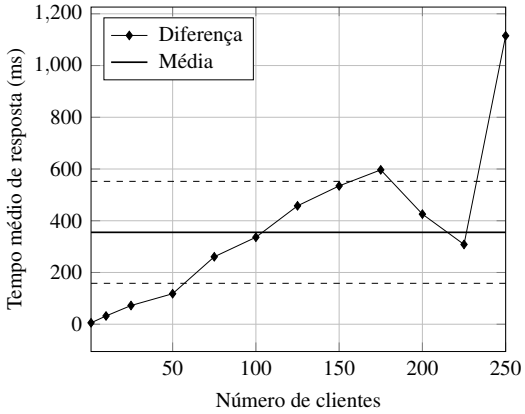
Os principais resultados obtidos do tempo médio de resposta do segundo experimento do cenário 1 são apresentados na Figura 30. As Figuras 28a e 28b mostram, respectivamente, as distribuições do tempo médio de resposta para requisições de leitura e escrita do *baseline* e BACOS.



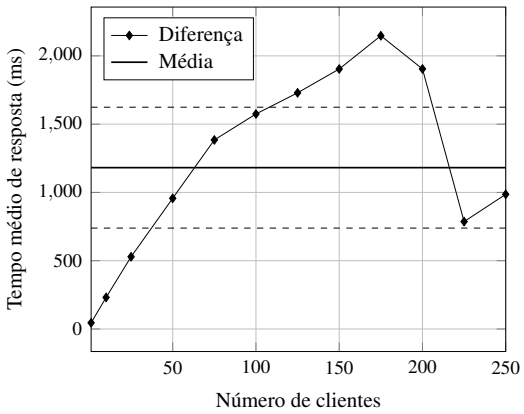
(a) Distribuições do tempo médio de resposta para operações de leitura.



(b) Distribuições do tempo médio de resposta para operações de escrita.



(c) Intervalo de confiança para a média da diferença das distribuições do tempo médio de resposta entre o *baseline* e BACOS para requisições de leitura com 95% de confiança.



(d) Intervalo de confiança para a média da diferença das distribuições do tempo médio de resposta entre o *baseline* e BACOS para requisições de escrita com 95% de confiança.

Figura 28 – Comparação do tempo médio de resposta entre o *baseline* e BACOS com padrão de requisições 50% leitura e 50% escrita.

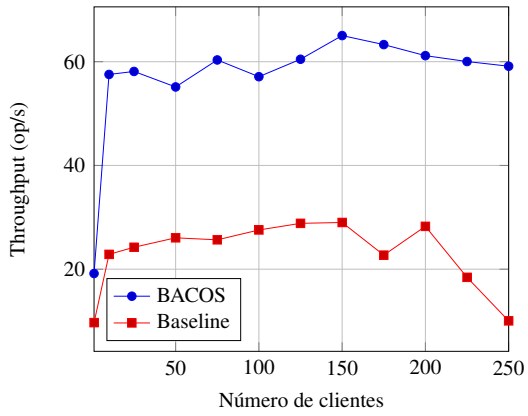
As Figuras 28c e 28d mostram a diferença das distribuições do tempo médio de resposta e o intervalo de confiança para a média da diferença das distribuições de tempo médio de resposta entre o *baseline* e BACOS. Houve uma diferença média de 355.19 milissegundos entre o *baseline* e o BACOS e erro máximo de estimativa igual a 197.05 para requisições de leitura e uma diferença média de 1181.47 milissegundos e erro máximo de estimativa igual a 442.63 para requisições de escrita. Como o intervalo de confiança não incluiu o valor zero e a média da diferença entre as distribuições do *baseline* e BACOS é positiva em ambos tipos de requisições. Portanto, aceita-se a hipótese nula H_0 e pode-se afirmar com 95% de confiança que a estratégia BACOS foi melhor que o *baseline* em relação ao tempo médio de resposta para requisições com 50% de leitura e 50% de escrita. Ainda, BACOS melhorou em média 45.79% o tempo médio de resposta de leitura e melhorou em média 49.41% o tempo médio de resposta de escrita em relação ao *baseline*.

Nesse experimento, BACOS obteve um comportamento similar em relação ao tempo médio de resposta do padrão de carga de leitura/escrita 80%/20%, dado a diferença de escala em milissegundos. Enquanto o *baseline* apresentou mais instabilidades do tempo médio de resposta de leitura e escrita por efeito da alta concorrência provocada pelo número de clientes. Esse fenômeno pode ser observado entre 175 a 250 clientes nas Figuras 28a e 28b e explicado pelo fato do *baseline* não conseguir ajustar o número de requisições com a capacidade de processamento dos dispositivos de armazenamento, além de provocar queda da taxa de sucesso das operações, como será visto na análise do *throughput*.

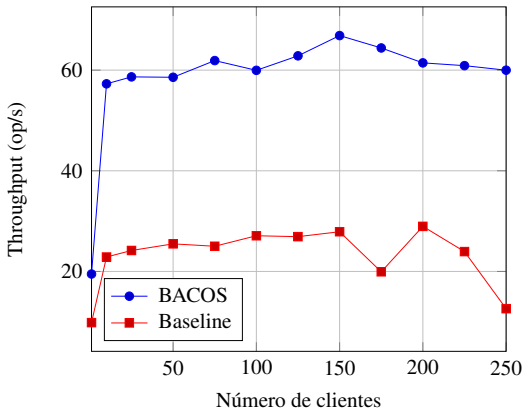
5.5.2.2 Throughput e taxa de sucesso

Os principais resultados obtidos do experimento podem ser visualizados na Figura 29. As Figuras 29a e 29b mostram, respectivamente, as distribuições do *throughput* para requisições de leitura e escrita das estratégias: *baseline* e BACOS. Para análise do *throughput*, foi considerado apenas o *throughput* das requisições realizadas com sucesso, computados a partir da taxa de sucesso de leitura e escrita mostradas nas Figuras 29e e 29f.

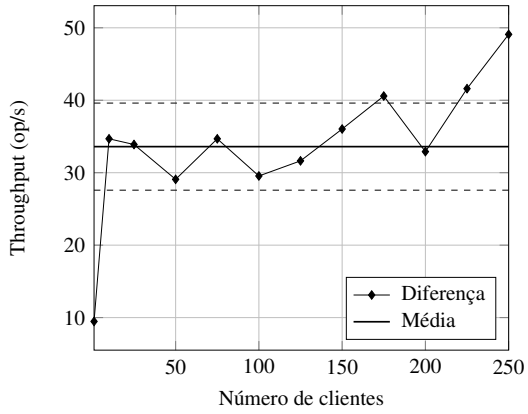
Calculando-se a diferença do *throughput* entre a estratégia BACOS e o *baseline* para requisições de leitura e escrita, obtém-se o intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* como mostrados nas Figuras 29c e 29d. Houve uma diferença média de 33.59 operações por segundo entre o *baseline* e o BACOS e erro máximo de estimativa igual a 6.01 para requisições de leitura. Além disso, houve uma



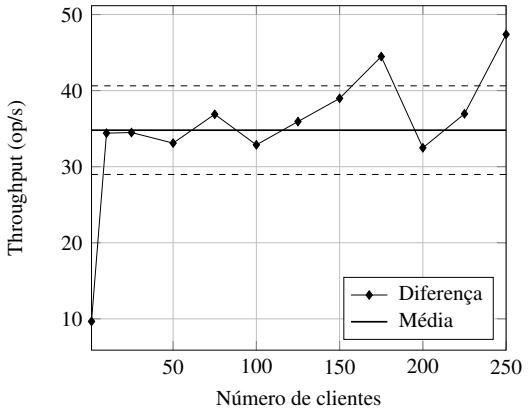
(a) Distribuições do *throughput* médio para operações de leitura.



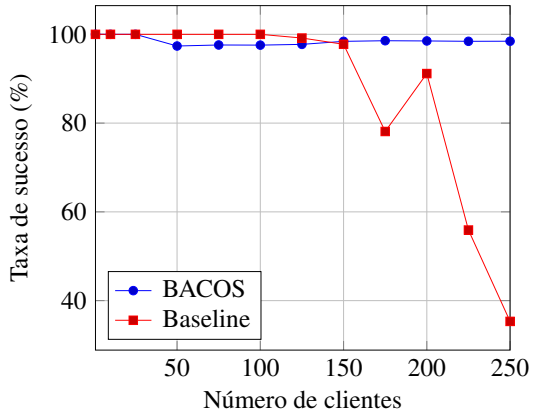
(b) Distribuições do *throughput* médio para operações de escrita.



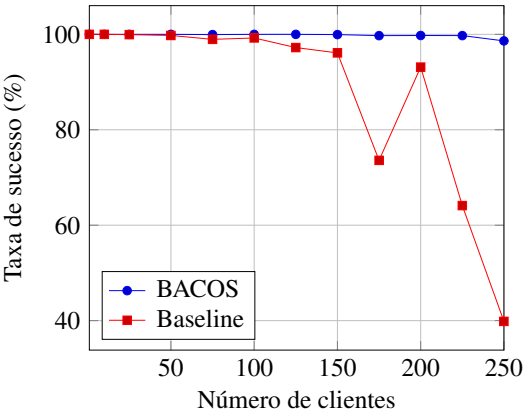
(c) Intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* para requisições de leitura com 95% de confiança.



(d) Intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* para requisições de escrita com 95% de confiança.



(e) Taxa de sucesso para requisições de leitura.



(f) Taxa de sucesso para requisições de escrita.

Figura 29 – Comparação do *throughput* entre BACOS e o *baseline* com padrão de requisições 50% leitura e 50% escrita.

diferença média de 34.8 operações por segundo e erro máximo de estimativa igual a 5.83 para requisições de escrita. O intervalo de confiança não incluiu o valor zero e a média da diferença entre as distribuições do BACOS e o *baseline* é positiva em ambos tipos de requisições. Assim,

aceita-se a hipótese nula H_0 e pode-se concluir com 95% de confiança que a estratégia BACOS foi melhor que o *baseline* em relação ao *throughput* para requisições com 50% de leitura e 50% de escrita. BACOS também melhorou em média 58.99% o *throughput* de leitura e em média 62.76% o *throughput* de escrita em relação ao *baseline*.

BACOS gerou resultados mais suaves ao longo do crescimento linear do número de clientes em relação a estratégia *baseline*. BACOS conseguiu manter o *throughput* mesmo sobre alta demanda de requisições com uma pequena queda da taxa de sucesso das operações de leitura e escrita, como pode-se perceber nas Figuras 29e e 29f. Ao contrário desse comportamento, a estratégia de balanceamento do *baseline* não conseguiu lidar com alta demanda de requisições. Por consequência, a taxa de sucesso das requisições diminuiu exponencialmente, ocasionando taxas abaixo de 40% para escrita e leitura. Além disso o *throughput* foi degradado assim como o tempo médio de resposta analisado anteriormente.

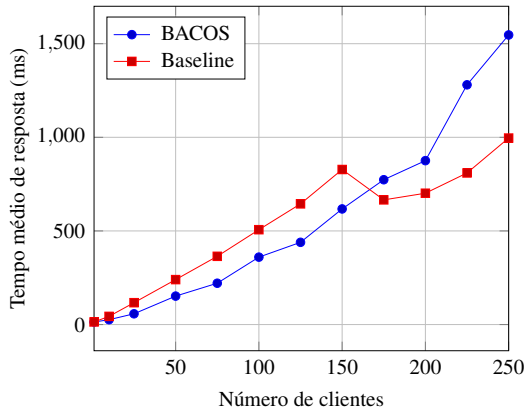
5.5.3 Cenário 2: carga de trabalho com taxa de leitura/escrita igual a 60%/40%

5.5.3.1 Tempo médio de resposta

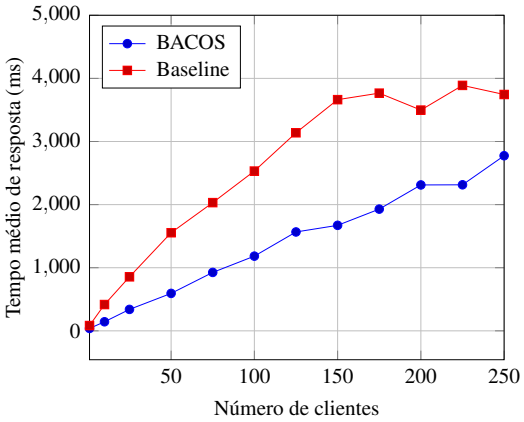
Os resultados obtidos do tempo médio de resposta oriundo do experimento do cenário 2 podem ser visualizados na Figura 30. As Figuras 30a e 30b mostram, respectivamente, as distribuições do tempo médio de resposta para requisições de leitura e escrita das estratégias *baseline* e BACOS.

Calculando-se a diferença das distribuições do tempo médio de resposta entre a estratégia *baseline* e BACOS para requisições de leitura e escrita, obtém-se o intervalo de confiança para a média da diferença das distribuições de tempo médio de resposta entre o *baseline* e BACOS mostrados nas Figuras 30c e 30d. Houve uma diferença média de -35.94 milissegundos e erro máximo de estimativa igual a 159.26 entre o *baseline* e o BACOS para requisições de leitura. A Figura 30c mostra que a diferença dos valores dos tempos médios de resposta entre 175 e 250 clientes contribuíram para que a diferença média do tempo médio de resposta de leitura entre as distribuições caísse drasticamente a ponto de gerar valor negativo. Em relação ao tempo médio de resposta de escrita, houve uma diferença média de 1115.125 milissegundos e erro máximo de estimativa igual a 384.36 como mostra a Figura 30d.

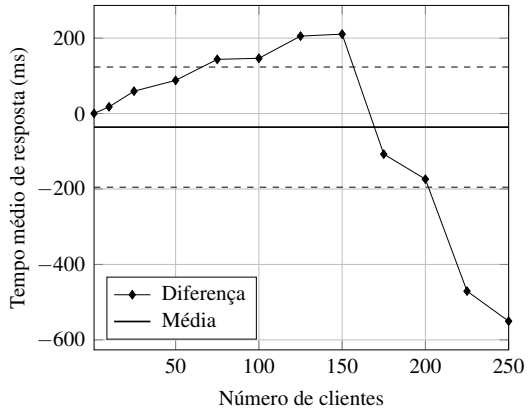
Analisando apenas requisições de leitura, o intervalo de confiança na Figura 30c incluiu o valor zero e a média da diferença entre as distribuições do *baseline* e BACOS tem valor



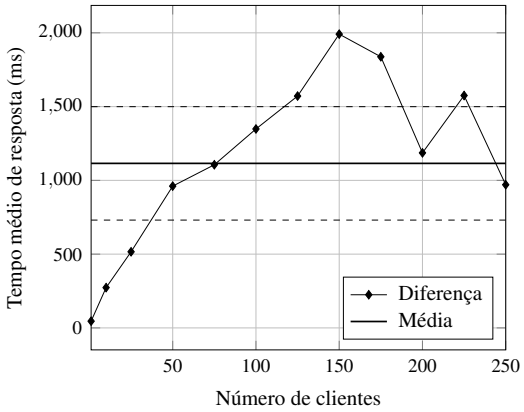
(a) Distribuições do tempo médio de resposta para operações de leitura.



(b) Distribuições do tempo médio de resposta para operações de escrita.



(c) Intervalo de confiança para a média da diferença das distribuições do tempo médio de resposta entre o *baseline* e BACOS para requisições de leitura com 95% de confiança.



(d) Intervalo de confiança para a média da diferença das distribuições do tempo médio de resposta entre o *baseline* e BACOS para requisições de escrita com 95% de confiança.

Figura 30 – Comparação do tempo médio de resposta entre o *baseline* e BACOS com padrão de requisições 60% leitura e 40% escrita.

negativo. Portanto, rejeitando a hipótese nula H_0 e aceitando a hipótese alternativa H_1, com 95% de confiança, conclui-se que ambas estratégias não são significantemente diferentes em relação a leitura para requisições com 60% de leitura e 40% de escrita. Apesar das estratégias não serem significantemente diferentes em relação a leitura, BACOS conseguiu melhorar em média 8.45% o tempo médio de resposta em relação ao *baseline*. O mau gerenciamento dos recursos por parte do *baseline* provocou uma anomalia no tempo médio de resposta de leitura quando submetido ao aumento da carga de trabalho entre 175 e 250 clientes. O resultado da queda da taxa de sucesso das requisições de leitura que será mostrado na próxima seção, aponta que a queda brusca do tempo médio de resposta de leitura entre 175 e 250 clientes foi devido a poucas amostras de requisições para estimar o tempo médio de resposta, pois só foi possível calcular o tempo médio de resposta das requisições concluídas com sucesso.

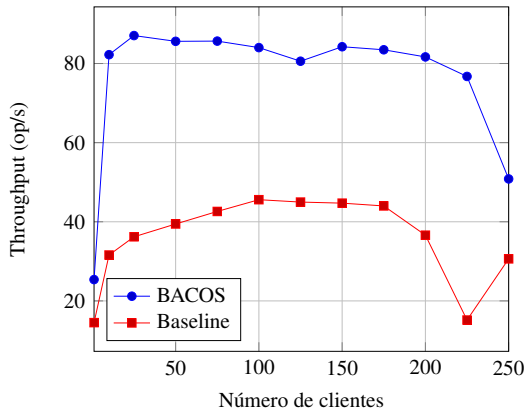
Em relação as operações de escrita, o resultado foi similar a todos os outros experimentos visto até agora. Ou seja, como mostrado na Figura 30d, o intervalo de confiança não incluiu o valor zero e a média da diferença entre as distribuições do *baseline* e BACOS foi positiva, consequentemente, aceita-se a hipótese nula H_0 e pode-se concluir com 95% de confiança que a estratégia BACOS foi melhor que o *baseline* em relação ao tempo médio de resposta para requisições de escrita com padrão de carga de 60% de leitura e 40% de escrita. Além disso, BACOS melhorou em média 22.05% o tempo médio de resposta de leitura e em média 60.04% o tempo médio de resposta de escrita em relação ao *baseline*.

5.5.3.2 Throughput e taxa de sucesso

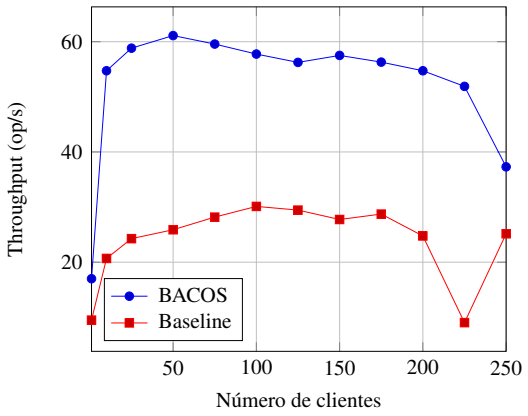
Os principais resultados obtidos do experimento podem ser visualizados na Figura 31. As Figuras 31a e 31b mostram, respectivamente, as distribuições do *throughput* para requisições de leitura e escrita das estratégias: *baseline* e *BACOS*. Para análise do *throughput*, foi considerado apenas o *throughput* das requisições realizadas com sucesso, computados a partir da taxa de sucesso de leitura e escrita mostradas nas Figuras 31e e 31f.

Calculando-se a diferença do *throughput* entre a estratégia BACOS e o *baseline* para requisições de leitura e escrita, obtém-se o intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline*, mostrados nas Figuras 31c e 31d. Veja que houve uma diferença média de 40.12 operações por segundo entre o *baseline*e o BACOS apresentando um erro máximo de estimativa igual a 8.63 para requisições de leitura e uma diferença média de 28.29 operações por segundo com um erro máximo de estimativa igual a 6.19. O intervalo de confiança não incluiu o valor zero e a média da diferença entre as distribuições do BACOS e o *baseline* foi positiva em ambos tipos de requisições. Logo, aceita-se a hipótese nula H_0 e pode-se concluir com 95% de confiança que a estratégia BACOS foi melhor que o *baseline* em relação ao *throughput* para requisições com 60% de leitura e 40% de escrita. BACOS também melhorou em média 52.19% o *throughput* de leitura e em média 56.21% o *throughput* de escrita em relação ao *baseline*.

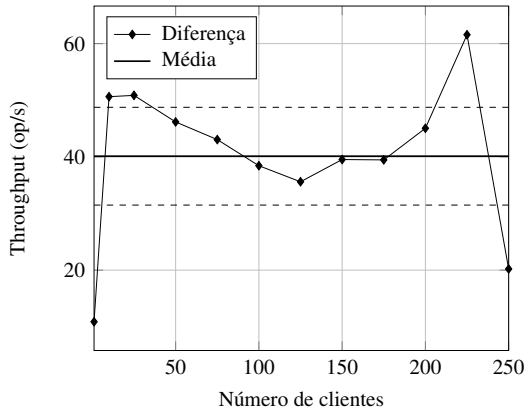
BACOS conseguiu manter um bom desempenho do *throughput* e taxa de sucesso das requisições de leitura e escrita durante o crescimento da carga de trabalho, com exceção do número de clientes entre 200 e 250. Esse comportamento foi esperado pelo fato da estratégia BACOS ter decidido previamente balancear a distribuição dos dados entre os dispositivos de armazenamento mediante da migração de objetos. Durante a migração de dados, a tendência



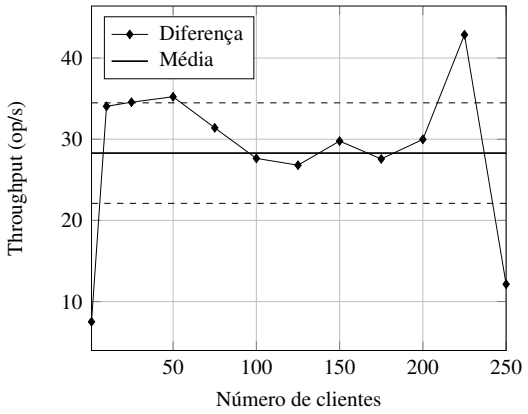
(a) Distribuições do *throughput* médio para operações de leitura.



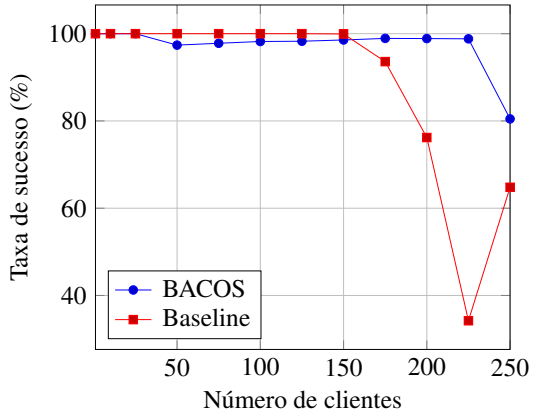
(b) Distribuições do *throughput* médio para operações de escrita.



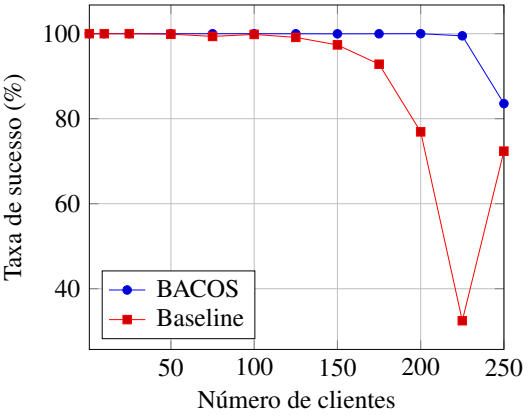
(c) Intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* para requisições de leitura com 95% de confiança.



(d) Intervalo de confiança para a média da diferença das distribuições do *throughput* entre BACOS e o *baseline* para requisições de escrita com 95% de confiança.



(e) Taxa de sucesso para requisições de leitura.



(f) Taxa de sucesso para requisições de escrita.

Figura 31 – Comparação do *throughput* entre BACOS e o *baseline* com padrão de requisições 60% leitura e 40% escrita.

é haver uma queda de desempenho, mas após esse processo, o esperado seria melhorar o desempenho geral do *object storage*.

5.6 Conclusão

Neste capítulo, foi verificada a efetividade da estratégia BACOS comparado com a estratégia base do sistema de armazenamento *OpenStack Swift* ao explorar o desempenho geral do sistema com diferentes padrões de acessos e volumes de requisições das operações de leitura e escrita em um ambiente de nuvem com dispositivos de armazenamento heterogêneos. Ainda, foi realizado uma análise sobre o desempenho do BACOS quando o conhecimento sobre o padrão da carga de trabalho não foi previamente registrado em sua base de treino.

Por meio dos resultados experimentais, pode-se concluir que o BACOS conseguiu melhorar o tempo médio de resposta e o *throughput* tanto para requisições de escrita quanto para leitura das aplicações em relação ao *baseline*, na grande maioria dos casos. Em geral, BACOS conseguiu melhores resultados sobre operações de escrita do que em operações de leitura por causa da estratégia de quórum do *OpenStack Swift* que exige apenas a confirmação da maioria simples para confirmar o sucesso de uma requisição de escrita. Os resultado não foram melhores para requisições de leitura devido ao método de seleção de réplica padrão do *Opentack Swift*, ou seja, o sistema escolhe aleatoriamente entre as réplicas disponíveis sem se importar com o desempenho dos dispositivos de armazenamento (YING *et al.*, 2017). Mesmo sobre uma carga de trabalho com leitura moderada, não registrada na base de treino da estratégia BACOS, ela conseguiu melhorar em média 8.45% o tempo médio de resposta de leitura e em média 52.19% o *throughput* de leitura da estratégia padrão do *OpenStack Swift*.

6 CONSIDERAÇÕES FINAIS

6.1 Conclusão

Este trabalho apresentou BACOS, uma estratégia de balanceamento de carga para sistemas de armazenamento baseado em objetos que são utilizados em ambientes de computação em nuvem. BACOS foi projetado para utilizar conhecimento prévio sobre o comportamento da carga de trabalho com a finalidade de melhorar o desempenho geral do sistema de armazenamento, tirando vantagem do desempenho heterogêneo dos dispositivos de armazenamento. BACOS é uma estratégia não intrusiva que utilize interfaces de alto nível de abstração do sistema capaz de mudar a demanda de requisições e a distribuição de dados de acordo com a variação dinâmica da carga de trabalho. A estratégia proposta busca atribuir uma quantidade de requisições de leitura e escrita para dispositivos de armazenamento com desempenho heterogêneos proporcionalmente à vazão de cada um. BACOS gerencia os recursos dos dispositivos de armazenamento de forma eficiente, visto que uma má gestão de recursos gera resultados imprevisíveis e queda de desempenho do sistema de armazenamento.

O desempenho da estratégia BACOS foi avaliado em uma nuvem privada, utilizando o sistema de armazenamento baseado em objetos *Openstack Swift* como caso de teste. O sistema *OpenStack Swift* é um *object storage* comercialmente popular e com oportunidades de pesquisa. Alguns experimentos variando a carga de trabalho e coletando métricas de desempenho foram conduzidos para avaliar a efetividade da estratégia BACOS comparado a estrategia padrão recomendada na documentação oficial do *OpenStack Swift* (SWIFT, 2017a). Basicamente, à estratégia padrão do *OpenStack Swift* atribui objetos e requisições a dispositivos de armazenamento proporcionalmente à capacidade de armazenamento total deles.

De um modo geral, BACOS conseguiu melhorar o tempo de resposta e a vazão das requisições tanto para requisições de leitura quanto para escrita. A estratégia obteve melhores resultados com requisições de escrita do que em requisições de leitura em virtude da estratégia de quorum adotado pelo *OpenStack Swift* para a confirmação de sucesso das requisições do tipo escrita. Além disso, BACOS conseguiu melhorar em média 8.5% o tempo de resposta e em média 52.19% a vazão para requisições do tipo leitura diante de uma carga desconhecida, ou seja, avaliado sobre um padrão de carga não registrada na base de treino do BACOS. Embora os experimentos tenham sido executados em uma infraestrutura de nuvem de baixa/média escala, BACOS não está limitado a uma pequena quantidade de servidores ou dispositivos de

armazenamento. Os resultados reforçam que a estratégia de balanceamento de carga BACOS consegue melhorar o desempenho geral de *object storages* em ambientes de computação em nuvem.

6.2 Trabalhos futuros

Existem algumas oportunidades de trabalhos futuros que derivam da pesquisa realizada nesse trabalho, incluindo i) Considerar outros tipos de recursos computacionais ao modelo que podem degradar o desempenho do sistema quando sobrecarregados, tais como CPU e memória principal. ii) Investigar o impacto do tamanho do objetos. iii) Adicionar suporte à predição da carga de trabalho com modelos de *forecast* para a construção de uma estratégia de balanceamento de carga proativo. iv) Explorar com mais profundidade o custo de migração através da configuração autônoma do parâmetro α, proposto no capítulo 4, independente do ambiente de implantação e v) conduzir um estudo sobre replicação e cache de objetos com o intuito de reduzir o acesso a objetos populares de forma dinâmica e escalável.

Um trabalho futuro que já se encontra em andamento, consiste na otimização do desempenho das requisições do tipo leitura mediante da integração do BACOS com a solução de seleção de réplicas chamada *Last Read Data Chunk Throughput and Total of Read Data Chunks* (LB-RLT) proposta em (ALMEIDA *et al.*, 2016). Esse trabalho é motivado pelo fato de que o algoritmo padrão implementado pelo *OpenStack Swift* é ingênuo e não leva em conta características de desempenho. Assim, o algoritmo LB-RLT considera o *throughput* corrente dos servidores de armazenamentos em um ambiente heterogêneo.

REFERÊNCIAS

ABADI, D. Consistency tradeoffs in modern distributed database system design: Cap is only part of the story. **Computer**, v. 45, n. 2, p. 37–42, Feb 2012. ISSN 0018-9162.

ALMEIDA, A. M. R.; CAVALCANTE, D. M.; SOUSA, F. R. C.; MACHADO, J. C. LB-RLT approach for load balancing heterogeneous storage nodes. In: **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)**. [S.l.: s.n.], 2016.

ALMES, G.; ZEKAUSKAS, M. J.; KALIDINDI, S. A round-trip delay metric for ippm. 1999.

AMAZON. **Amazon Elastic Block Store**. 2017. <aws.amazon.com/ebs>. Accessed: 2017-04-03.

AMAZON. **Amazon Elastic Compute Cloud**. 2017. <https://aws.amazon.com/pt/ec2/>. Accessed: 2017-04-03.

ANDERSON, E.; KALLAHALLA, M.; UYSAL, M.; SWAMINATHAN, R. Buttress: A toolkit for flexible and high fidelity i/o benchmarking. In: USENIX ASSOCIATION. **Proceedings of the 3rd USENIX Conference on File and Storage Technologies**. [S.l.], 2004. p. 4–4.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I. *et al.* A view of cloud computing. **Communications of the ACM**, ACM, v. 53, n. 4, p. 50–58, 2010.

ARNOLD, J. **Openstack swift: Using, administering, and developing for swift object storage**. [S.l.]: "O'Reilly Media, Inc.", 2014.

AXBOE, J. **FIO**. 2017. <https://github.com/axboe/fio>. Accessed: 2017-04-10.

AZAGURY, A.; DREIZIN, V.; FACTOR, M.; HENIS, E.; NAOR, D.; RINETZKY, N.; RODEH, O.; SATRAN, J.; TAVORY, A.; YERUSHALMI, L. Towards an object store. In: IEEE. **Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on**. [S.l.], 2003. p. 165–176.

BOSE, R.; ROY, S.; SARDDAR, D. On demand iops calculation in cloud environment to ease linux-based application delivery. In: SPRINGER. **Proceedings of the First International Conference on Intelligent Computing and Communication**. [S.l.], 2017. p. 71–77.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation computer systems**, Elsevier, v. 25, n. 6, p. 599–616, 2009.

CANONICAL. **Canonical**. 2015. <https://insights.ubuntu.com/2015/05/18/what-are-the-different-types-of-storage-block-object-and-file/>. Accessed: 2017-04-03.

CEPH. **Ceph Documentation**. 2017. <http://docs.ceph.com/docs/hammer/architecture/>. Accessed: 2017-04-03.

DECANDIA, G.; HASTORUN, D.; JAMPANI, M.; KAKULAPATI, G.; LAKSHMAN, A.; PILCHIN, A.; SIVASUBRAMANIAN, S.; VOSSHALL, P.; VOGELS, W. Dynamo: amazon's highly available key-value store. **ACM SIGOPS operating systems review**, ACM, v. 41, n. 6, p. 205–220, 2007.

DESAI, T.; PRAJAPATI, J. A survey of various load balancing techniques and challenges in cloud computing. **International Journal of Scientific & Technology Research**, Citeseer, v. 2, n. 11, p. 158–161, 2013.

DESHMUKH, S. C.; DESHMUKH, S. S. A survey: Load balancing for distributed file system. **International Journal of Computer Applications**, Foundation of Computer Science, v. 111, n. 5, 2015.

DEWAN, H.; HANSDAH, R. A survey of cloud storage facilities. In: IEEE. **Services (SERVICES), 2011 IEEE World Congress on**. [S.l.], 2011. p. 224–231.

DONG, B.; LI, X.; WU, Q.; XIAO, L.; RUAN, L. A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers. **J. Parallel Distrib. Comput.**, Academic Press, Inc., Orlando, FL, USA, v. 72, n. 10, p. 1254–1268, out. 2012. ISSN 0743-7315. Disponível em: <http://dx.doi.org/10.1016/j.jpdc.2012.05.006>.

DROPBOX. **Dropbox**. 2017. <https://www.dropbox.com/>. Accessed: 2017-04-03.

FACTOR, M.; METH, K.; NAOR, D.; RODEH, O.; SATRAN, J. Object storage: The future building block for storage systems. In: IEEE. **Local to Global Data Interoperability-Challenges and Technologies, 2005**. [S.l.], 2005. p. 119–123.

FARIAS, V. A. E. **MODELAGEM DE DESEMPENHO E DE ELASTICIDADE PARA BANCOS DE DADOS EM NUVEM**. Dissertação (Mestrado) — (Mestrado em Ciências da Computação) - Universidade Federal do Ceará, 2016.

FELBER, P.; KROPF, P.; SCHILLER, E.; SERBU, S. Survey on load balancing in peer-to-peer distributed hash tables. **IEEE Communications Surveys & Tutorials**, IEEE, v. 16, n. 1, p. 473–492, 2014.

FLEXISCALE. **FlexiScale Cloud Comp and Hosting**. 2017. <http://www.flexiscale.com/>. Accessed: 2017-04-03.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The google file system. In: ACM. **ACM SIGOPS operating systems review**. [S.l.], 2003. v. 37, n. 5, p. 29–43.

GIBSON, G. A.; METER, R. V. Network attached storage architecture. **Commun. ACM**, ACM, New York, NY, USA, v. 43, n. 11, p. 37–45, nov. 2000. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/353360.353362>.

GONG, C.; LIU, J.; ZHANG, Q.; CHEN, H.; GONG, Z. The characteristics of cloud computing. In: **2010 39th International Conference on Parallel Processing Workshops**. [S.l.: s.n.], 2010. p. 275–279. ISSN 0190-3918.

GOOGLE. **Google App Engine**. 2017. <https://appengine.google.com/>. Accessed: 2017-04-03.

GOOGLE. **Google Docs**. 2017. <https://docs.google.com/>. Accessed: 2017-04-03.

GROSSMAN, R. L.; GU, Y.; SABALA, M.; ZHANG, W. Compute and storage clouds using wide area high performance networks. **Future Generation Computer Systems**, Elsevier, v. 25, n. 2, p. 179–183, 2009.

GUNAWI, H. S.; AGRAWAL, N.; ARPACI-DUSSEAU, A. C.; ARPACI-DUSSEAU, R. H.; SCHINDLER, J. Deconstructing commodity storage clusters. In: IEEE COMPUTER SOCIETY. **ACM SIGARCH Computer Architecture News**. [S.l.], 2005. v. 33, n. 2, p. 60–71.

HDFS. **Hadoop Distributed File System (HDFS)**. 2017. <https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html>. Accessed: 2017-04-03.

HONG, I. **Consistent Hashing Algorithm**. 2017. <https://ihong5.wordpress.com/2014/08/19/consistent-hashing-algorithm/>. Accessed: 2017-04-03.

HONICKY, R.; MILLER, E. L. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In: IEEE. **Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International**. [S.l.], 2004. p. 96.

HOU, B.; CHEN, F.; WANG, R.; MESNIER, M.; OU, Z. Understanding i/o performance behaviors of cloud storage from a client's perspective. In: . [S.l.: s.n.], 2016.

HSIAO, H. C.; CHUNG, H. Y.; SHEN, H.; CHAO, Y. C. Load rebalancing for distributed file systems in clouds. **IEEE Transactions on Parallel and Distributed Systems**, v. 24, n. 5, p. 951–962, May 2013. ISSN 1045-9219.

JAIN, R. **The art of computer systems performance analysis : techniques for experimental design, measurement, simulation, and modeling**. New York: J. Wiley & sons, 1991. ISBN 0-471-50336-3. Disponível em: <http://opac.inria.fr/record=b1088381>.

JU, J.; WU, J.; FU, J.; LIN, Z.; ZHANG, J. A survey on cloud storage. **JCP**, v. 6, n. 8, p. 1764–1771, 2011.

KARGER, D.; LEHMAN, E.; LEIGHTON, T.; PANIGRAHY, R.; LEVINE, M.; LEWIN, D. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: ACM. **Proceedings of the twenty-ninth annual ACM symposium on Theory of computing**. [S.l.], 1997. p. 654–663.

KHATTAR, R. K.; MURPHY, M. S.; TARELLA, G. J.; NYSTROM, K. E. **Introduction to Storage Area Network, SAN**. [S.l.]: IBM Corporation, International Technical Support Organization, 1999.

KIM, H.; SESHADRI, S.; DICKEY, C. L.; CHIU, L. Evaluating phase change memory for enterprise storage systems: A study of caching and tiering approaches. In: **Proceedings of the 12th USENIX Conference on File and Storage Technologies**. Berkeley, CA, USA: USENIX Association, 2014. (FAST'14), p. 33–45. ISBN 978-1-931971-08-9. Disponível em: <http://dl.acm.org/citation.cfm?id=2591305.2591309>.

KUNKEL, J. M.; LUDWIG, T. Bottleneck detection in parallel file systems with trace-based performance monitoring. In: ____. **Euro-Par 2008 – Parallel Processing: 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 212–221. ISBN 978-3-540-85451-7. Disponível em: <http://dx.doi.org/10.1007/978-3-540-85451-7_23>.

LAKSHMAN, A.; MALIK, P. Cassandra: a decentralized structured storage system. **ACM SIGOPS Operating Systems Review**, ACM, v. 44, n. 2, p. 35–40, 2010.

LEHMANN, T. M.; GONNER, C.; SPITZER, K. Survey: interpolation methods in medical image processing. **IEEE Transactions on Medical Imaging**, v. 18, n. 11, p. 1049–1075, Nov 1999. ISSN 0278-0062.

LUA, E. K.; CROWCROFT, J.; PIAS, M.; SHARMA, R.; LIM, S. A survey and comparison of peer-to-peer overlay network schemes. **IEEE Communications Surveys & Tutorials**, IEEE, v. 7, n. 2, p. 72–93, 2005.

MEGHARAJ, M. K. G. A survey on load balancing techniques in cloud computing. **IOSR Journal of Computer Engineering**, Foundation of Computer Science, v. 18, n. 2, p. 55–61, 2016. ISSN 2278-0661.

MELL, P.; GRANCE, T. *et al.* The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.

MESBAHI, M.; RAHMANI, A. M. Load balancing in cloud computing: A state of the art survey. **International Journal of Modern Education and Computer Science**, Modern Education and Computer Science Press, v. 8, n. 3, p. 64, 2016.

MESNIER, M.; GANGER, G.; RIEDEL, E. Object-based storage: pushing more functionality into storage. **IEEE Potentials**, v. 24, n. 2, p. 31–34, April 2005. ISSN 0278-6648.

MESNIER, M.; GANGER, G. R.; RIEDEL, E. Object-based storage. **IEEE Communications Magazine**, IEEE, v. 41, n. 8, p. 84–90, 2003.

MICROSOFT. **Microsoft Azure**. 2017. <https://azure.microsoft.com/>. Accessed: 2017-04-03.

NUAIMI, K. A.; MOHAMED, N.; NUAIMI, M. A.; AL-JAROODI, J. A survey of load balancing in cloud computing: Challenges and algorithms. In: IEEE. **Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on**. [S.l.], 2012. p. 137–142.

NUAIMI, K. A.; MOHAMED, N.; NUAIMI, M. A.; AL-JAROODI, J. A partial replication load balancing algorithm for distributed data as a service (daas). In: **Int. Conf. on High Performance Computing and Simulation (HPCS)**. [S.l.: s.n.], 2013. p. 35–40.

PAIVA, J.; RODRIGUES, L. On data placement in distributed systems. **ACM SIGOPS Operating Systems Review**, ACM, v. 49, n. 1, p. 126–130, 2015.

PAULA, M. R. P.; RODRIGUES, E.; FARIAS, V. A. E.; SOUSA, F. R. C.; MACHADO, J. C. BACOS: A dynamic load balancingstrategy for cloud object storage. In: **Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)**. [S.l.: s.n.], 2017.

RACKSPACE. **Rackspace**. 2017. <https://www.rackspace.com/>. Accessed: 2017-04-03.

RAJESHWARI, B.; DAKSHAYINI, M. Comprehensive study on load balancing techniques in cloud. **Compusoft**, COMPUSOFT, An International Journal of Advanced Computer Technology, v. 3, n. 6, p. 900, 2014.

RICHARDSON, L.; RUBY, S. **RESTful web services**. [S.l.]: "O'Reilly Media, Inc.", 2008.

ROSADO, T.; BERNARDINO, J. An overview of openstack architecture. In: ACM. **Proceedings of the 18th International Database Engineering & Applications Symposium**. [S.l.], 2014. p. 366–367.

SALESFORCE. **Salesforce**. 2017. <https://www.salesforce.com>. Accessed: 2017-04-03.

SCIPY. **SciPy**. 2017. <https://docs.scipy.org/doc/scipy-0.18.1/reference/index.html>. Accessed: 2017-04-03.

SHEPLER, S.; EISLER, M.; ROBINSON, D.; CALLAGHAN, B.; THURLOW, R.; NOVECK, D.; BEAME, C. Network file system (nfs) version 4 protocol. **Network**, 2003.

SHU, J.; LI, B.; ZHENG, W. Design and implementation of an san system based on the fiber channel protocol. **IEEE Transactions on Computers**, IEEE, v. 54, n. 4, p. 439–448, 2005.

SHVACHKO, K.; KUANG, H.; RADIA, S.; CHANSLER, R. The hadoop distributed file system. In: **Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)**. Washington, DC, USA: IEEE Computer Society, 2010. (MSST '10), p. 1–10. ISBN 978-1-4244-7152-2. Disponível em: <http://dx.doi.org/10.1109/MSST.2010.5496972>.

SWIFT. **OpenStack Swift**. 2017. <https://docs.openstack.org/developer/swift/>. Accessed: 2017-04-03.

SWIFT. **OpenStack Swift Source Code**. 2017. <https://github.com/openstack/swift>. Accessed: 2017-04-03.

SWIFTSTACK. **SwiftStack**. 2017. <https://www.swiftstack.com/>. Accessed: 2017-04-03.

TAN, Z.; ZHOU, W.; FENG, D.; ZHANG, W. Aldm: Adaptive loading data migration in distributed file systems. **IEEE Transactions on Magnetics**, v. 49, n. 6, p. 2645–2652, June 2013. ISSN 0018-9464.

TOLIA, N.; ANDERSEN, D. G.; SATYANARAYANAN, M. Quantifying interactive user experience on thin clients. **Computer**, IEEE, v. 39, n. 3, p. 46–52, 2006.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: Towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, dez. 2008. ISSN 0146-4833. Disponível em: <http://doi.acm.org/10.1145/1496091.1496100>.

WANG, W.; XIE, T.; ZHOU, D. Understanding the impact of threshold voltage on mlc flash memory performance and reliability. In: ACM. **Proceedings of the 28th ACM international conference on Supercomputing**. [S.l.], 2014. p. 201–210.

WANG, Z.; CHEN, H.; FU, Y.; LIU, D.; BAN, Y. Workload balancing and adaptive resource management for the swift storage system on cloud. **Future Generation Computer Systems**, v. 51, p. 120 – 131, 2015. ISSN 0167-739X. Special Section: A Note on New Trends in Data-Aware Scheduling and Resource Provisioning in Modern {HPC} Systems. Disponível em: <//www.sciencedirect.com/science/article/pii/S0167739X14002404>.

WEIL, S. A.; BRANDT, S. A.; MILLER, E. L.; LONG, D. D. E.; MALTZAHN, C. Ceph: A scalable, high-performance distributed file system. In: **Proceedings of the 7th Symposium on Operating Systems Design and Implementation**. Berkeley, CA, USA: USENIX Association, 2006. (OSDI '06), p. 307–320. ISBN 1-931971-47-1. Disponível em: <http://dl.acm.org/citation.cfm?id=1298455.1298485>.

WEIL, S. A.; BRANDT, S. A.; MILLER, E. L.; LONG, D. D.; MALTZAHN, C. Ceph: A scalable, high-performance distributed file system. In: USENIX ASSOCIATION. **Proceedings of the 7th symposium on Operating systems design and implementation**. [S.l.], 2006. p. 307–320.

WEIL, S. A.; LEUNG, A. W.; BRANDT, S. A.; MALTZAHN, C. Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In: ACM. **Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing'07**. [S.l.], 2007. p. 35–44.

WINER, B. J.; BROWN, D. R.; MICHELS, K. M. **Statistical principles in experimental design**. [S.l.]: McGraw-Hill New York, 1971. v. 2.

WU, J.; PING, L.; GE, X.; WANG, Y.; FU, J. Cloud storage as the infrastructure of cloud computing. In: IEEE. **Intelligent Computing and Cognitive Informatics (ICICCI), 2010 International Conference on**. [S.l.], 2010. p. 380–383.

XIN, Q.; MILLER, E. L.; SCHWARZ, T.; LONG, D. D.; BRANDT, S. A.; LITWIN, W. Reliability mechanisms for very large storage systems. In: IEEE. **Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on**. [S.l.], 2003. p. 146–156.

XU, C.; WANG, W.; ZHOU, D.; XIE, T. An ssd-hdd integrated storage architecture for write-once-read-once applications on clusters. In: IEEE. **Cluster Computing (CLUSTER), 2015 IEEE International Conference on**. [S.l.], 2015. p. 74–77.

XU, Q.; AUNG, K. M. M.; ZHU, Y.; YONG, K. L. Building a large-scale object-based active storage platform for data analytics in the internet of things. **The Journal of Supercomputing**, Springer, v. 72, n. 7, p. 2796–2814, 2016.

YANG, B.; SONG, G.; ZHENG, Y.; WU, Y. Qosc: A qos-aware storage cloud based on hdfs. In: **2015 International Symposium on Security and Privacy in Social Networks and Big Data (SocialSec)**. [S.l.: s.n.], 2015. p. 32–38.

YING, L.; SRIKANT, R.; KANG, X. The power of slightly more than one sample in randomized load balancing. **Mathematics of Operations Research**, INFORMS, 2017.

ZHENG, Q.; CHEN, H.; WANG, Y.; ZHANG, J.; DUAN, J. COSBench: cloud object storage benchmark. In: ACM. **Proceedings of the 4th ACM/SPEC Int. Conf. on Performance Engineering**. [S.l.], 2013. p. 199–210.

ZHOU, J.; XIE, W.; NOBLE, J.; ECHO, K.; CHEN, Y. Suora: A scalable and uniform data distribution algorithm for heterogeneous storage systems. In: IEEE. **Networking, Architecture and Storage (NAS), 2016 IEEE International Conference on**. [S.l.], 2016. p. 1–10.

ZWOLENSKI, M.; WEATHERILL, L. *et al.* The digital universe: Rich data and the increasing value of the internet of things. **Australian Journal of Telecommunications and the Digital Economy**, Telecommunications Association, v. 2, n. 3, p. 47, 2014.