Avaliação de Algoritmos de Balanceamento de Carga para Ambientes em Nuvem

Alternative Title: Analysis of Workload Balancing Algorithms for Cloud Environments

Natan B. Morais
UNIFEI
Federal University of Itajubá
Itajubá, Brazil
morais_natan@hotmail.com

Rafael M. D. Frinhani
UNIFEI
Federal University of Itajubá
Itajubá, Brazil
frinhani@unifei.edu.br

Bruno T. Kuehne
UNIFEI
Federal University of Itajubá
Itajubá, Brazil
brunokuehne@unifei.edu.br

Dionisio M. L. Filho
UFMS
Federal University of Mato Grosso do Sul
Mato Grosso do Sul, Brazil
dionisio.leite@ufms.br

Maycon L. M. Peixoto
UFBA
Federal University of Bahia
Salvador, Brazil
mayconleone@dcc.ufba.br

Bruno G. Batista
UNIFEI
Federal University of Itajubá
Itajubá, Brazil
brunoguazzelli@unifei.edu.br

RESUMO

Computação em nuvem é um estilo de computação no qual os provedores de recursos podem oferecer serviços sob demanda de forma transparente e os clientes geralmente pagam de acordo com o uso. A nuvem introduz um novo nível de flexibilidade e escalabilidade para usuários abordando desafios como a rápida alteração em cenários de Tecnologia de Informação (TI) e a necessidade de reduzir custos e tempo no gerenciamento de infraestrutura. No entanto, para ser capaz de oferecer garantias de qualidade de serviço (QoS) sem limitar o número de requisições aceitas, os provedores devem ser capazes de escalonar de forma dinâmica e eficiente as requisições de serviços para serem executadas nos recursos disponíveis. O balanceamento de carga não é uma tarefa trivial, envolvendo desafios relacionados à demanda de serviço, a qual pode mudar instantaneamente, à modelagem de desempenho, e implantação e monitoramento de aplicações em recursos de TI virtualizados. Dessa forma, o objetivo deste artigo é desenvolver e avaliar algoritmos de balanceamento de carga para um ambiente em nuvem de forma a estabelecer um mapeamento mais eficiente entre as requisições de serviços e as máquinas virtuais que as executarão, garantindo a qualidade do serviço conforme definido no acordo de níveis de serviço. Nos experimentos verificou-se que o algoritmo proposto permitiu uma redução no tempo de execução das requisições, aumentando a quantidade de requisições atendidas durante o tempo de observação.

SBSI 2017 June 5^{th} – 8^{th}, 2017, Lavras, Minas Gerais, Brazil

Palavras-Chave

Computação em Nuvem; Balanceamento de Carga; Qualidade de Serviço.

ABSTRACT

Cloud computing is a computing style in which resource providers can offer on demand services in a transparent way and clients usually pay according to use. The cloud introduces a new level of flexibility and scalability for users addressing challenges such as rapid change in IT scenarios and the need to reduce costs and time in infrastructure management. However, to be able to offer quality of service (QoS) guarantees without limiting the number of requests accepted, providers must be able to dynamically and efficiently scale service requests to run on the available resources. Load balancing is not a trivial task, involving challenges related to service demand, which can shift instantly, to performance modeling, and deployment and monitoring of applications in virtualized IT resources. In this way, the aim of this paper is to develop and evaluate load balancing algorithms for a cloud environment in order to establish a more efficient mapping between the service requests and the virtual machines that will execute them, guaranteeing the quality of service as defined in the service level agreement. In the experiments it was observed that the proposed algorithm allowed a reduction in the requests execution time, increasing the number of requests served during the observation time.

CCS Concepts

•**Networks** → **Cloud computing;** *Packet scheduling;* •**Theory of computation** → **Scheduling algorithms;**

Keywords

Cloud Computing; Workload Balancing; Quality of Service.

1. INTRODUÇÃO

Nos últimos anos um dos tópicos mais discutidos na área de Tecnologia de Informação (TI) tem sido computação em nuvem. Segundo o NIST (*National Institute of Standards and Technology*), computação em nuvem é um modelo que permite ubiquidade, conveniência e acesso sob demanda para um conjunto de recursos compartilhados que são configuráveis e que podem ser rapidamente entregues com um esforço mínimo de gestão por parte dos usuários [9].

Subashini et al. [15] definem nuvem como um modelo de computação no qual os recursos de TI são fornecidos como um serviço aos clientes externos por meio da Internet. Os provedores desses recursos oferecem serviços sob demanda de forma transparente, uma vez que os clientes não têm ciência de onde e como esses serviços são executados.

A utilização de computação em nuvem permite aumentar dinamicamente a capacidade de prestação de serviços de uma empresa ou de atender às solicitações de serviços dos clientes, sem que estes precisem investir em infraestrutura como compra de *hardware*, de licenças de *software* ou treinamento de funcionários [11].

No entanto, por se tratar de um sistema distribuído que provê serviços, supõe-se que o sistema computacional que compõe uma nuvem opere apropriadamente, oferecendo desempenho tanto em termos de rapidez na resposta, quanto em termos de disponibilidade (minimizando a interrupção no oferecimento de serviço) e segurança (evitando perda de dados ou mensagens), a fim de conquistar a confiança e satisfação dos seus clientes. Para isso, os provedores de serviços devem garantir diferentes atributos de Qualidade de Serviço (*Quality of Service* – QoS).

O conceito de qualidade de serviço refere-se ao efeito provocado pelas características de desempenho de um serviço, ou seja, o conjunto de características de um sistema necessário para atingir uma determinada funcionalidade [16]. QoS pode ser descrito ainda como um conjunto de atributos que mensuram a qualidade de um fluxo de dados específico, como por exemplo, a largura de banda ou a prioridade atribuída a um determinado cliente. Esses atributos de um sistema podem ser medidos quantitativamente por meio de métricas e utilizados na definição de níveis de QoS [19]. Entre as métricas pode-se citar o tempo médio de resposta, *throughput*, o tempo médio de execução, utilização de recursos, perda de pacotes, *jitter*, latência de rede, segurança, entre outros.

Garantir QoS em um ambiente de nuvem não é uma tarefa trivial, pois existem diversos perfis de clientes com as mais variadas exigências de prestação de serviços [12]. O processo de definição de QoS começa com o estabelecimento dos parâmetros exigidos pelos clientes. Esses parâmetros são mapeados e negociados entre os componentes do sistema, assegurando que todos devem atingir um nível de QoS aceitável, definindo assim um acordo de nível de serviço (*Service Level Agreement – SLA*). Após a definição das cláusulas do SLA, os recursos são alocados e monitorados, havendo a possibilidade de renegociação caso as condições do sistema se alterem [1].

Dessa forma, a qualidade de um serviço e o cumprimento dos SLAs têm sido tópicos de grande interesse nos últimos anos tanto no âmbito acadêmico quanto industrial. Esses tópicos envolvem diversos desafios como, por exemplo, caracterização de clientes e serviços, controle de admissão, predição, análise e balanceamento de carga, provisionamento de recursos, otimização de sistemas, dentre outros. Considerando esses desafios, o foco do trabalho apresentado neste artigo envolve o balanceamento de carga em nuvem.

Atualmente, diversos serviços, sejam eles aplicativos, ferramentas de desenvolvimento e/ou infraestruturas computacionais, são desenvolvidos e fornecidos aos usuários/clientes de forma prática e rápida por meio da computação em nuvem. Estes clientes por sua vez, podem acessar esses serviços utilizando qualquer equipamento dotado de recursos computacionais e com acesso à Internet. Tem-se assim uma grande heterogeinidade de clientes e provedores, serviços e equipamentos que os acessam. Consequentemente, a prestação de serviço com qualidade torna-se mais complexa dada a demanda de requisições.

Em um determinado momento a demanda por serviços em uma nuvem pode variar de forma imprevisível, aumentando ou reduzindo o número de requisições. Para o cliente, toda a complexidade por trás de um ambiente em nuvem é transparente, visto que este se preocupa apenas com a disponibilidade, acesso, desempenho e custo de um serviço. O provedor, por sua vez, deve garantir que essa demanda seja atendida e alocada para execução nos recursos computacionais virtualizados de forma a cumprir o SLA, bem como o uso eficiente dos recursos, a um preço justo. Para isso, algoritmos eficientes de balanceamento de carga devem ser considerados.

O balanceamento de carga é extremamente importante em um ambiente em nuvem e tópico de grande interesse na comunidade científica e industrial. Há diversos trabalhos que discutem, propõem e analisam diferentes mecanismos envolvendo diversas áreas como, por exemplo, a Inteligência Artificial [10].

Em face ao exposto, o objetivo deste projeto é desenvolver e avaliar algoritmos de balanceamento de carga para um ambiente em nuvem, com o intuito de garantir o cumprimento dos acordos de níveis de serviço (SLAs), bem como o uso eficiente dos recursos computacionais. Esses algoritmos devem considerar desde premissas básicas presentes em algoritmos tradicionais, até técnicas de otimização e de inteligência artificial.

Todo o conteúdo abordado neste projeto é apresentado nas seguintes seções: um estudo sobre alguns trabalhos relacionados disponíveis na literatura é apresentado na seção 2; o ambiente utilizado para o desenvolvimento do estudo proposto é detalhado na seção 3; a seção 4 discorre sobre o planejamento dos experimentos executados; os resultados das experimentações definidas são discutidos na seção 5; por fim, as conclusões e premissas para trabalhos futuros são elaboradas na seção 6.

2. TRABALHOS RELACIONADOS

Devido à heterogeneidade física encontrada em um ambiente de computação em nuvem, é de essencial importância que dentre os recursos necessários para que um serviço de qualidade seja prestado esteja o balanceamento de carga. O balanceamento de carga é o principal responsável em distribuir a carga de serviços prestados entre seus recursos físicos, e consequentemente virtuais, de forma a evitar a subutilização ou superutilização dos recursos disponíveis, bem como garantir ao cliente a execução do SLA acordado [1].

A imprevisibilidade e constante mudança de comportamento na computação em nuvem, aliado à possibilidade de oferecer ao cliente a maior satisfação possível com relação à prestação de um serviço serve como fato de demonstra-

ção da importância do balanceamento de carga [7]. Desta forma, ele pode ser descrito como o responsável por fazer os recursos trabalharem corretamente independente da carga aplicada sobre o ambiente.

Contudo, a aplicação de mecanismos de balanceamento de carga envolve certos desafios. O primeiro refere-se à técnica de balanceamento implementada. Esta técnica, para que possa ser utilizada de forma eficiente e garantir os seus objetivos, deve ser analisada por uma série de parâmetros de verificação de qualidade [13], dos quais destacam-se:

- **Confiabilidade:** a técnica escolhida deve ser confiável, de forma que não haja erros e modificações indesejadas nos dados do cliente, uma vez que estes dados serão transferidos de um local para o outro durante a execução do balanceamento.
- **Adaptabilidade:** o balanceamento de carga deve se adaptar ao estado do ambiente, mudando a forma de operar ao encontrar situações adversas e atribuindo as cargas aos recursos computacionais no menor tempo possível, respeitando sempre o SLA.
- **Tolerância a Erros:** o algoritmo deve lidar com exceções. Desta forma, ao encontrar uma situação atípica e/ou erros durante a execução, ele deve continuar operante.
- ***Throughput*:** a técnica deve garantir que exista a maior taxa de transferência possível com o menor gasto. Segundo [13], se um algoritmo não aumenta o rendimento do sistema, ele não cumpre o seu propósito.
- **Tempo de Espera:** o tempo de espera de uma requisição para ser alocada em um recurso deve ser minimizado sempre que possível.

Há na literatura diversos trabalhos que tratam do balanceamento de carga, sendo que alguns propõem algumas inovações e outros analisam os que já existem no meio acadêmico. Por se tratar de um assunto complexo, alguns estudos apresentam os algoritmos com foco em *throughput*, QoS e análises da forma que diferentes métodos se comportam em cada aspecto do balanceamento. Todos estes trabalhos expressam os desafios por trás de diferentes metodologias de balanceamento de carga e servem de base para o desenvolvimento e análise de novas técnicas que garantam o SLA e o uso eficiente dos recursos.

Em [7] é definida a importância do balanceamento de carga devido ao comportamento característico de um ambiente em computação em nuvem, no qual, segundo os autores, o balanceamento de carga eficiente não depende inteiramente do método, mas do meio em que a nuvem está. Os autores apresentam as técnicas de Min-Min e Max-Min, explorando e comparando a forma com que eles agem. Nos experimentos realizados, verificou-se que o tempo de processamento é menor utilizando a técnica Max-Min em um ambiente simulado pelos autores.

Tyagi e Kumar [17] apresentam três métodos diferentes de balanceamento de carga, sendo eles: o *Round Robin*, método baseado na divisão do tempo em *quantum*; o *Equally Spread Current Execution Location*, que provê recursos de forma aleatória para o processamento de requisições levando em consideração a ociosidade do recurso; e o *Throttled Load Balancing Policy*, que define a ordem de adequação dos recursos para cada tipo de requisição, na qual se todos estiverem ocupados, a requisição entra em uma fila de espera, na qual aguarda para ser executada. Uma análise é feita sobre esses métodos considerando como variável de resposta o tempo médio de resposta. De acordo com os resultados, o método *Throttled* mostrou-se mais eficiente em relação aos demais.

Singh et al. [13] citam possíveis formas de balanceamento de carga, sendo a Estática responsável por um ambiente em nuvem pequeno, com Internet rápida e sem foco em atraso de comunicação; a Dinâmica, focada em reduzir o atraso de comunicação e tempo de execução, além de ser apropriada para ambientes maiores; e a Mista, que utiliza uma junção da concepção das duas anteriores. Deste modo, o ambiente analisado conta com um balaceamento dinâmico, o qual foi analisado pelos autores utilizando o método de agentes autonômos. Dos três agentes presentes neste trabalho, dois deles utilizam a computação natural, sendo a colônia de formigas a implementação escolhida. Análises verificaram que esse mecanismo mostrou-se uma forma eficiente de balanceamento, tanto em momentos de subutilização, quanto em momentos de superutilização dos recursos computacionais.

No trabalho apresentado em [6], os autores discutem uma forma totalmente diferente de lidar com o problema de balanceamento de carga. Eles propõem um método que utiliza um algoritmo genético, o qual segue os princípios básicos de população inicial, cruzamento entre genes e mutação. Nas experimentações, os autores utilizam um ambiente simulado com diferentes números de *Data Centers* disponíveis para verificar e comparar os métodos *Round Robin*, *First Come First Serve*, *Stochastic Hill Climbing* e o Algoritmo Genético proposto. Nesta análise é possível verificar que quanto mais *Virtual Machines - VMs* existem em um *Data Center*, maior a efetividade do Algoritmo Genético frente aos outros, e com o aumento gradual de recursos, a sua efetividade é gradativamente superior. Vale ressaltar que o *Stochastic Hill Climbing*, outra técnica de inteligência artificial, obteve resultados melhores que os outros algoritmos em praticamente todos os experimentos, ficando atrás apenas do Algoritmo Genético. O algoritmo FCFS foi o menos efetivo, sendo ele também o menos complexo para implementação.

Em [18], os autores não propõem nenhum método novo, mas apresentam uma extensão para o algoritmo de escalonamento de tarefas chamado HySARC^2. Esse algoritmo é composto por três partes: análise dos recursos disponíveis e agrupamento em *clusters*; provimento de diferentes grupos de tarefas para diferentes clusters; e escalonamento das atividades em cada cluster designado anteriormente. Os autores propõem uma mudança com base em uma atuação híbrida, com dois algoritmos diferentes para o escalonamento. A atuação do algoritmo é estritamente ligada à concepção de um dígrafo acíclico. O diferencial do HySARC^2 é a possibilidade de trabalhar com diferentes algoritmos de acordo com a necessidade de cada grupo de tarefas.

Com base nos trabalhos analisados, foi possível analisar as características de cada algoritmo, bem como as carências de cada um, visando o desenvolvimento de novos algoritmos capazes de lidar com diferentes demandas de serviços. Esses algoritmos são descritos na seção 4. Além disso, torna-se interessante a utilização de um ambiente em nuvem para as implementações, análises e validações propostas neste estudo. Para isso, o trabalho desenvolvido em [2] foi utilizado

como base.

3. AMBIENTE DE EXPERIMENTAÇÕES

O objetivo deste projeto é desenvolver e avaliar algoritmos de balanceamento de carga para ambientes em nuvem. Dessa forma, pode-se avaliar o comportamento do ambiente com variações na demanda de serviços, nos recursos computacionais e dos algoritmos de balanceamento de carga. Para as análises mencionadas, será utilizado o ReMM (*Resource Management Module*).

O ReMM é um módulo de gerenciamento de recursos para nuvem, inicialmente implementado no simulador CloudSim [4], que considera a manipulação dos recursos computacionais durante o tempo de execução, respeitando as métricas de qualidade de serviço definidas no contrato de níveis de serviço. Ele visa atender a qualquer demanda de requisições, aplicando tanto a escalabilidade horizontal quanto a vertical de forma dinâmica com impacto sobre o preço [2] [3]. Na escalabilidade horizontal altera-se a quantidade de máquinas virtuais disponíveis para um cliente. Por outro lado, na escalabilidade vertical altera-se a configuração dos recursos computacionais que compõem uma VM.

Na atual versão do ReMM, apenas o algoritmo *First Come First Served* (FCFS) está disponível para realizar o balanceamento de carga. Dessa forma, considerando a grande dinamicidade da demanda de serviços, torna-se necessário a aplicação de mecanismos de balanceamento de carga eficientes, com o objetivo de satisfazer o cliente, garantindo a QoS contratada a um preço justo, e o provedor, com o uso eficiente dos recursos disponíveis no sistema. A Figura 1 apresenta o funcionamento do ReMM.

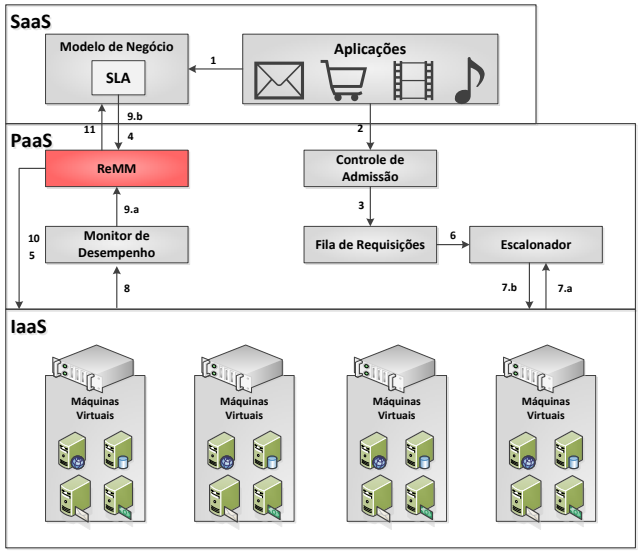

Figure 1: Funcionamento do ReMM [2].

Uma requisição do cliente será respondida por um provedor, o qual utiliza três camadas básicas para oferecer diferentes serviços: camada de aplicação (SaaS), camada de plataforma (PaaS) e camada de infraestrutura (IaaS). A camada de aplicação gerencia todas as aplicações que são oferecidas para os clientes. A camada de plataforma inclui as políticas de mapeamento e de escalonamento responsáveis por transferir à camada de infraestrutura as exigências feitas pelos clientes na definição do SLA e, dessa forma, alocar as máquinas virtuais para atendê-las. Por fim, a camada de infraestrutura controla a inicialização e remoção de instâncias, bem como a alteração dos recursos de uma VM de forma transparente ao cliente.

Inicialmente, cliente e provedor devem negociar um SLA, no qual as métricas de QoS bem como os detalhes do contrato, como por exemplo, níveis de segurança e o prazo para a execução do serviço, devem ser definidos (1). Após esse procedimento, diferentes clientes requisitam serviços a um provedor. O módulo Controle de Admissão é responsável por analisar a requisição e decidir se ela pode ser atendida ou não (2). Durante a sobrecarga do sistema, por exemplo, o provedor de serviços pode definir regras para decidir entre rejeitar novas requisições ou violar o SLA. A violação do SLA deve implicar em penalidades ao provedor. Se a requisição for aceita, ela será armazenada no módulo Fila de Requisições (3), onde ela receberá uma prioridade que definirá a sequência de execução (pode-se aplicar uma diferenciação de clientes, por exemplo).

Em seguida, o ReMM alocará os recursos virtuais baseado nas informações definidas no SLA (4), hospedando-os nos recursos físicos (5). Após a alocação dos recursos, o módulo Escalonador encaminha as requisições (6) para serem executadas nos recursos providos (7.a) (7.b) utilizando políticas de escalonamento. Em intervalos de tempo, o módulo Monitor de Desempenho coleta informações sobre as execuções das requisições (8) e as envia ao ReMM (9.a). Essas informações são comparadas com as informações definidas no SLA (9.b) e, caso alguma medição não esteja de acordo com o contrato, o ReMM ajusta dinamicamente a quantidade de recursos utilizando os algoritmos de escalabilidade horizontal e vertical, visando cumprir o SLA (10). Todas as alterações no sistema influenciam no custo definido no Modelo de Negócio (11). Maiores informações sobre o ReMM podem ser obtidas em [2] [3].

O foco do projeto descrito neste documento está no Escalonador, o qual utiliza algoritmos de escalonamento para atribuir as requisições para serem executadas nos recursos computacionais e garantir o balanceamento de carga do ambiente. A próxima seção descreve o planejamento dos experimentos executados.

4. METODOLOGIA

Conforme mencionado, o objetivo deste trabalho é a implementação e análise do comportamento de três algoritmos de balanceamento de carga distintos em um ambiente simulado em nuvem por meio do ReMM. Para essa comparação, dois algoritmos tradicionais e amplamente explorados por trabalhos acadêmicos foram utilizados: o FCFS, disponível na atual versão do ReMM, e o Min-Min[1]. Com base no estudo apresentado na seção 2, um novo algoritmo, chamado Min-Min QoS, foi proposto com o objetivo de abordar variáveis e situações que não são devidamente exploradas na implementação básica dos algoritmos tradicionais. Esses algoritmos são descritos a seguir.

4.1 Algoritmos

O *First Come First Served* (FCFS) é o algoritmo de maior

[1]Em testes preliminares verificou-se que o Min-Min apresentou resultados melhores quando comparado ao algoritmo Max-Min.

simplicidade em termos de implementação, sendo responsável por encaminhar uma requisição (*cloudlet*) para ser executada na próxima VM em uma fila de VMs disponíveis na nuvem. Ele mantêm um índice que percorre a lista, atribuindo de forma consecutiva a *cloudlet* à VM. Este algoritmo tende a ser o mais usado em situações de simulação, como é o caso deste projeto, pois ele costuma ser o algoritmo base para comparações envolvendo o balanceamento de carga dos simuladores e de implementações públicas de sistemas em nuvem. Isto ocorre pois é um algoritmo extremamente simples de ser implementado e ainda assim costuma ter resultados mais satisfatórios que alguns algoritmos de distribuição aleatória, nos quais uma máquina pode ser super ou subutilizada [5].

O algoritmo Min-Min analisa a requisição criando uma diferenciação entre requisições leves e pesadas de acordo com a exigência por processamento. Em seguida, o algoritmo envia para as VMs mais potentes, ou seja, de maior capacidade de processamento, as requisições mais leves, e para VMs menos potentes, as requisições mais pesadas, buscando assim um maior *throughput* do ambiente, pois os melhores recursos são liberados mais rapidamente, podendo assim executar novas tarefas. As VMs são ordenadas em uma lista utilizando o seu poder de processamento, a qual é dividida ao meio, a primeira metade possui as VMs menos potentes e a segunda possui as VMs mais potentes. Além disso, um índice é utilizado, assim como no FCFS, para distribuir as requisições entre as VMs de forma homogênea [14].

Por fim, tem-se o algoritmo proposto, chamado Min-Min QoS, o qual foi desenvolvido baseado no algoritmo Min-Min. Para isso, algumas alterações foram realizadas. A primeira modificação se dá por parte da exclusão da média fixa para consideração do tamanho das requisições, onde o algoritmo proposto utiliza as *cloudlets* já executadas para formar três novas médias chamadas: média baixa, média geral e média alta. Desta forma, pode-se atribuir cada faixa de valores para um grupo de VMs disponíveis, ou seja, toda *cloudlet* que tem tamanho abaixo da média baixa é enviada ao grupo de VMs mais potentes, as cloudlets entre a média baixa e média geral são enviadas ao grupo de VMs com capacidade de processamento intermediária, e assim por diante. Estas médias são atualizadas toda vez que uma nova *cloudlet* chega ao balanceador. Desta forma, o algoritmo se torna inteligente ao considerar o tamanho das mesmas.

Outra modificação consiste na utilização de um valor de tempo de execução médio para o sistema, o qual não é restritório, isto é, o ambiente pode e provavelmente não atenderá à este valor, mas buscará atender, definindo assim um prazo para a execução de uma requisição. Nos experimentos da próxima seção, foi definido um prazo de 500 milisegundos para a execução das requisições, com uma margem de tolerância de 10%. Esses valores foram definidos aleatoriamente. Desta forma, toda vez que uma nova requisição chega ao sistema, o balanceador faz a média de tempo de execução de todas as requisições que já foram executadas, e se a média for maior que o valor definido, um grupo com VMs mais potentes será atribuido para a execução da requisição.

4.2 Planejamento dos Experimentos

Para a execução dos experimentos um provedor composto por *data centers* foi configurado, os quais hospedam as máquinas virtuais que executam as requisições de serviço. Os recursos físicos foram considerados ilimitados, e dessa forma, todas as requisições são aceitas e executadas pelas VMs. Além disso, não há diferenciação de usuários por meio de prioridades. As configurações dos *data centers* são apresentadas na Tabela 1. Vale destacar que no ReMM a capacidade do processamento é dada em MIPS (Milhões de Instruções por Segundo).

Table 1: Especificação das máquinas físicas presentes nos *data centers*.

Processador	Intel Xeon 8 cores 2.6 Ghz
Capacidade de Processamento	10.000 MIPs/core
Memória Principal	32GB
Disco	2TB
S.O.	Ubuntu Server 11.10
Virtualizador	Xen 4.1

As máquinas virtuais utilizadas foram modeladas baseadas nos tipos de instâncias M3 da Amazon[2]. A Tabela 2 apresenta as configurações de cada VM.

Table 2: Especificação das instâncias.

Instância	Núcleo Virtual	Memória Principal (GB)	Disco SSD
m3.medium	1	3,75	1 x 4
m3.large	2	7,5	1 x 32
m3.xlarge	4	15	2 x 40
m3.2xlarge	8	30	2 x 80

As requisições em tempo real, leves e pesadas, são dinamicamente criadas e enviadas durante a simulação. Esse tipo de submissão possibilita a criação de simulações mais realistas, pois torna o comportamento do usuário mais próximo do que se espera em um ambiente real[3].

Os experimentos apresentados na próxima seção foram conduzidos com o objetivo de avaliar métricas relacionadas ao desempenho do serviço utilizando a infraestrutura de um provedor. Desta forma, foram consideradas as seguintes variáveis de resposta:

- **Tempo Médio de Execução (TME):** média do tempo gasto em milisegundos na execução das requisições de serviço durante o tempo de experimentação.
- **Quantidade de Requisições Atendidas:** taxa média de requisições atendidas por um ambiente em nuvem durante o tempo de observação.

Um planejamento fatorial completo foi utilizado seguindo a metologia apresentada por [8], na qual os fatores correspondem as características do ambiente analisado e os níveis são as possíveis variações que o ambiente pode apresentar. Dessa forma, 2 fatores e seus respectivos níveis foram considerados e apresentados na Tabela 3.

Table 3: Fatores e níveis.

Fatores	Níveis
Algoritmo	FCFS, Min-Min ou Min-Min QoS
Instância	*m3.medium*, *m3.large*, *m3.xlarge* ou *m3.2xlarge*

Na Tabela 3, o fator Algoritmo define se as requisições, as quais podem ser leves ou pesadas, serão enviadas para

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

[3]Outros tipos de cargas podem ser exploradas em trabalhos futuros.

serem executadas nas Instâncias disponíveis (*m3.medium*, *m3.large*, *m3.xlarge* ou *m3.2xlarge*) por meio dos algoritmos de balanceamento FCFS, Min-Min ou Min-Min QoS.

Outro fator importante é o número de VMs disponíveis para atender a demanda de serviços. Para esse fator foi considerado um valor fixo de 500 VMs[4], as quais são agrupadas de acordo com os casos definidos.

Portanto, tem-se um sistema em nuvem com 500 VMs, as quais foram organizadas em quantidades distintas de acordo com o cenário (Caso), permitindo avaliações dos algoritmos de balanceamento na predominância de diferentes tipos de máquinas virtuais. A predominância de um determinado tipo de VM no ambiente é dada por:

- **Divisão Igualitária (Caso 1):** das 500 VMs disponíveis no ambiente, 125 pertencem a cada um dos quatro tipos apresentados na Tabela 2. Portanto, tem-se neste caso a situação de um ambiente em nuvem com a mesma quantidade de VMs para todos os tipos.

- **Majoritariamente Pequena (Caso 2):** neste caso tem-se que a maior parte do ambiente concentra VMs de menor poder de processamento, sendo 200 VMs do tipo *m3.medium*, 200 do *tipo m3.large*, 50 do *tipo m3.xlarge* e 50 *m3.2xlarge*. Dessa forma, 80% das VMs do ambiente são dos dois tipos com menor potência computacional. Este tipo de ambiente sugere um gasto menor com recursos computacionais, tendo apenas um pequeno valor alocado para recursos mais potentes, possivelmente necessário para situações mais críticas ou para aliviar problemas como fila de requisições para serem atendidas.

- **Majoritariamente Média (Caso 3):** para este caso tem-se a predominância de VMs enquadradas na parte média com relação à potência de processamento, utilizando mais as instâncias dos tipos *m3.large* e *m3.xlarge*, com 200 VMs cada. Para os dois extremos, *m3.medium* e *m3.2xlarge*, há um total 100 VMs disponíveis, 50 VMs para cada tipo.

- **Majoritariamente Grande (Caso 4):** por fim, existe o caso em que 200 VMs *m3.xlarge* e 200 VMs *m3.2xlarge* compõem o ambiente e apenas 50 VMs são atribuídas para cada um das duas instâncias de menor valor de processamento (*m3.medium* e *m3.large*). Dessa forma, têm-se um ambiente com maior poder de processamento acumulado.

Cada experimento foi executado 10 vezes, com tempo de observação de aproximadamente 5600 segundos ou 1,5 hora. Por meio das 10 repetições percebeu-se que os resultados obtidos para cada variável de resposta não apresentavam grandes variações. Dessa forma, foram calculadas a média, o desvio padrão e o intervalo de confiança de 95% de cada configuração dos experimentos.

5. ANÁLISE DOS RESULTADOS

À partir do planejamento especificado na seção anterior foi possível modelar um ambiente de experimentação com diferentes agrupamentos de VMs, permitindo comparar os algoritmos de balanceamento por meio das variáveis de resposta.

A Figura 2 e a Tabela 4 apresentam os resultados das experimentaçoes considerando o tempo médio de execução das requisições, em milisegundos, durante o tempo de observação. Verifica-se que o algoritmo Min-Min QoS obteve os menores tempos para todos os casos analisados, com exceção do quarto caso, onde não há diferença estatística entre os algoritmos Min-Min QoS e o FCFS. Além disso, o algoritmo proposto buscou manter os tempos de execução das requisições o mais próximo possível do prazo definido.

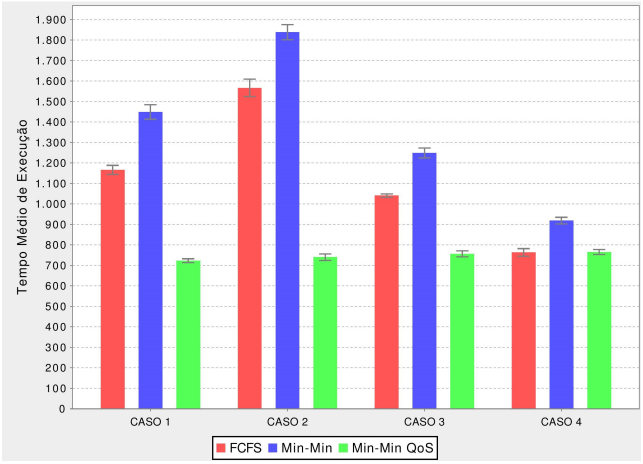

Figure 2: Tempo médio de execução para cada algoritmo.

Table 4: Comparação dos tempos médios de execução das requisições com os algoritmos de balanceamento.

Tipo de VM	Algoritmo	TME	Desvio Padrão	Intervalo de Confiança	Limite Inferior	Limite Superior
Caso 1	FCFS	1166,21	35,63	22,08	1144,13	1188,30
Caso 1	Min-Min	1448,80	56,56	35,05	1413,75	1483,85
Caso 1	Min-Min QoS	722,94	15,10	9,36	713,58	732,30
Caso 2	FCFS	1566,32	68,36	42,37	1523,95	1608,69
Caso 2	Min-Min	1838,16	59,51	36,88	1801,27	1875,04
Caso 2	Min-Min QoS	740,22	25,93	16,07	724,15	756,29
Caso 3	FCFS	1040,57	14,12	8,75	1031,82	1049,32
Caso 3	Min-Min	1248,56	38,94	24,13	1224,43	1272,69
Caso 3	Min-Min QoS	756,44	23,34	14,46	741,98	770,91
Caso 4	FCFS	762,97	30,72	19,04	743,93	782,00
Caso 4	Min-Min	918,62	26,23	16,26	902,36	934,88
Caso 4	Min-Min QoS	765,30	19,09	11,83	753,47	777,13

Para o Caso 1, no qual a quantidade de máquinas virtuais *m3.medium*, *m3.large*, *m3.xlarge* e *m3.2xlarge* foi a mesma, o algoritmo Min-Min QoS reduziu o TME das requisições em aproximadamente 38% quando comparado ao algoritmo FCFS e em 50% em relação ao Min-Min. Essa redução foi de 52% e 60%, respectivamente, para os experimentos com as premissas definidas no Caso 2, no qual houve a predominância de VMs com menor potência computacional (*m3.medium* e *m3.large*). Para o Caso 3, no qual houve a predominância de VMs com potência computacional intermediária dentre os quatro tipos disponíveis (*m3.large* e *m3.xlarge*), houve uma redução no TME de aproximadamente 27% do algoritmo Min-Min QoS para o FCFS e de 39% do Min-Min QoS para o Min-Min. Por fim, no Caso 4, no qual a quantidade de VMs dos tipos *m3.xlarge* e *m3.2xlarge* foi predominante, é

[4]Outras configurações e quantidades de VMs podem ser exploradas em trabalhos futuros.

possível afirmar que não houve diferenças estatísticas entre os algoritmos Min-Min QoS e FCFS, uma vez que os intervalos de confiança de ambos se sobrepuseram. Por outro lado, houve uma redução de aproximadamente 17% do algoritmo Min-Min QoS em relação ao Min-Min. A Tabela 5 apresenta os percentuais utilizados nas comparações.

Table 5: Percentual de (-)redução no TME proporcionado pelo algoritmo Min-Min QoS.

Cenários	FCFS	Min-Min
Caso 1	-38%	-50%
Caso 2	-52%	-60%
Caso 3	-27%	-39%
Caso 4	0%	-17%

Com relação à quantidade de requisições atendidas, Figura 3 e Tabela 6, quanto menor o tempo gasto para a execução de uma requisição, maior o *throughput* do ambiente, ou seja, maior a quantidade de requisições atendidas. Dessa forma, o algoritmo Min-Min QoS apresentou um desempenho superior em relação aos outros nos três primeiros casos das experimentações. Apenas no último caso esse algoritmo não foi superior ao FCFS, assim como na análise anterior.

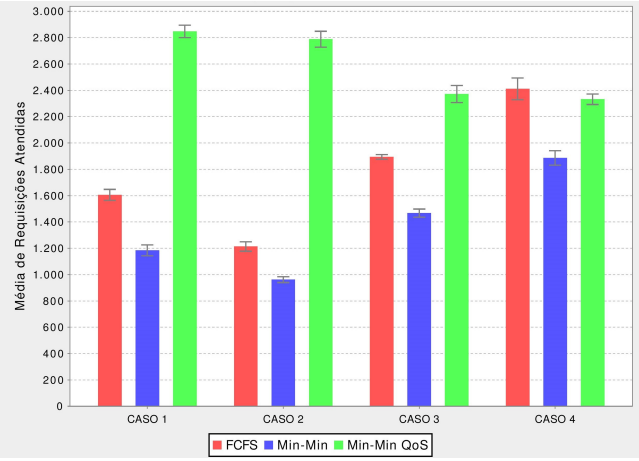

Figure 3: Média de requisições atendidas pelo sistema com cada algoritmo.

Table 6: Comparação das médias de requisições atendidas com os algoritmos de balanceamento.

Tipo de VM	Algoritmo	Média	Desvio Padrão	Intervalo de Confiança	Limite Inferior	Limite Superior
Caso 1	FCFS	1605,30	68,07	42,19	1563,11	1647,49
Caso 1	Min-Min	1185,20	66,45	41,19	1144,01	1226,39
Caso 1	Min-Min QoS	2847,20	75,90	47,04	2800,16	2894,24
Caso 2	FCFS	1213,10	57,37	35,56	1177,54	1248,66
Caso 2	Min-Min	961,90	35,99	22,31	939,59	984,21
Caso 2	Min-Min QoS	2788,40	97,29	60,30	2728,10	2848,70
Caso 3	FCFS	1894,10	29,84	18,50	1875,60	1912,60
Caso 3	Min-Min	1467,30	49,45	30,65	1436,65	1497,95
Caso 3	Min-Min QoS	2372,40	104,34	64,67	2307,73	2437,07
Caso 4	FCFS	2411,10	134,44	83,33	2327,77	2494,43
Caso 4	Min-Min	1886,40	88,14	54,63	1831,77	1941,03
Caso 4	Min-Min QoS	2332,00	64,12	39,74	2292,26	2371,74

No Caso 1, o algoritmo Min-Min QoS proporcionou um aumento de aproximadamente 77% no número de requisições atendidas em relação ao algoritmo FCFS e de 140% considerando o Min-Min. Para o Caso 2, esse aumento foi de aproximadamente 130% e 190%, respectivamente, enquanto que no Caso 3 houve um aumento de 25% (FCFS) e 62% (Min-Min). Por fim, no Caso 4, assim como mencionado anteriormente, os limites inferior do algoritmo FCFS e superior do Min-Min QoS se sobrepuseram, não definindo diferenças estatísticas entres eles. Porém, em relação ao seu algoritmo base, o Min-Min, o algoritmo proposto neste artigo, o Min-Min Qos, proporcionou um aumento na quantidade de requisições atendidas de aproximadamente 24%. A Tabela 7 apresenta os valores discutidos.

Table 7: Percentual de (+)aumento na média de requisições atendidas proporcionado pelo algoritmo Min-Min QoS.

Cenários	FCFS	Min-Min
Caso 1	+77%	+140%
Caso 2	+130%	+190%
Caso 3	+25%	+62%
Caso 4	0%	+24%

Uma análise interessante considera o comportamento de cada algoritmo nos diferentes casos com a alternância no tipo de potência computacional predominante. À medida em que a potência computacional do provedor tornou-se maior, os algoritmos FCFS e Min-Min melhoraram os respectivos desempenhos, expressos pelos tempos médios de execução e quantidade média de requisições atendidas, diminuindo as diferenças estatísticas para o algoritmo Min-Min QoS.

O princípio do algoritmo FCFS consiste em distribuir a carga para o recurso computacional disponível no momento. Dessa forma, a melhora no seu desempenho com o alternância dos casos ocorreu em razão da maior concentração de instâncias de maior poder de processamento, ou seja, mais requisições foram executadas em VMs mais potentes, o que evidentemente reduziu o TME e, consequentemente, permitiu a execução de uma quantidade maior de requisições.

Considerando o algoritmo Min-Min, a melhora no desempenho com a alternância dos casos ocorreu pelo fato do algoritmo ser extremamente simples. Como a base do algoritmo é dividir todas as VMs em apenas dois grupos de igual quantidade, nos casos finais existe a predominância de VMs mais potentes nos dois grupos e, consequentemente, há uma maior quantidade de requisições pesadas sendo executadas em recursos dotados de maior potência computacional, o que diminui o TME destas e também reduz as filas nas instâncias, aumentando a quantidade de requisições atendidas.

Por fim tem-se o algoritmo Min-Min QoS, no qual verificou-se que o seu desempenho manteve-se o mesmo para os Casos 3 e 4, uma vez que não houve diferenças estatísticas entre eles com relação aos TMEs e à quantidade média de requisições atendidas. Esse algoritmo analisa o tipo de requisição e os recursos computacionais disponíveis. Por essa razão, ele lida de forma diferente com cada tipo de instância do ambiente, agrupando-as de acordo com a potência computacional, ao contrário do Min-Min, que divide a quantidade de VMs disponíveis em dois grupos. Além disso, esse algoritmo define um prazo, o qual serve de base para a execução das requisições. Dessa forma, o sistema busca executar as requisições no tempo mais próximo possível do prazo, permitindo que instâncias mais potentes possam ser alocados para requisições que originalmente seriam executadas em grupos menos potentes, limitando a perda de desempenho.

6. CONCLUSÕES

A computação em nuvem introduz um novo nível de flexibilidade e escalabilidade nas organizações de TI. Esta flexibilidade ajuda a solucionar desafios que os clientes e provedores de serviços enfrentam, que incluem a rápida alteração de cenários de TI, redução do custo e tempo com gerenciamento de infraestrutura. Porém, uma vez que as requisições de serviço dos clientes são executadas nas máquinas virtuais disponíveis no provedor, faz-se necessário o estudo e análise de mecanismos de balanceamento de carga para que o mapeamento entre as requisições e as VMs seja o mais eficiente possível, garantindo a QoS definida no SLA. Por essa razão, este trabalho desenvolveu e comparou três algoritmos de balanceamento de carga utilizando diferentes cenários de experimentações através do ReMM.

De acordo com os resultados, o algoritmo proposto chamado Min-Min QoS obteve os melhores resultados com relação ao tempo médio de execução das requisições e à quantidade média de requisições atendidas.

Quatro configurações distintas de predominância dos tipos de máquinas virtuais foram definidas, as quais impactaram sobre a capacidade de processamento do provedor. Dessa forma, foi possível comparar e analisar o comportamento dos algoritmos FCFS, Min-Min e Min-Min QoS com variações na capacidade de processamento, explorando a heterogeneidade de um ambiente em nuvem.

O algoritmo Min-Min QoS apresentou uma maior superioridade em relação aos outros algoritmos em ambientes com predominância de máquinas virtuais com menor potência computacinal como visto nos experimentos com o Caso 2. No entanto, à medida que aumentou-se a potência computacional das VMs nos experimentos com o Casos 3 e 4, reduziu-se os percentuais numéricos entre os algoritmos, ou seja, as diferenças entre os resultados obtidos por meio das variáveis de resposta foram menores.

Outros estudos serão conduzidos a partir dos resultados e constatações encontradas durante o desenvolvimento deste trabalho. Eles incluem o desenvolvimento de um protótipo utilizando máquinas reais, o que permitirá uma comparação de dados oriundos de ambientes simulados e prototipados, além de análises sobre diferentes topologias de redes, demandas de serviços e recursos computacionais. Outros algoritmos de balanceamento de carga serão desenvolvidos, incluindo técnicas de inteligência artificial e de otimização.

7. AGRADECIMENTOS

Os autores gostariam de agradecer a infraestrutura e o apoio financeiro fornecido pela UNIFEI.

8. REFERENCES

[1] D. Ardagna, G. Casale, M. Ciavotta, J. F. Pérez, and W. Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *Journal of Internet Services and Applications*, 5(1):1–17, 2014.

[2] B. G. Batista, J. C. Estrella, C. H. G. Ferreira, D. M. Leite Filho, L. H. V. Nakamura, S. Reiff-Marganiec, M. J. Santana, and R. H. C. Santana. Performance evaluation of resource management in cloud computing environments. *PloS one*, 10(11):21, 2015.

[3] B. G. Batista, C. H. G. Ferreira, D. C. M. Segura, D. M. Leite Filho, and M. L. M. Peixoto. A qos-driven approach for cloud computing addressing attributes of performance and security. *Future Generation Computer Systems*, 68:260–274, 2017.

[4] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[5] S. Dam, G. Mandal, K. Dasgupta, and P. Dutta. An ant colony based load balancing strategy in cloud computing. pages 403–413, 2014.

[6] K. Dasgupta, B. Mandal, P. Dutta, J. K. Mandal, and S. Dam. A genetic algorithm (ga) based load balancing strategy for cloud computing. *Procedia Technology*, 10:340–347, 2013.

[7] P. G. Gopinath and S. K. Vasudevan. An in-depth analysis and study of load balancing techniques in the cloud computing environment. *Procedia Computer Science*, 50:427–432, 2015.

[8] R. Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. New York, NY, USA, Wiley, 1991.

[9] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800:145, 2011.

[10] L. Nema, A. Sharma, and S. Jain. Load balancing algorithms in cloud computing: An extensive survey. *International Journal of Engineering Science*, 7463, 2016.

[11] J. W. Rittinghouse and J. F. Ransome. *Cloud computing: implementation, management, and security*. CRC press, 2016.

[12] R. R. Selmic, V. V. Phoha, and A. Serwadda. Quality of service. pages 179–196, 2016.

[13] A. Singh, D. Juneja, and M. Malhotra. Autonomous agent based load balancing algorithm in cloud computing. *Procedia Computer Science*, 45:832–841, 2015.

[14] S. I. Singh, T. C. Abraham, and N. C. S. N. Iyengar. A review: Different improvised min-min load balancing algorithm in cloud computing environment. *Journal of Computer and Mathematical Sciences*, 7(11):540–550, 2016.

[15] S. Subashini and V. Kavitha. A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1):1–11, 2011.

[16] A. S. Tanenbaum. Computer networks, 5-th edition. *ed: Prentice Hall*, 2011.

[17] V. Tyagi and T. Kumar. Ort broker policy: Reduce cost and response time using throttled load balancing algorithm. *Procedia Computer Science*, 48:217–221, 2015.

[18] M.-A. Vasile, F. Pop, R.-I. Tutueanu, V. Cristea, and J. Kołodziej. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems*, 51:61–71, 2015.

[19] T. Weber. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. *Relatório técnico, Instituto de Informática UFRGS*, 2002.