UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Modelo de Escalonamento Aplicativo para Bag of Tasks em Ambientes de Nuvem Computacional

Maicon Ança dos Santos

Pelotas, 2017

Maicon Ança dos Santos

Modelo de Escalonamento Aplicativo para Bag of Tasks em Ambientes de Nuvem Computacional

Dissertação apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação

Orientador: Prof. Dr. Gerson Geraldo H. Cavalheiro

Pelotas, 2017

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

S237m Santos, Maicon Ança dos

Modelo de escalonamento aplicativo para Bag of Tasks em ambientes de nuvem computacional / Maicon Ança dos Santos ; Gerson Geraldo Homrich Cavalheiro, orientador. — Pelotas, 2016.

84 f. : il.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2016.

1. Nuvem computacional. 2. Bag of Tasks. 3. Escalonamento. 4. Consolidação de tarefas. I. Cavalheiro, Gerson Geraldo Homrich, orient. II. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Universidade Federal de Pelotas
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

ATESTADO

ATESTO para os devidos fins, que **MAICON ANÇA DOS SANTOS** foi aprovado pela Banca Examinadora da Defesa de Dissertação realizada em 14 de Julho de 2016, intitulada: "Modelo de Escalonamento Aplicativo para Bag of Tasks em Ambientes de Nuvem Computacional", sob a orientação do Prof. Dr. Gerson Cavalheiro. Realizadas as correções recomendadas pela Comissão Examinadora, o trabalho será encaminhado para homologação pelo Colegiado do Programa de Pós-Graduação em Computação. Igualmente, informamos que os demais requisitos necessários para a obtenção do grau de **Mestre em Ciência da Computação** pela Universidade Federal de Pelotas foram cumpridos.

Pelotas, 14 de Julho de 2016

Ricardo Matsumura de Araujo
Coordenador do Programa de Pós-Graduação em Computação

IFSUL
Pró-reitoria de Gestão de Pessoas
CONFERE COM O ORIGINAL
Em 15 / 07 / 2016

Hilbert David de Oliveira Sousa
Coordenador de Desenvolvimento de Pessoas
PROGEP / IFSUL

Dedico este trabalho aos meus amados Rosane e Mathias.

AGRADECIMENTOS

Com mais esta etapa de aprendizado concluída, gostaria de deixar registrado meus sinceros agradecimentos a todos que, de uma forma ou outra, estiveram presentes nessa empreitada.

Primeiro, agradeço a Deus, fonte de toda a sabedoria e conhecimento. Sem Ele nada disto seria possível em minha vida.

Quero agradecer ao meu orientador, Prof. Dr. Gerson Cavalheiro, por ter apostado firmemente neste trabalho e ter me incentivado e cobrado mesmo nos momentos mais difíceis.

À minha amada esposa, Rosane, por todo amor, carinho e paciência durante este tempo de estudos no Mestrado. Momentos de esforço e ausência que foram compensados pelo sucesso de mais esta conquista ao teu lado. Te amo.

Aos colegas do LUPS (Laboratory of Ubiquitous and Parallel Systems), Vilnei Neves, Rodrigo Duarte, Patrícia Davet, Giovane Torres, Michael Costa, Fernando Angelin e Jerônimo Ramos pelo companheirismo e amizade desprendidos neste período de convivência.

Agradeço o apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil, pelo suporte aportado ao projeto LEAPAD, no qual esta dissertação se insere.

Finalmente, agradeço a todos aqueles que não foram citados, mas que de alguma forma também foram muito importantes para concretização deste trabalho.

Para tudo há um tempo,
para cada coisa há um momento debaixo dos céus:
tempo para nascer, e tempo para morrer;
tempo para plantar, e tempo para arrancar o que foi plantado.
— ECL 3, 1-2

RESUMO

SANTOS, Maicon Ança dos. **Modelo de Escalonamento Aplicativo para Bag of Tasks em Ambientes de Nuvem Computacional**. 2017. 84 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2017.

Nuvens computacionais são uma realidade nos ambientes de pesquisa, desenvolvimento e mercado. Com isso, surgem novos desafios relacionados ao escalonamento dos recursos disponíveis por parte dos administradores dos provedores de serviço. Considerando as nuvens computacionais como infraestrutura de suporte à execução de tarefas com alto custo computacional, o contexto de desenvolvimento deste trabalho está inserido em uma realidade onde aplicações com grande demanda de processamento são submetidas a ambientes distribuídos. A opção se deu por tratar aplicações do tipo *Bag of Tasks* (BoT) em uma infraestrutura de nuvem, suportada por uma plataforma de código aberto. Aplicações BoT são representativas dentre as aplicações que mais demandam alto poder de processamento, logo, a definição de uma estratégia de escalonamento adequada para tais aplicações se faz necessária para um melhor uso dos recursos de hardware. O modelo de estratégia de escalonamento aplicativo proposto neste estudo trata da consolidação de tarefas e leva em consideração unicamente os atributos empregados na descrição de aplicações BoT, como data de chegada do *job*, duração, número de tarefas, quantidade de processamento requerida e o número de processadores disponíveis. Nesta estratégia, destaca-se o algoritmo de escalonamento aplicativo, responsável pela consolidação das tarefas das aplicações *Bag of Tasks*, principal contribuição deste trabalho. Para avaliação de funcionamento do modelo de escalonamento proposto, foram desenvolvidos estudos de caso com a submissão de tarefas de processamento, a partir de aplicações BoT, executados sobre uma infraestrutura real gerenciada com OpenStack. Os experimentos mostraram que a distribuição de carga gerada por tarefas de BoTs sobre diferentes números de máquinas virtuais tem impacto no tempo total de execução.

Palavras-Chave: nuvem computacional; bag of tasks; escalonamento; consolidação de tarefas

ABSTRACT

SANTOS, Maicon Ança dos. **Scheduling Model Application for Bag of Tasks in Cloud Computing Environments**. 2017. 84 f. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2017.

Cloud computing is a reality in research, development and business environments. Thus, new chalenges related to scheduling of available resources by the managers of cloud service providers. Considering cloud computing as an infrastructure for execution of tasks with high computational cost, the development context of this work is part of a reality where applications with large processing demands are subjected to distributed environments. It was decided to use Bag of Tasks applications (BoT) to run in a cloud infrastructure, supported by an open-source platform. Since BoT are one of the most demanding type of applications, an adequate scheduling strategy is necessary for a better usage of hardware resources. The scheduling strategy model proposed in this study deals is task consolidation and takes into account only the attributes used to describe BoT applications, like arrival time of the job, duration, number of tasks, amount of processing required and number of CPUs available. In this strategy, it stands out the scheduling algorithm in application level, responsible to consolidate BoT application tasks, the main contribution of this work. For the evaluation of the proposed scheduling model, it was developed different study cases with submission of processing tasks, from BoT applications, executed in a real cloud infrastructure with OpenStack. The experiments showed that the distribution of load generated by BoT taks in different number of virtual machines have an impact in total execution time.

Key-words: cloud coputing; bag of tasks; scheduling; task consolidation

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
BoT	Bag of Tasks
CPU	Central Processing Unit
IaaS	Infrastructure as a Service
MV	Máquina Virtual
NIST	National Institute of Standards and Technology
PaaS	Platform as a Service
SaaS	Software as a Service

SUMÁRIO

1 INTRODUÇÃO

A computação em nuvem é um componente essencial nas novas arquiteturas de sistemas computacionais, se apresentando como a infraestrutura para acesso a recursos para uma nova geração de aplicações distribuídas e ubíquas (MOHAMMADI; JAMALI; BEKRAVI, 2014). O modelo de computação em nuvem busca fazer um melhor uso dos recursos distribuídos, colocando-os juntos, a fim de alcançar um maior rendimento e ser capaz de resolver problemas de computação de grande escala (RIMAL; CHOI; LUMB, 2009).

Nuvens computacionais são hoje tanto uma realidade em ambientes de pesquisa e desenvolvimento como de mercado. Uma nuvem típica é composta por um conjunto de recursos de processamento acessíveis via rede e os usuários finais fazem uso destes recursos, visando suprir as necessidades para execução de suas aplicações. É comum que a infraestrutura física de uma nuvem seja inferior ao somatório das necessidades de todos os seus usuários, bem como, é comum que nem todos os recursos disponibilizados nesta mesma nuvem estejam sendo utilizados a pleno, durante 100% do tempo de sua operação. Assim, o usual é que as nuvens computacionais apresentem seus recursos de processamento aos usuários, de forma virtualizada.

A computação em nuvem tem adotado a virtualização de hardware como ponto chave para o uso eficiente dos recursos computacionais e as estruturas em nuvem podem ser especializadas para oferecer diversos tipos de serviços. No que diz respeito a este trabalho, é dado enfoque à organização do tipo IaaS (*Infrastructure as a Service*).

Considerando uma nuvem como suporte à computação de tarefas, a virtualização se dá por meio de máquinas virtuais. Usuários dispõem de máquinas virtuais sobre as quais lançam suas aplicações. As máquinas virtuais, por sua vez, são executadas sobre os nós de processamento físicos disponibilizados na nuvem. Um mecanismo para realizar o mapeamento das máquinas virtuais sobre os recursos reais de processamento se faz então necessário. Tal mecanismo é implementado por uma estratégia de escalonamento.

O escalonamento de tarefas em ambientes distribuídos pode ser entendido como

um mecanismo responsável pelo gerenciamento do uso de um conjunto de recursos limitados para atendimento das demandas de consumo (CASAVANT; KUHL, 1988). No caso de um escalonador de tarefas para uma nuvem, trata-se de uma ferramenta para gerenciar o uso dos recursos de hardware disponibilizados para execução de tarefas que caracterizam as demandas de uma, ou de várias, aplicações. Embora a premissa principal do mecanismo de escalonamento seja a de permitir que as tarefas da aplicação executem corretamente, em tempo finito, sem que o mesmo recurso seja alocado a duas ou mais tarefas, de forma sobreposta, é comum agregar ao escalonador uma política que racionalize o uso dos recursos de forma a apresentar um ganho de desempenho. Entende-se então que a responsabilidade de um escalonador de tarefas em uma nuvem é o de agendar recursos para que as máquinas virtuais, que representam as demandas de aplicações de usuários, possam evoluir.

Este trabalho aborda a questão de escalonamento de tarefas, representadas por máquinas virtuais, sobre um ambiente de nuvem. Como caso de estudo são consideradas aplicações do tipo *Bag of Tasks* (BoT). O cenário considerado é aquele em que um usuário requisita à nuvem recursos para processar máquinas virtuais descrevendo a demanda de sua(s) aplicação(ções) em função do tempo. A nuvem, ela própria, possui limitações de recursos físicos para atender as demandas a ela submetidas. O escalonamento é visto em dois níveis (FEITELSON, 1994): aplicativo e sistema.

O escalonamento aplicativo tem como entrada a especificação de uma aplicação BoT do usuário e o número de máquinas virtuais que suportarão a execução desta aplicação. Considera-se máquinas virtuais idênticas e a aplicação é descrita em termos de número de tarefas, tempo de chegada de cada tarefa e seus respectivos custos computacionais. A saída deste escalonamento é a distribuição, no tempo, da carga computacional associada a cada uma das máquinas virtuais disponíveis.

O escalonamento em nível de sistema se dá pelo mapeamento da execução das máquinas virtuais sobre os recursos físicos (nós) que compõem uma nuvem. O resultado é a efetiva execução das máquinas virtuais, conforme a demanda exposta, mensurada em termos de tempo total de execução.

1.1 Contribuições

A principal contribuição deste trabalho está na apresentação de uma estratégia para escalonamento de tarefas geradas por aplicações do tipo *Bag of Tasks* em nível aplicativo, acompanhada da análise das relações deste escalonamento com o realizado em nível sistema. A principal operação realizada pelo escalonamento proposto é chamada de *Consolidação de Tarefas*, nas quais, segundo diferentes políticas, as tarefas geradas por uma aplicação são reagrupadas e atribuídas às máquinas virtuais de forma a identificar a carga computacional de cada uma destas máquinas virtuais

em função do tempo.

Outras contribuições deste trabalho são:

- Um modelo para descrição de aplicações BoT;
- Instalação e operacionalização de uma plataforma de nuvem.

1.2 Contexto de Desenvolvimento

O contexto de desenvolvimento deste trabalho está inserido em uma realidade onde aplicações com grande demanda computacional são submetidas a processamento em ambiente de nuvem computacional. A opção se deu por tratar aplicações do tipo *Bag of Tasks* em uma infraestrutura de nuvem suportada por uma plataforma de código aberto.

Plataformas de computação em nuvem de código aberto vêm ganhando espaço dentre os provedores de serviço por reduzir drasticamente os custos operacionais de implantação e desenvolvimento de soluções (HU; YU, 2013). O OpenStack (OPENSTACK, 2015) é uma destas plataformas e tem se destacado por reduzir o tempo de execução de múltiplas máquinas virtuais em *hosts* físicos, proporcionando uma gestão mais eficiente, flexível e dinâmica dos recursos.

Em relação às aplicações BoT, destaca-se que elas representam um conjunto significativo, em termos de número, dentre as aplicações que demandam alto custo computacional (IOSUP; EPEMA, 2011). Direcionar o trabalho para tratar questões específicas desta classe de aplicações, portanto, teve como objetivo aumentar o horizonte de aplicação dos resultados obtidos.

Como infraestrutura de nuvem, o trabalho contou com um *cluster* composto de 10 nós, utilizado de forma exclusiva durante os experimentos. As aplicações submetidas a esta nuvem foram sintéticas, geradas conforme o modelo desenvolvido para este fim. Demais características do cluster utilizado, bem como das aplicações sintéticas construídas, são apresentadas no Capítulo 6.

1.3 Organização do Texto

O texto desta Dissertação está organizado como segue.

O Capítulo 2 traz um apanhado geral sobre os conceitos de aplicações *Bag of Tasks*, computação em nuvem e escalonamento de tarefas. São discutidos aspectos gerais de escalonamento e aspectos direcionados para ambientes distribuídos, com suas características e desafios.

No Capítulo 3 são apresentados trabalhos relacionados que buscam justificar a abordagem de problemas de escalonamento de aplicações BoT em sistemas de grades computacionais.

O Capítulo 4 apresenta o conjunto de módulos do OpenStack e suas características.

No Capítulo 5 é apresentada a principal contribuição do trabalho, caracterizado pelo escalonamento em nível aplicativo de aplicações BoT. O capítulo inicia descrevendo o modelo de aplicação BoT desenvolvido e então apresenta o processo de consolidação de tarefas proposto. Também é caracterizado, em termos operacionais, o escalonamento em nível sistema e o relacionamento entre os dois níveis.

Resultados experimentais são apresentados no Capítulo 6. São conduzidos experimentos caracterizando cenários nos quais aplicações BoT são submetidas à execução no ambiente de nuvem computacional disponível para este trabalho.

E, por fim, o Capítulo 7 apresenta as considerações finais sobre o texto e projeta trabalhos futuros.

2 CONCEITOS RELACIONADOS

Uma nuvem computacional oferece uma abstração para acessos a recursos computacionais virtualmente ilimitados. Neste trabalho, tal infraestrutura é explorada para suportar a execução de aplicações do tipo *Bag of Tasks* (BoT), havendo o problema do escalonamento destas tarefas geradas por tais aplicações nos recursos de processamento disponíveis.

2.1 Aplicações Bag of Tasks

De acordo com (CIRNE; BRASILEIRO; SAUVÉ, 2003), *Bag of Tasks* (BoT) são aplicações paralelas cujas tarefas são independentes entre si, não exigindo, portanto, cuidados de sincronização mútua em tempo de execução. A característica de independência das aplicações se apresenta em uma vasta gama de cenários, tais como mineração de dados, simulações, buscas massivas e muitas outras aplicações científicas com alta demanda de processamento. Outro ponto de destaque se deve ao fato da independência entre as tarefas tornar possível que aplicações BoT sejam executadas com sucesso em ambientes altamente distribuídos, como nuvens computacionais. Em cenários de processamento distribuído, o cálculo é realizado em computadores (nós) interligados em rede, devendo a carga de processamento ser distribuído entre estes.

Na estratégia de processamento das aplicações BoT, as tarefas geradas são colocadas em “sacolas" (ou *bags*). Um nó da rede, denominado servidor de tarefas é responsável pela distribuição do trabalho do *bag* entre nós processadores. Cada processador, ao receber uma tarefa, tem a responsabilidade de executá-la, sendo permitido que uma tarefa em execução crie novas tarefas e as coloque no *bag*.

A arquitetura geral de uma abordagem BoT, composta de um servidor de tarefas e nós processadores é exemplificada na Figura 1.

O nó servidor gerencia a sacola de tarefas e implementa uma política de distribuição das mesmas entre os processadores disponíveis. Em geral, também é atribuição do servidor integrar os vários resultados computados pelas tarefas individuais na solução final.

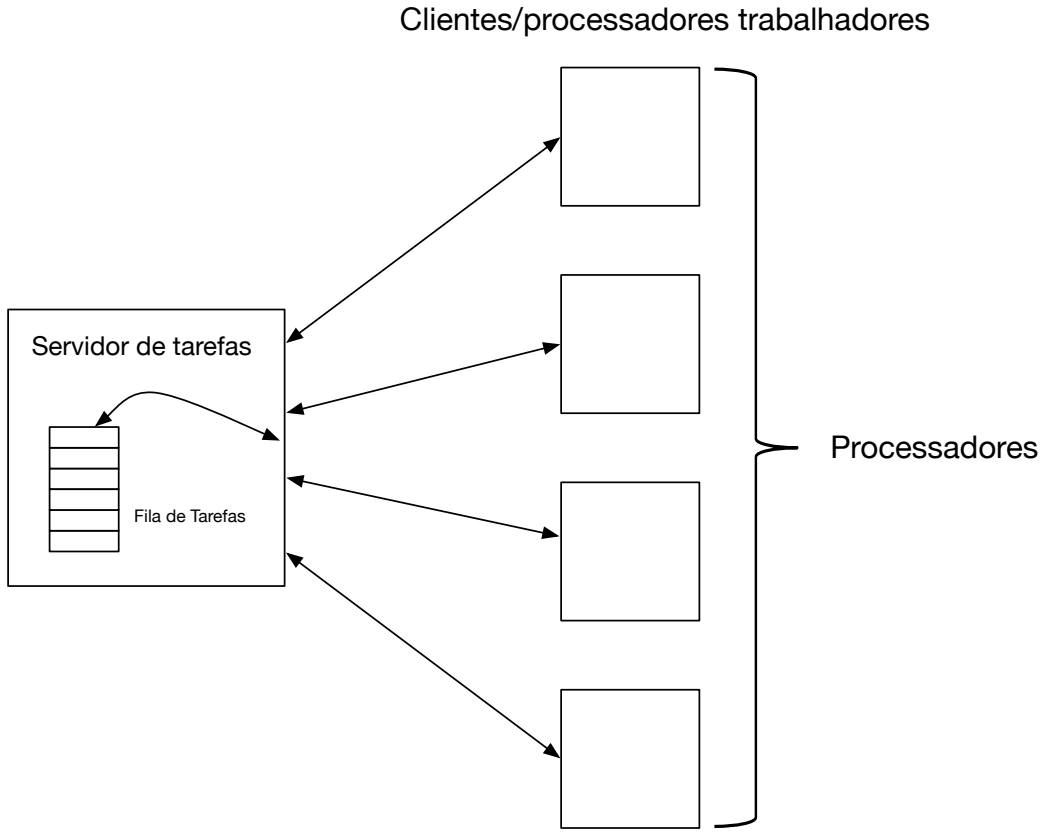


Figura 1 – Modelo de controle centralizado para *Bag of Tasks*.

2.2 Modelo de Escalonamento para Bags of Tasks

O escalonamento de aplicações BoT pode ser apresentado por meio de quatro componentes ((IOSUP et al., 2008)): ***1***: modelo de sistema e *jobs*, ***2***: arquitetura para gerenciamento de recursos, ***3***: políticas para seleção de tarefas e ***4***: políticas para escalonamento de tarefas.

2.2.1 Modelo de sistema e *jobs*

Em modelos de sistemas distribuídos de computação, os recursos computacionais representados por processadores são agrupados em aglomerados de computadores (ou *clusters*). Os processadores podem ter um desempenho diferente em função dos agrupamentos aos quais são submetidos. Os clusters podem ser, portanto, heterogêneos em relação aos seus próprios recursos e entre si.

A carga de trabalho do sistema consiste em trabalhos enviados por vários usuários e cada um destes trabalhos constitui uma sacola de tarefas. Após a chegada dos trabalhos no sistema, eles são colocados em fila à espera de recursos disponíveis para a execução. Uma vez iniciado o processamento, as tarefas são executadas até a conclusão, de modo que não são consideradas preempção de tarefas ou migração durante a execução.

2.2.2 Arquitetura para gerenciamento de recursos

Clusters podem operar como um sistema de computação distribuída de larga escala utilizando uma arquitetura para gerenciamento de recursos que dita como os trabalhos devem ser distribuídos ao longo dos recursos do sistema. Neste modelo, cada cluster tem o seu gerenciador de recursos local, mas outros gerenciadores podem ser utilizados com a intenção de construir uma arquitetura completa.

Cada gerenciador de recursos do sistema executa o mesmo procedimento de escalonamento na chegada de novas aplicações BoT, na conclusão das tarefas e, também, periodicamente, da seguinte maneira: primeiro, o gerenciador de recursos chama a política de seleção de tarefas que seleciona o conjunto elegível de tarefas a partir da fila atual do gerenciador de recursos. Em seguida, o gerenciador de recursos executa o conjunto selecionado, utilizando a política de escalonamento de tarefas, o qual ordena o conjunto e classifica os recursos para criar o escalonamento. Somente depois que todas as tarefas do escalonamento estiverem concluídas, novos conjuntos são gerados.

As arquiteturas de gerenciamento de recursos utilizadas se baseiam em clusters independentes, centralizados e descentralizados, conforme descrito na sequência.

1. **Separated clusters *(sep-c)***: Cada cluster opera separadamente com seu próprio gerenciador de recursos local e sua própria fila local para chegada de trabalhos. Os usuários podem submeter tarefas para somente um gerenciador.

2. **Escalonador centralizado com monitoramento de processador *(csp)*:** Cada cluster opera separadamente com seu próprio gerenciador de recursos local e sua própria fila local para chegada de trabalhos. Além disso, um gerenciador de recursos global, com uma fila global, opera no topo do cluster de gerenciadores de recursos. Os usuários submetem tarefas somente para a fila global e, no momento em que o gerenciador global observa recursos ociosos em um cluster, algumas tarefas da fila global são movidas para o sistema com recursos disponíveis. Informações sobre processadores livres são recuperadas por um serviço de monitoramento.

3. **Condor-like com reunião de capacidades *(fcondor)*:** é modelada uma arquitetura *Condor-like*[1] com reunião de capacidades. Com semelhança ao modelo *sep-c*, cada cluster opera separadamente com seu próprio gerenciador de recursos local e sua própria fila local para chegada de trabalhos. Porém, neste cenário, cada usuário pode enviar tarefas para qualquer gerenciador de recursos.

[1] Arquitetura que compartilha as características do software HTCondor, especializado para gerenciamento de cargas de trabalho para tarefas de computação intensiva. Disponível em: `http://research.cs.wisc.edu/htcondor/description.html`.

O usuário mantém a submissão de tarefas ao mesmo gerenciador enquanto as tarefas são iniciadas imediatamente. No momento em que as tarefas começam a ser enfileiradas, o usuário mudará para outro gerenciador de recursos, em uma ordem *round-robin*.

2.2.3 Políticas para seleção de tarefas

As políticas para seleção de tarefas determinam os conjuntos elegíveis de tarefas para execução a partir das filas dos gerenciadores de recursos locais (IOSUP et al., 2008). A seguir, são apresentadas as características destas políticas.

1. **S-T:** A política *Select-Tasks* seleciona todas as tarefas do sistema. Para cada BoT que chega, o conjunto de suas tarefas é adicionado ao conjunto de elegíveis.

2. **S-BoT:** A política *Select-BoTs* seleciona BoTs na ordem em que chegam. Quando o conjunto de tarefas elegíveis se torna vazio, o conjunto de tarefas da BoT selecionada passa a ser considerado o novo conjunto elegível.

3. **S-U-Prio:** A política *Select-User-Priority* assume que cada usuário do sistema possui uma prioridade única. Esta política seleciona todas as tarefas do usuário com a maior prioridade. Idealmente, cada usuário tem uma prioridade única, que distingue os seus direitos de uso de recursos dos outros usuários. Na prática, um sistema será configurado com apenas algumas prioridades distintas.

4. **S-U-T:** A política *Select-User-Tasks* visa o igual compartilhamento de recursos entre os usuários do sistema. Primeiro, seleciona o usuário com menor consumo de recursos e, logo após, seleciona todas as tarefas do usuário escolhido.

5. **S-U-BoT:** Semelhante à política S-U-T, a *Select-User-BoT* visa o igual compartilhamento de recursos entre os usuários do sistema. Também seleciona o usuário com menor consumo de recursos e, então, são ordenadas as BoTs dos usuários, respeitando sua ordem de chegada, dentro de um conjunto ordenado. A partir deste conjunto, as tarefas da primeira BoT são consideradas o novo conjunto elegível.

6. **S-U-GRR:** A política *Select-User-Global-Round-Robin* seleciona o próximo usuário com base no algoritmo round-robin. Todas as tarefas pendentes para o usuário selecionado são adicionadas ao conjunto elegível. Se um usuário submeter uma BoT adicional durante este processo, as tarefas correspondentes não serão consideradas para seleção.

7. **S-U-RR:** A política *Select-User-Round-Robin* é uma variação de S-U-GRR, na qual somente uma tarefa por usuário é selecionada em cada rodada. De acordo

com a política S-U-RR, um BoT de tamanho *N* receberá um serviço completo após, exatamente, *N* rodadas.

2.2.4 Políticas para escalonamento de tarefas

Existem muitas políticas para escalonamento de aplicações BoT em sistemas distribuídos de computação. Nesta seção serão apresentadas políticas que levam em consideração o desempenho de processador e o tempo de execução de tarefas.

1. **ECT:** A política *Earliest Completion Time* (Primeiro Tempo de Conclusão) atribui cada tarefa para o recurso (cluster ou processador) que leva ao primeiro tempo de conclusão possível. Se o recurso é um cluster, a política também leva em consideração a fila de tarefas para computar o tempo mais próximo da conclusão. Um exemplo de política semelhante inclui o algoritmo dinâmico para escalonamento de tarefas independentes MinMin.

2. **FPLT:** A política *Fastest Processor Largest Task* (Processador mais Rápido Tarefa Maior) atribui a tarefa maior para o processador mais rápido disponível.

3. **RR:** A política *Round-robin Replication* (Replicação Round-robin) primeiro, atribui todas as tarefas aos processadores, na ordem inicial do conjunto elegível. Após o término de todas as tarefas do conjunto, estas são replicadas no máximo uma vez sobre os recursos que se tornam disponíveis, de acordo com o escalonamento round-robin.

4. **WQR:** A política *Work Queue with Replication* (Fila de Trabalho com Replicação) difere da política de replicação round-robin pelo fato de poder replicar tarefas várias vezes.

5. **DFPLT:** A política *Dynamic Fastest Processor Largest Task* (Processador mais Rápido Tarefa Maior - Dinâmico) parte do princípio que o desempenho dos recursos é dinâmico ao longo do tempo. Após a conclusão de uma tarefa, o desempenho do recurso no qual a tarefa foi executada é recalculado e o recurso recebe uma classificação de desempenho. A política atribui a maior tarefa para o recurso com a maior classificação de desempenho.

6. **ECT-P:** A política *Earliest Completion Time - Prediction* (Primeiro Tempo de Conclusão com predição de tempo de execução de tarefas) opera de forma semelhante à política de primeiro tempo de execução, porém, usa predição em vez de valores reais de tempo de execução de tarefas.

7. **STFR:** A política *Shortest Task First with Replication* (Menor Tarefa Primeiro com Replicação) sempre atribui a menor tarefa primeiro. Depois de terminar a execução de todas as tarefas do conjunto elegível, replica no máximo uma vez sobre

os recursos que se tornam disponíveis, de acordo com o escalonamento round-robin.

8. **FPF:** A política *Fastest Processor First* (Processador Mais Rápido Primeiro) atribui as tarefas na ordem inicial do conjunto elegível. E cada tarefa é atribuída para o processador mais rápido disponível.

2.3 Computação em Nuvem

O NIST (*National Institute of Standards and Technology*) (MELL; GRANCE, 2011) define computação em nuvem como um modelo que permite o acesso de rede ubíquo e sob demanda a um conjunto de recursos computacionais compartilhados e configuráveis, por exemplo, servidores, armazenamento, aplicações e serviços. Esses recursos podem ser rapidamente provisionados e liberados com um esforço mínimo de gerenciamento por parte do provedor de serviços em nuvem. As características gerais de uma nuvem computacional (Figura 2) são:

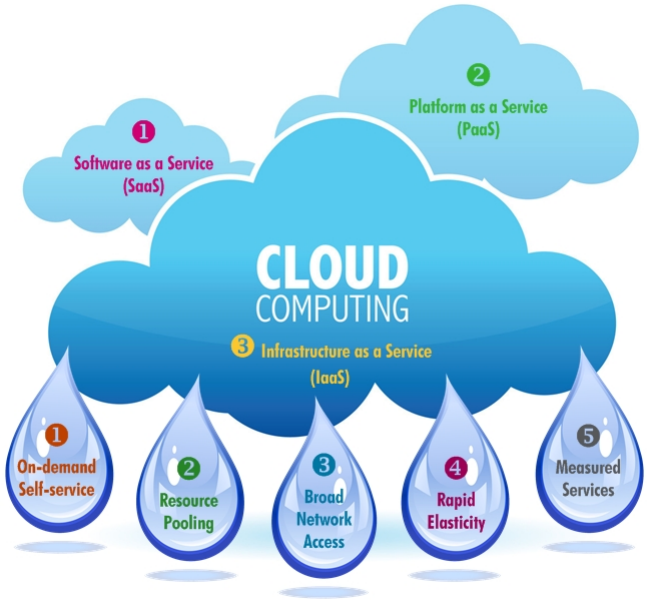


Figura 2 – Computação em nuvem (CALVEY, 2013).

1. **Auto-serviço sob demanda**: um consumidor pode solicitar provisionamento de recursos, unilateralmente, sem necessidade de interação com o provedor.

2. ***Pool* de recursos**: nesta característica, os recursos de computação do provedor são agrupados para servir a vários consumidores usando um modelo *multi-tenant*[2], com diferentes recursos físicos e virtuais atribuídos e realocados dina-

[2]Modelo *multi-tenant* ou multi-inquilinos é uma arquitetura essencial para um ambiente em nuvem, pois permite que múltiplos inquilinos (empresas/clientes) compartilhem os mesmos recursos físicos como um aplicativo ERP, por exemplo, mas permaneçam logicamente isolados.

micamente de acordo com a demanda do consumidor.

3. **Amplo acesso à rede**: os recursos estão disponíveis através da rede e podem ser acessados por meio de mecanismos padronizados que promovem o uso de plataformas heterogêneas pelos clientes, por exemplo, celulares, *tablets*, notebooks e estações de trabalho.

4. **Elasticidade rápida**: as capacidades podem ser elasticamente provisionadas e liberadas, em alguns casos, automaticamente, com o objetivo de escalar o sistema de forma compatível com a demanda. Para o consumidor, as capacidades disponíveis para provisionamento parecem ser ilimitadas e podem ser apropriadas em qualquer quantidade a qualquer momento.

5. **Serviço mensurado**: sistemas em nuvem automaticamente controlam e otimizam o uso dos recursos, que podem ser monitorados, controlados e reportados, oferecendo transparência tanto para o provedor quanto para o consumidor do serviço utilizado.

2.3.1 Modelos de serviço

Na computação em nuvem, os serviços são divididos em classes de acordo com o nível de abstração da capacidade provida e do modelo de serviços do provedor, no qual os recursos são disponibilizados aos usuários por meio de, basicamente, três modalidades que são apresentados a seguir:

- **SaaS (*Software as a Service*)**: neste modelo, os provedores instalam e operam os softwares na nuvem e os clientes tem acesso liberado a tais aplicações. Os usuários não gerenciam a infraestrutura nem a plataforma na qual os aplicativos estão instalados.

- **PaaS (*Platform as a Service*)**: modelo que oferece uma plataforma computacional, tipicamente composta por sistema operacional, ambiente de execução de linguagens de programação, base de dados e servidor *web*. O cliente não gerencia nem controla a infraestrutura subjacente, incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem controle sobre os aplicativos implementados e, possivelmente, configurações para o ambiente de hospedagem de aplicativos.

- **IaaS (*Infrastructure as a Service*)**: modelo mais básico de serviços em nuvem, no qual os provedores disponibilizam computadores (físicos ou, mais frequentemente, virtuais) e demais recursos para os usuários. *Pools* de recursos são montados para suportar as escalabilidades dos serviços que podem variar para cima ou para baixo, de acordo com as necessidades dos clientes.

2.3.2 Modelos de implementação

Além dos modelos citados anteriormente, serviços de computação em nuvem são diferenciados de acordo com os seus modelos de implantação. Nestes modelos, a diferenciação se dá a partir das necessidades das aplicações que serão implementadas. Sendo assim, a restrição ou abertura de acesso depende do processo de negócios, do tipo de informação e do nível de visão desejado.

Conforme (MELL; GRANCE, 2011), a seguir são apresentados os principais modelos de implantação dos serviços de nuvem.

- **Nuvem privada**: é uma infraestrutura de nuvem operando exclusivamente para um única organização, gerenciada interna ou externamente. Sua implantação pode melhorar significativamente os negócios de uma empresa, mas requer uma reavaliação das decisões sobre os recursos existentes.

- **Nuvem comunitária**: este modelo envolve o compartilhamento da infraestrutura computacional de organizações de uma mesma comunidade. Exemplo: todos os órgãos governamentais de um determinado estado podem compartilhar a infraestrutura de nuvem para gerenciar dados relativos aos cidadãos residentes naquele estado.

- **Nuvem pública**: os serviços são providos aos usuários através de uma rede aberta para uso público. Modelo muito próximo ao das nuvens privadas, divergindo em algumas questões relacionadas a segurança, visto que seu acesso somente é liberado via internet, sem conexões diretas.

- **Nuvem híbrida**: nuvens híbridas são uma composição de um ou mais modelos de implementação (privadas, públicas ou comunitárias), que se comportam como entidades distintas mas permanecem unidas, oferecendo os benefícios dos vários modelos.

2.4 Escalonamento de Recursos em Computação em Nuvem

Em ambientes computacionais, o escalonamento de recursos desempenha um papel vital. *Throughput*, latência e tempo de resposta são parâmetros importantes e o objetivo dos métodos de escalonamento em computação dispersa é distribuir a carga de trabalho para os processadores e aumentar o seu consumo, enquanto diminui o tempo total de execução do trabalho (VAITHIYANATHAN, 2013).

Tarefas e recursos precisam ser alocados e escalonados de tal maneira que os usuários da nuvem possam concluir seus trabalhos com o mínimo de tempo e custo, maximizando a satisfação do usuário e a taxa de transferência do provedor de recursos

em nuvem (SOMASUNDARAM; GOVINDARAJAN, 2014). Em sentido mais amplo, o usuário deverá concluir seus trabalhos com tempo mínimo e custo mínimo.

O escalonamento de tarefas da computação em nuvem faz referência a despachar as tarefas com o intuito de partilhar os recursos entre os diferentes usuários, de acordo com certas regras de uso do ambiente.

O processo de agendamento de tarefas em uma nuvem computacional pode ser feito em três etapas (GOEL; CHAMOLI, 2014):

- **Descoberta de recursos e filtragem**: um agente sabe o estado atual de todos os recursos que estão disponíveis na nuvem e também os demais recursos que podem ser disponibilizados. Na verdade, esses recursos são geralmente as máquinas virtuais. Ele recolhe regularmente o estado de cada recurso conectado à nuvem.

- **Seleção de recursos**: com base nas informações obtidas a partir do status de recursos, com informações a respeito de trabalhos em fila atuais e informações sobre o estado dos recursos de nuvem, o agendador de tarefas da nuvem toma decisões sobre a criação ou supressão de nós específicos na nuvem (máquinas virtuais), a fim de melhor atender o conjunto de trabalhos esperando para ser executado.

- **Submissão de trabalhos**: nesta etapa, finalmente o trabalho é submetido ao melhor recurso disponível selecionado.

Um escalonador precisa levar em consideração algumas operações que serão relevantes e auxiliarão no processo de tomada de decisão. A seguir, são apresentadas e detalhadas as quatro fases que prestam subsídios para as operações de tomada de decisão de um escalonador de processos em um ambiente de computação distribuído (WILLEBEEK-LEMAIR; REEVES, 1993).

- **Avaliação da carga do processador**: um valor de carga é estimado para cada processador do sistema. Estes valores são usados como entrada para o balanceador de carga a fim de detectar desequilíbrios e tomar decisões de migração de carga.

- **Determinação de rentabilidade do balanceamento de carga**: um fator quantifica o grau de desequilíbrio de carga dentro de um domínio de processamento. Ele é usado como uma estimativa do potencial de aumento de velocidade obtido por meio de balanceamento de carga, determinado se o escalonamento é rentável ou não naquele momento.

- **Estratégia de migração de tarefas**: origens e destinos para migração de tarefas são determinadas. Fontes são notificadas sobre a quantidade e destino das tarefas de balanceamento de carga.
- **Estratégia de seleção de tarefas**: processadores na origem selecionam as tarefas mais adequadas para uma carga eficiente e as enviam para os destinos apropriados.

As principais métricas de desempenho dos algoritmos de escalonamento adotadas são (GOEL; CHAMOLI, 2014):

- ***Makespan***: é a soma dos tempos de processamento das tarefas e representa quando todos os trabalhos do agendamento terminaram.
- **Custo de execução**: é definido como o custo total de todos os recursos utilizados na execução de um trabalho.
- **Taxa de rejeição de trabalhos**: é definido como o total de trabalhos rejeitados devido a ultrapassagem do tempo de execução em comparação ao prazo máximo ou um custo de execução maior para o número total de trabalhos apresentados.
- **Tempo de execução**: é definido como o tempo de quando a tarefa é enviada para o ambiente computacional de nuvem até sua execução final.
- **Nível de satisfação do usuário**: é definido como quão longe de dar satisfação estão os serviços do provedor, em termos de recursos como armazenamento e computação.

Em tempo, é importante destacar os tipos de escalonamento que podem ser aplicados em ambientes de nuvem (GOEL; CHAMOLI, 2014).

- **Escalonamento estático *versus* escalonamento dinâmico**: no modelo estático, toda a informação sobre o estado de todos os recursos disponíveis na nuvem, bem como todas as necessidades dos trabalhos, é mapeada para o recurso adequado. Ao passo que, no modelo dinâmico, a alocação de tarefas é feita à medida que se executa a aplicação, situação na qual não é possível determinar o tempo de execução. A vantagem do escalonamento dinâmico sobre o estático é que o sistema não precisa possuir o comportamento de tempo de execução do pedido antes de ser executado.
- **Escalonamento centralizado, descentralizado e hierárquico**: no escalonamento centralizado, há um planejador central ou vários agendadores distribuídos que têm a responsabilidade de tomar as decisões globais de agendamento.

Não há mais controle sobre os recursos: o agendador monitora continuamente o estado dos recursos disponíveis e, portanto, é mais fácil obter escalonadores eficientes. Este modelo tem como vantagem a facilidade de implementação, mas, como desvantagem, a falta de escalabilidade, tolerância a falhas e desempenho. No escalonamento descentralizado não existe uma entidade central para controlar os recursos: os escalonadores menores, conhecidos como máquinas de recursos locais, gerenciam e mantém a fila de trabalhos. Ele é menos eficiente que o escalonamento centralizado. O escalonamento hierárquico, por sua vez, apresenta a computação em três níveis. O nível superior é um meta-nível, no qual as tarefas não são agendadas diretamente, mas acontece a reconfiguração do agendador em função das características dos trabalhos que estão entrando para o processamento. No nível intermediário, também chamado de nível de grupo, o gestor de cada grupo colabora com o outro e aloca tarefas para os trabalhadores do grupo. O nível inferior, chamado de nível dentro do grupo, é o nível no qual acontece o auto-agendamento da tarefas.

- **Escalonamento preemptivo *versus* não-preemptivo**: no modelo preemptivo, a preempção é permitida, ou seja, a execução atual do trabalho pode ser interrompida e o trabalho migrado para outro recurso. Já no não-preemptivo, um trabalho deve ser totalmente concluído no recurso (o recurso não pode ser tirado da tarefa, trabalho ou aplicativo).

- **Escalonamento imediato *versus batch***: no escalonamento imediato, assim que o trabalho chega, é escalonado pois não haverá espera para o intervalo de tempo seguinte. No escalonamento por *batch*, os trabalhos são agrupados, primeiramente, em lotes e, em seguida, eles são atribuídos aos recursos por parte do escalonador.

Os ambientes de computação em nuvem podem ser escalonados horizontalmente com a adição, por exemplo, de um balanceador de carga capaz de fornecer a lógica de tempo de execução capaz de distribuir uniformemente a carga de trabalho entre os recursos disponíveis.

Este modelo de arquitetura fundamental pode ser aplicado a qualquer recurso computacional, com a distribuição de carga de trabalho comumente realizada em apoio a servidores distribuídos virtuais, dispositivos de armazenamento em nuvem e serviços em nuvem (TIDMARSH, 2013).

2.4.1 Desafios do escalonamento

Também é necessário identificar as principais questões e desafios envolvidos que podem afetar o modo como o escalonamento de recursos irá acontecer (AL NUAIMI et al., 2012). Os desafios do escalonamento podem ser resumidos como segue:

- **Distribuição espacial dos nós da nuvem**: alguns algoritmos são projetados para serem eficientes apenas para uma intranet ou sistemas próximos nos quais atrasos de comunicação sejam insignificantes. No entanto, é um desafio para a concepção de um algoritmo de balanceamento de carga poder funcionar para nós espacialmente distribuídos. Isto acontece porque outros fatores devem ser levados em consideração, tais como a velocidade das ligações de rede entre os nós, a distância entre o cliente e os nós de processamento de tarefas e as distâncias entre os nós envolvidas no fornecimento do serviço.

- **Replicação de armazenamento**: um algoritmo de replicação total não leva em conta a utilização eficiente do armazenamento. Isto ocorre porque os mesmos dados serão armazenados em todos os nós de replicação, gerando custos elevados, uma vez que é necessário mais espaço para armazenamento. No entanto, os algoritmos de replicação parcial podem salvar partes dos conjuntos de dados em cada nó (com um certo nível de sobreposição) com base em características específicas, como poder de processamento e capacidade de armazenamento.

- **Complexidade de algoritmos**: algoritmos de balanceamento de carga tendem a ser menos complexos em termos de implementação e operações. A alta complexidade de implementação conduz a um processo mais complexo que poderia causar alguns problemas de desempenho. Além disso, quando os algoritmos requerem mais informações e maior comunicação para monitoramento e controle, os atrasos causariam mais problemas e a eficiência diminuiria. Portanto, os algoritmos de balanceamento de carga devem ser projetados nas formas mais simples possíveis.

- **Ponto de falha**: o controle do balanceamento de carga e a coleta de dados sobre os diferentes nós devem ser concebidos de uma forma que se evite ter um ponto único de falha no algoritmo. Alguns algoritmos (centralizados) podem fornecer mecanismos eficientes e eficazes para resolver o balanceamento de carga em um determinado padrão. No entanto, eles têm o problema de um controlador para todo o sistema. Nesses casos, se o controlador falhar, então todo o sistema seria um fracasso. Qualquer algoritmo de balanceamento de carga deve ser concebido de modo a superar este desafio. Algoritmos de balanceamento de carga distribuída parecem fornecer uma melhor abordagem, mas eles são muito mais complexos e exigem mais coordenação e controle para funcionar corretamente.

Os algoritmos de escalonamento podem ser classificados em duas categorias: técnica de escalonamento estável e técnica de escalonamento instável. Esta dupla tem seus méritos e limitações: em comparação à técnica de escalonamento estável, a técnica instável alcança um desempenho mais eficiente (VAITHIYANATHAN, 2013).

Nos ambientes de computação em nuvem, o escalonamento de tarefas é gerido por meio da tecnologia de virtualização. Isto tem sido usado para tornar as tarefas dos usuários completamente transparentes. O agendamento de tarefas torna-se mais complexo por causa da transparência e flexibilidade dinâmica dos ambientes de computação em nuvem e pelas diferentes necessidades de recursos solicitados por diferentes aplicações.

2.4.2 Características do escalonamento em computação em nuvem

O escalonamento de tarefas em ambientes de computação em nuvem apresenta as seguintes características (SUN et al., 2013):

- **Plataforma de recursos unificada**: como os sistemas de computação em nuvem utilizam tecnologias de virtualização, os recursos físicos subjacentes podem ser abstraídos como se fossem um *pool* de recursos unificado.
- **Escalonamento centralizado global**: a computação em nuvem é um modelo que fornece recursos centralizados por meio do espelhamento para múltiplas aplicações distribuídas. E esta implementação pode tornar mais fácil a execução de procedimentos heterogêneos. Com isso, a virtualização de recursos e os serviços espelhados fazem o agendamento de tarefas conseguir um escalonamento centralizado global.
- **Cada nó da nuvem é independente**: o escalonamento interno de cada nó na nuvem é autônomo, e os escalonadores na nuvem não irão interferir na política de escalonamento desses nós.
- **Escalabilidade do agendamento de tarefas**: a escala da oferta de recursos de um provedor de serviços de nuvem pode ser limitada em estágios iniciais. Com a adição de uma variedade de recursos de computação, o tamanho dos recursos virtuais pode tornar-se grande, e a procura pode aumentar também. Em ambientes computacionais de nuvem, o agendamento de tarefas deve cumprir os requisitos de escalabilidade, de modo que a taxa de transferência do agendamento de tarefas na nuvem pode não ser muito baixo.
- **Escalonamento pode ser auto-adaptativo**: a expansão e o encolhimento de aplicações em nuvem pode ser necessário dependendo da sua natureza. Os recursos de computação virtual no sistema de nuvem também podem aumentar ou diminuir ao mesmo tempo. Estes estão em constante mudança, alguns recursos podem falhar, novos recursos podem juntar-se às nuvens.
- **O conjunto de escalonamento de tarefas**: o escalonamento de tarefas é dividido em duas partes: uma é usada como um conjunto de recursos unificados

para escalonamento e é o principal responsável pelo agendamento de aplicações e APIs (*Application Programming Interfaces*) da nuvem; a outra é utilizada para o agendamento de recursos unificados na nuvem. No entanto, cada escalonamento consiste de dois processos de duas vias: o escalonador aluga recursos na nuvem e, logo após, retorna os recursos solicitados após o uso. A combinação da estratégia de escalonamento e de retorno de recursos é o conjunto de escalonamento de tarefas.

Em seu trabalho, (ANNETTE; BANU; SHRIRAM, 2013) propõe uma taxonomia para os algoritmos de escalonamento em ambientes de computação em nuvem baseada em dependência de tarefas, tanto para tarefas dependentes quanto para independentes. A taxonomia é apresentada na Figura 3.

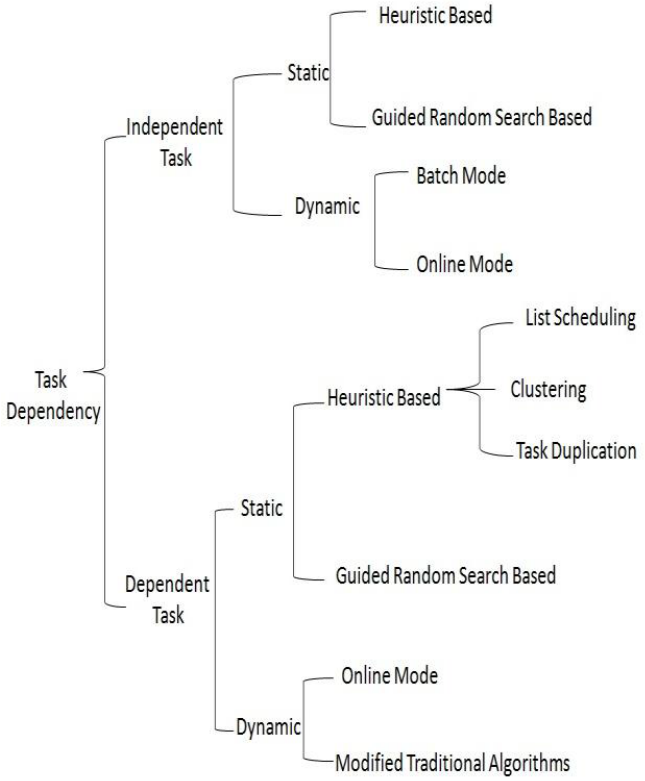


Figura 3 – Taxonomia dos algoritmos de escalonamento em nuvem baseados em dependência de tarefas (ANNETTE; BANU; SHRIRAM, 2013).

Com base na dependência, as tarefas podem ser classificadas como independentes, na qual uma tarefa não requer nenhuma comunicação entre as demais ou dependentes, que diferem das tarefas independentes pelo fato de existir uma ordem de precedência que deve ser obedecida durante o processo de escalonamento.

2.5 Considerações Sobre o Capítulo

Como citado anteriormente, o escalonamento de tarefas em ambientes de nuvem desempenha um papel importante, visto que os trabalhos submetidos e recursos precisam ser alocados de tal maneira que os usuários do sistema possam concluir suas tarefas com o mínimo de tempo e custo.

Os ambientes computacionais em nuvem sob o modelo de infraestrutura como serviço (IaaS) se destacam por possibilitar um menor tempo de execução de tarefas ao fazerem uso de tecnologias de virtualização de recursos de hardware. Sendo assim, aplicações com alta demanda de processamento, como por exemplo, processamento de alto desempenho, mineração de dados, são candidatas à execução em tais ambientes com o objetivo de obter um escalonamento de tarefas mais adequado.

Aplicações que executam em plataformas IaaS são custosas e, não raro, demandam alta quantidade de recursos por um período extenso de tempo. A exemplo disto tem-se tarefas de compilação, previsão meteorológica, prospecção de petróleo. Os usuários tem uma expectativa quanto ao atendimento de suas tarefas e pagam por isso, logo, a nuvem computacional tem o comprometimento de atender o usuário conforme a expectativa. No entanto, os recursos físicos são de fato limitados e para garantir a operação da nuvem em uma margem de custos aceitável (quantidade de equipamentos, energia, satisfação dos usuários), alguma política de alocação de recursos deve ser empregada como o escalonamento de recursos em ambientes distribuídos.

O escalonamento de tarefas em computação em nuvem é caracterizado por possuir uma plataforma de recursos unificada, escalonamento centralizado global, independência entre os nós, escalabilidade no agendamento de tarefas e auto-adaptação do escalonamento.

Em ambientes de nuvens computacionais especializadas em prover infraestrutura como serviço, os algoritmos de escalonamento podem ser classificados, em função da dependência das tarefas, como algoritmos para escalonamento de tarefas independentes, nos quais não existe nenhuma comunicação entre as tarefas em execução, ou algoritmos para escalonamento de tarefas dependentes, nos quais existe uma ordem de dependência das tarefas que deve ser obedecida durante o processo de escalonamento.

3 TRABALHOS RELACIONADOS

Aplicações do tipo *Bag of Task* são as que produzem um maior número de tarefas e que consomem mais recursos em grades computacionais (IOSUP; EPEMA, 2011), sendo desta forma, justificados os trabalhos na literatura que abordam o problema de escalonamento deste tipo de aplicação sobre grades computacionais. Como exemplos destes trabalhos, citam-se (GUTIERREZ-GARCIA; SIM, 2013), (OPRESCU; KIELMANN, 2010) e (NETTO; BUYYA, 2009).

Em (GUTIERREZ-GARCIA; SIM, 2013), o problema abordado se refere a maximizar o uso dos recursos de processamentos disponíveis para execução das tarefas geradas por um BoT em uma nuvem heterogênea. Para tanto, é proposto um conjunto de 15 heurísticas de escalonamento executando duas operações básicas: ordenamento das tarefas e o mapeamento das tarefas sobre os recursos disponíveis. O modelo de BoT considera que as tarefas possuem atributos de tempo de chegada e custo computacional expresso em termos de número de instruções. Para avaliação foram utilizadas quatro combinações para determinar o tamanho das tarefas em cada BoT. Os resultados deste trabalho foram obtidos por simulação e indicaram sucesso na distribuição de carga realizada.

As questões de heterogeneidade de processadores, associadas ao custo monetário de cada recurso, são objetos de estudo em (OPRESCU; KIELMANN, 2010). A proposta de escalonamento visa obter o menor tempo de execução para uma aplicação considerando o limite do orçamento apresentado pelo proprietário do BoT. O escalonamento é realizado em tempo de execução, considerando o custo computacional das tarefas por meio de *profilling* de execução. A avaliação dos resultados se deu por prototipação em um ambiente de nuvem desenvolvido no próprio grupo.

Também considerando um ambiente heterogêneo é proposta a estratégia de escalonamento em (NETTO; BUYYA, 2009). A abordagem, no entanto, considera um ambiente em que o provedor não disponibiliza informações sobre a carga de processamento disponível nem sobre o custo energético dos recursos mas que negocia, com o usuário, uma data limite para término do processamento. O escalonamento é dinâmico e, em função da carga assumida pelo provedor, é definida sua capacidade para

receber novos BoTs considerando sua carga atual. A avaliação de desempenho foi realizada em um ambiente composto de múltiplos clusters de larga escala, assumindo que 50% da carga submetida determinava a execução em um cluster específico e os outros 50% podendo ser executados em qualquer, ou quaisquer, clusters.

Como ferramenta de apoio à análise de tais estratégias, em particular quando da definição de um modelo de carga para BoTs utilizados na avaliação de desempenho das propostas, o trabalho de A. Iosup e seu grupo (IOSUP; EPEMA, 2011; IOSUP et al., 2008) contribui apresentando como modelar aplicações BoT submetidas a grades e nuvens computacionais. Estas características foram elencadas após o estudo de um grande número de traços de execução sobre ambientes reais (IOSUP et al., 2008). Estas características são apresentadas a seguir, bem como a distribuição que melhor representa as amostras observadas.

- Probabilidade de um BoT ser submetido por um determinado usuário, modelada pela distribuição Zipf.
- Tempo decorrido entre a chegada de dois BoTs, modelada pela distribuição Weibull.
- Número de tarefas em um BoT submetido, modelada pela distribuição Weibull.
- Carga computacional das tarefas de um BoT, modelada pela distribuição Weibull.
- Tempo de processamento requerido pelas tarefas de um BoT, modelada pela distribuição Normal.
- Variabilidade dos tempos de execução das tarefas de um BoT, modelada pela distribuição Weibull.

No presente trabalho, a posição adotada difere das estratégias de escalonamento apresentadas por considerar o escalonamento em nível aplicativo. Na abordagem proposta, é considerada a variação da carga de processamento de uma aplicação em função do tempo, sendo produzido, como saída, o consumo de CPU de máquinas virtuais que serão, em uma etapa posterior, submetidas ao escalonamento sistema em uma nuvem computacional. A hipótese considerada é que o tempo total de execução pode ser reduzido agrupando tarefas em uma etapa prévia à execução efetiva (GOKILAVANI; SELVI; UDHAYAKUMAR, 2013).

Por outro lado, o modelo de aplicação BoT apresentado em (IOSUP et al., 2008), bem como os casos de estudo desenvolvidos nas análises das estratégias de escalonamento da literatura, são importantes no que diz respeito à concepção de aplicações sintéticas para validação da presente proposta de escalonamento.

4 OPENSTACK E SUA ARQUITETURA

OpenStack é uma tecnologia de computação em nuvem que produz uma plataforma ubíqua open source para nuvens públicas e privadas (HU; YU, 2013). A plataforma é composta por vários componentes de software desenvolvidos de forma independente, organizados em um subprojeto separado para cada serviço e oferece uma ampla gama de funcionalidades necessárias para construir uma nuvem do tipo IaaS.

Trata-se de um conjunto de softwares que controla grandes *pools* de computação, armazenamento e recursos de rede ao longo de um *datacenter*, no qual tudo é gerenciado por meio de um painel que permite aos administradores controlar e capacitar seus usuários no provisionamento de recursos. O projeto do OpenStack está apoiado nos princípios de design aberto, desenvolvimento aberto e comunidade aberta, conforme é indicado em (OPENSTACK, 2015).

Na Figura 4 tem-se a ilustração de todos os componentes que fazem parte da arquitetura conceitual do OpenStack e suas interconexões disponíveis na versão utilizada para os estudos deste trabalho chamada de Kilo. (OPENSTACK, 2016).

O OpenStack é composto pelos serviços descritos abaixo:

- ***Horizon***: componente responsável pelo serviço OpenStack Dashboard. Provê aos administradores e usuários uma interface gráfica de acesso, fornecimento e automatização de recursos baseados na nuvem. Seu design é extensível, ou seja, pode-se facilmente ligar-se aos outros componentes, como por exemplo, monitoramento, cobrança e ferramentas de gestão adicionais.

- ***Nova***: componente responsável pelo serviço OpenStack Compute. Fornece recursos de computação sob demanda pelo provisionamento e gerenciamento de máquinas virtuais usando algum monitor de máquina virtual.

- ***Neutron***: componente responsável pelo serviço OpenStack Networking. Fornece um sistema de conexão automática, escalável e API orientada para a gestão de rede e endereços IP.

- ***Cinder***: componente responsável pelo serviço OpenStack Block Storage. Fornece a capacidade de realizar o armazenamento em bloco com mapeamento

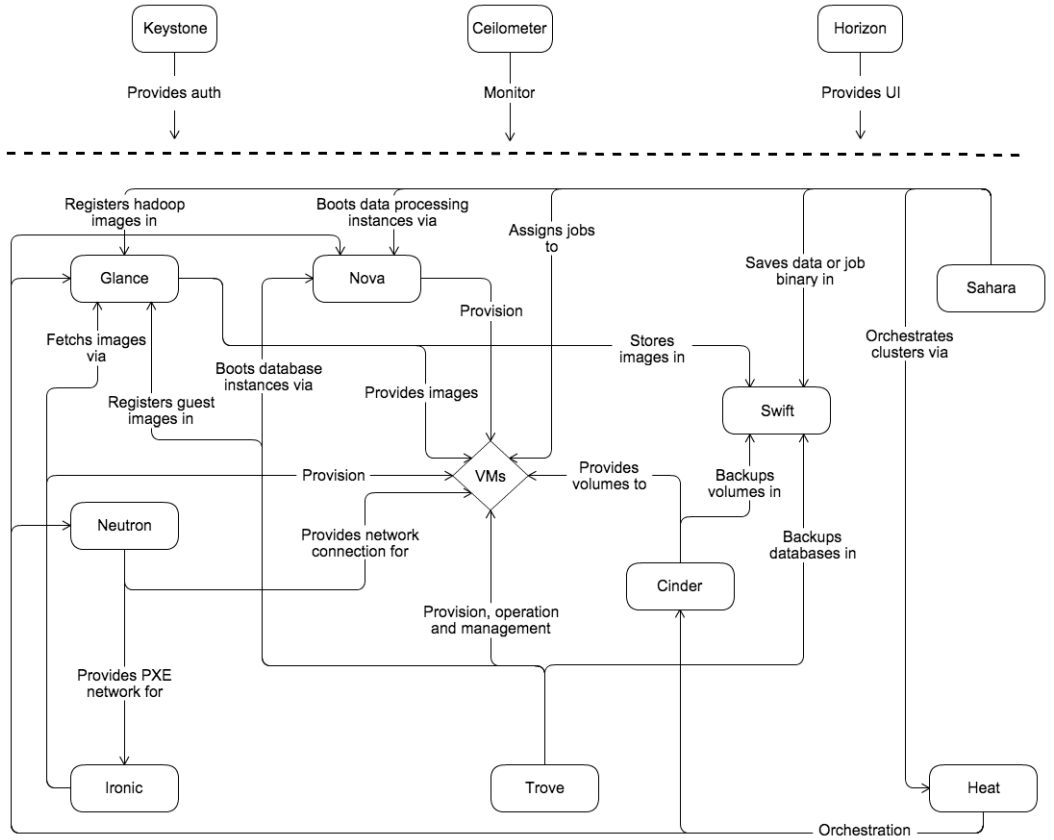


Figura 4 – Arquitetura conceitual do OpenStack (OPENSTACK, 2016).

persistente para instâncias de computação com suporte para uma variedade de soluções de armazenamento.

- ***Glance***: componente responsável pelo serviço OpenStack Image Service. Fornece um registro de cópias de disco de boot e um serviço para armazenamento dessas imagens de sistema.

- ***Swift***: componente responsável pelo serviço OpenStack Object Storage. Fornece o crescimento distribuído dos dados, armazenamento acessível pelas APIs que pode ser integrado diretamente em aplicações ou utilizados para backup, arquivamento e retenção de dados.

- ***Keystone***: componente responsável pelo serviço OpenStack Identity. Fornece um diretório central de usuários mapeado para os serviços OpenStack. Atua como um sistema de autenticação comum em todo o sistema operacional em nuvem e pode integrar-se com serviços de diretório de *back-ends* existentes.

- ***Ceilometer***: componente responsável pelo serviço OpenStack Telemetry. Fornece infraestrutura comum para coletar medições de uso e desempenho dentro de uma nuvem gerida pelo OpenStack. A partir destes dados coletados, pode-se calcular, por exemplo, o quanto será cobrado pelo serviço que certo usuário

utilizou.

- ***Heat***: componente responsável pelo serviço OpenStack Orchestration. Implementa um serviço para orquestrar múltiplas aplicações em nuvens compostas usando o modelo da AWS.

Estes serviços podem ser agrupados em três grandes áreas: comunicação, armazenamento e computação conforme é ilustrado na Figura 5.

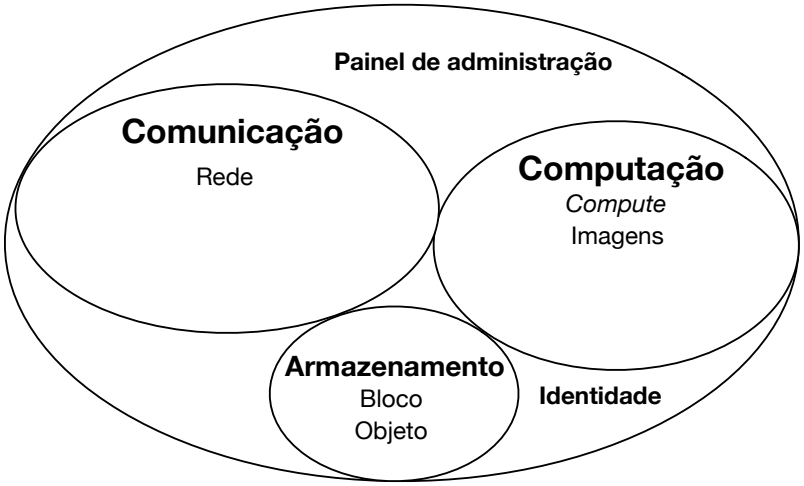


Figura 5 – Principais serviços do OpenStack. Adaptado de (LITVINSKI; GHERBI, 2013).

Os serviços do OpenStack são apoiados por dois componentes: *dashboard* e *identity*. O serviço *compute* gerencia os discos virtuais e metadados associados no serviço de imagens. O *dashboard* fornece um *front-end* baseado em web para o serviço de computação ao passo que o serviço *networking* fornece rede virtual para o *compute*. O serviço *block storage* fornece volumes de armazenamento para o *compute*. O serviço *Image* pode armazenar os arquivos atuais do disco virtual no armazenamento de objetos e todos os serviços autenticam contra o serviço *identity*.

4.1 Características de Escalonamento do OpenStack

O papel do escalonador no OpenStack é direcionar os pedidos de novas instâncias para os nós computacionais adequados. O nós, por sua vez, atenderão as solicitações entregues a eles para lançar e ativar novas máquinas virtuais disponíveis, dimensionadas de acordo com as especificações dos pedidos (HU; YU, 2013). O escalonador faz uso das informações de estado de cada *host*, as quais são enviadas pelos nós de computação como base de todas as decisões de escalonamento.

O OpenStack possui três tipos de escalonadores, por meio do módulo *Compute*, que são: *Filter*, *Chance* e *Simple*.

Quando é feita uma solicitação para iniciar uma nova máquina virtual, o serviço de computação faz contato com o escalonador para solicitar a alocação da nova instância. Por padrão, o escalonador *Filter* é escolhido para determinar a alocação citada anteriormente. Em primeiro lugar, um processo de filtragem é utilizado para determinar quais *hosts* são elegíveis para consideração e uma lista de possíveis *hosts* escolhidos é criada. Em seguida, um segundo algoritmo, baseado em custos e pesos, é aplicado contra a lista para determinar qual nó de computação é o ideal para o cumprimento da solicitação. A Figura 6 ilustra os processos do escalonador *Filter*.

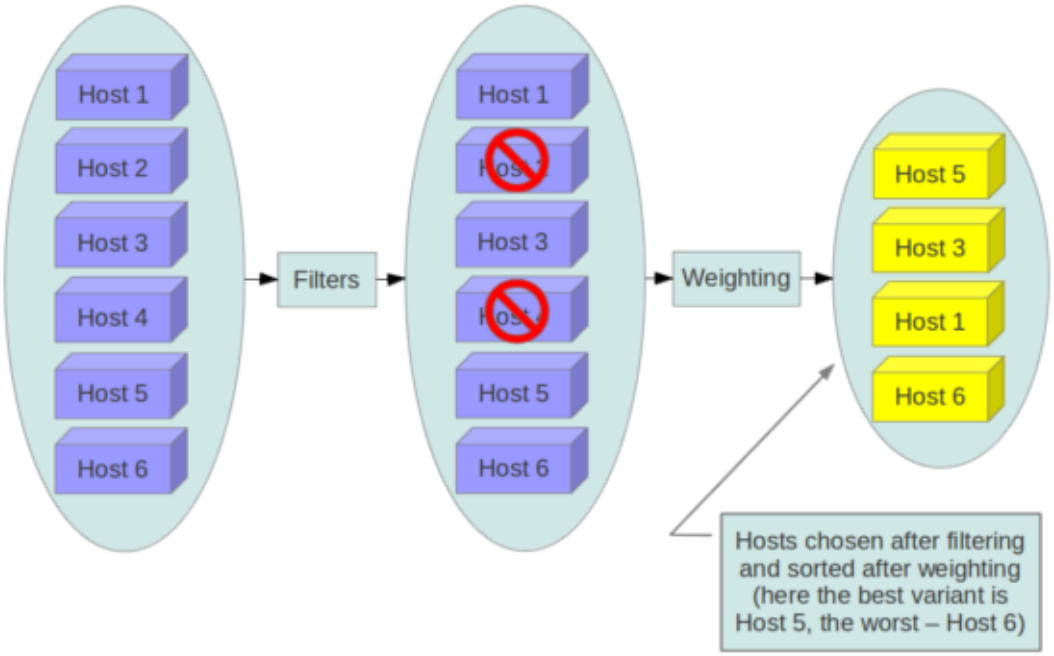


Figura 6 – Escalonador *Filter* (OPENSTACK, 2015).

Na sequência, o escalonador *Filter* aplica uma ou mais funções de custo em cada *host* eleito pelos filtros com a intenção de obter pontuações numéricas para cada máquina física, conforme ilustrado na Figura 7. Cada pontuação é multiplicada por uma constante definida nas configurações do escalonador.

O escalonador *Chance* escolhe aleatoriamente um nó disponível independentemente das suas características enquanto o agendador *Simple* tenta encontrar um nó disponível com a menor carga (LITVINSKI; GHERBI, 2013).

Atualmente, o OpenStack suporta somente uma estratégia inicial de escalonamento de recursos. Somente uma política pode ser habilitada e o *framework* inicial de escalonamento da plataforma de nuvem forçará todos os recursos do *host* a seguirem uma única política de escalonamento. Isto impede o administrador de configurar diferentes otimizações para *pools* de recursos individuais para fins de isolamento de

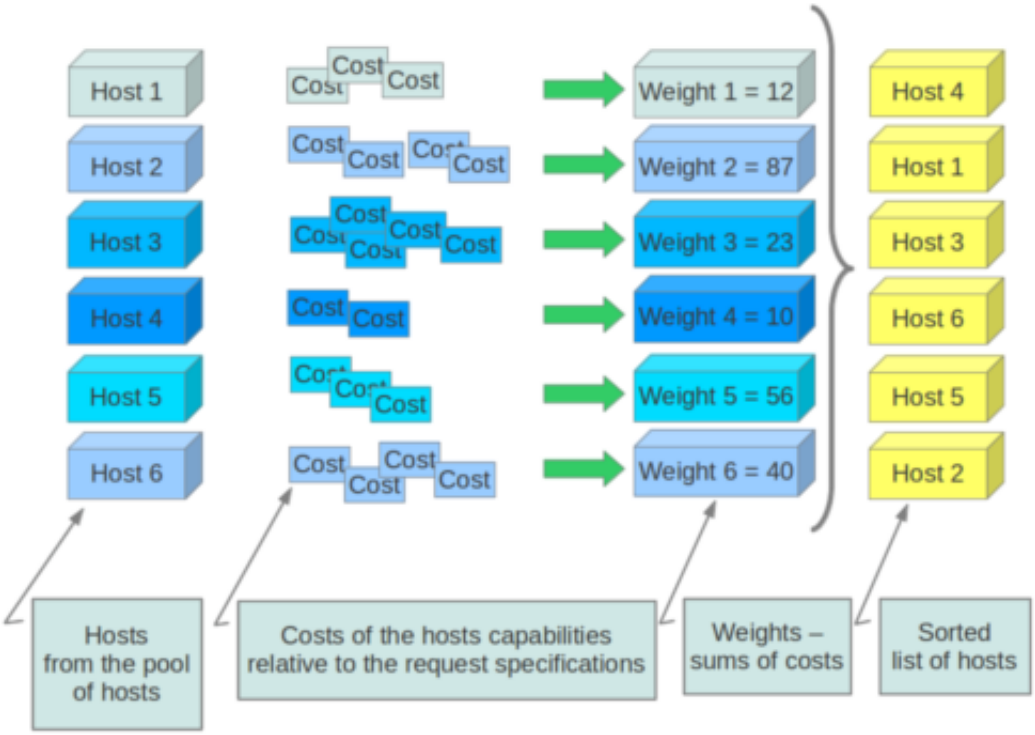


Figura 7 – Escalonador *Chance* ou *Simple* (OPENSTACK, 2015).

desempenho e segurança e afeta a flexibilidade de configuração da política de escalonamento por parte dos usuários (HUANLE; WEIFENG; TINGTING, 2013).

4.2 Considerações Sobre o Capítulo

O OpenStack consiste em um conjunto de softwares, com características semelhantes a de um sistema operacional executando sobre uma estrutura de nuvem computacional. Possui a capacidade de gerenciar conjuntos de recursos de processamento, armazenamento e comunicação de rede e permite, ainda, o controle por parte dos administradores, de todo o provisionamento dos recursos disponíveis aos usuários.

Com relação ao escalonamento de recursos no OpenStack, somente uma estratégia padrão é suportada, ou seja, todos os recursos dos *hosts* serão forçados a seguir uma única política de escalonamento. Embora o esquema padrão possa ser parametrizado, o administrador fica impedido de configurar otimizações mais específicas de acordo com a demanda individual de cada usuário.

No que diz respeito a estrutura de desenvolvimento do OpenStack, suas características permitem a conexão de novos módulos ou extensões que implementem novas funcionalidades.

5 ESCALONAMENTO DE BOT EM UMA NUVEM

Nas seções seguintes será apresentado o modelo sugerido para aplicações BoT concebido para este trabalho e como se dará sua aplicação em ambientes distribuídos de nuvens computacionais. Destaque é dado ao algoritmo de escalonamento aplicativo, responsável pela consolidação de tarefas de BoTs, que representa a principal contribuição deste trabalho.

5.1 Modelo de Aplicação BoT

A infraestrutura provida por nuvens computacionais fornece suporte para a execução de aplicações do tipo BoT. Em muitos casos, as tarefas pertencentes às aplicações BoT possuem um alto custo computacional e para evitar grandes investimentos em recursos de hardware, usuários podem submeter às infraestruturas computacionais em nuvem, de forma oportunista, suas demandas de processamento para este tipo de aplicação.

Neste trabalho, uma aplicação BoT é descrita por um conjunto A composto por $n \geq 1$ quádruplas descrevendo, cada uma, um conjunto de tarefas τ, na forma:

$$A = \{q_1 \dots q_n\} \text{ onde } q_i = [a_i, d_i, b_i, c_i], \forall i | 1 \leq i \leq n$$

Os elementos destas quádruplas são: ***a***: é o instante de tempo no qual o conjunto de tarefas chega para o processamento; ***d***: é o tempo de duração do grupo de tarefas; ***b***: é a quantidade de tarefas descritas por quádrupla correspondente; e ***c***: é a taxa de utilização de CPU para cada tarefa da quádrupla. O valor informado para o custo deve estar entre 1 e 100, indicando o percentual de uso da CPU de cada uma das tarefas quando em execução. As informações relacionadas a tempo, a e d, são informadas na unidade adotada. No modelo adotado, as informações de tempo consideram a existência de um número não limitado de processadores e sua execução em um ambiente livre de contenção. Assim, cada quádrupla q_i descreve um grupo de b_i tarefas $\tau_i^j, \forall j | 1 \leq j \leq b_i$ que são inseridas no *bag* no tempo a_i, com duração d_i e custo individual de c_i.

O número total de tarefas de uma aplicação A composta por n quádruplas é dado por:

$$N_A^n = \sum_{i=1}^{n} b_i$$

Não existe nenhuma restrição de ordem de execução entre as tarefas definidas em cada quádrupla, uma vez que, por definição, as tarefas em um BoT são independentes. No entanto, a data de chegada a_i das tarefas definidas pela tupla q_i impõe que nenhuma tarefa definida por q_i inicie antes do tempo a_i.

A seguir é apresentado um exemplo da descrição de uma aplicação conforme o modelo definido.

$$A = \{[0, 5, 3, 50], [0, 8, 5, 30], [3, 4, 6, 80], [4, 10, 2, 20]\}$$

O exemplo de aplicação apresentado consiste em uma aplicação A composta de um total de 16 tarefas, distribuídas em quatro grupos de tarefas idênticos. Os dois primeiros grupos definem tarefas que chegam no tempo 0 (zero) do processamento. As tarefas do terceiro e do quarto grupo são adicionadas ao *bag* nos tempos 3 e 4. No modelo adotado, estes tempos consideram a existência de um número não limitado de processadores e sua execução em um ambiente livre de contenção.

Para efeito de controle de evolução das aplicações BoT, o tempo é assumido discreto. Desta forma, os atributos descrevendo unidade tempo, os atributos a e d das quádruplas (tempo de chegada e duração) definem a relação de granularidade entre as tarefas, nas quais sendo maior o tempo, mais grossa a granularidade, podendo cada unidade ser mapeada em um valor qualquer, alterando, para uma mesma aplicação, a granularidade de todas as tarefas de modo uniforme.

Define-se *passo* a passagem de uma unidade de tempo. Desta forma, a duração d_i de uma tarefa τ_i descreve o número de passos necessários para executá-la. Ao conjunto de instruções de passo de uma tarefa se dá o nome de *job*. Assim, cada tarefa τ_i define d_i *jobs* pelo conjunto $\{w_1 \dots w_{d_i}\}$, os quais devem ser executados na estrita sequência de ordem na qual foram definidos, garantindo que w_1 não seja executado antes do passo definido por a_i. O passo a_i deve ser interpretado, no caso da execução da aplicação em uma arquitetura não contingenciada, pelo número total de passos previamente executados pela aplicação. Inexistem relações de ordem entre os *jobs* de diferentes tarefas, pelas razões já expostas. O número total de *jobs* de uma aplicação A descrita por n quádruplas é dado por:

$$N_A^* = \sum_{i=1}^{n} d_i \times b_i$$

Na Figura 8 são apresentados os 6 primeiros passos na evolução da aplicação A descrita no exemplo anterior. Observe que nesta representação está sendo assumido a existência de um número não limitado de recursos de processamento e inexistência de condições de contenção de execução. No passo inicial (passo=0), 8 tarefas estão presentes no bag e são lançadas para execução. Terminado o passo, avança o tempo (passo=1) e um *job* de cada uma das tarefas é completado. O mesmo ocorre ao término do passo 1 e do passo 2. No passo 3, um novo conjunto de tarefas é recebido no bag e lançado. Ao final do passo 3, todas as tarefas em execução finalizam a execução de um *job*. No passo 4, como no passo anterior, um novo conjunto de tarefas inicia a execução e todas as tarefas completam a execução de um *job*. Neste passo ocorre também que um conjunto de tarefas executa seu último *job*, finalizando sua execução. No passo seguinte (passo=5) as tarefas remanescentes avançam, concluindo mais um passo. A execução prossegue reproduzindo o padrão apresentado.

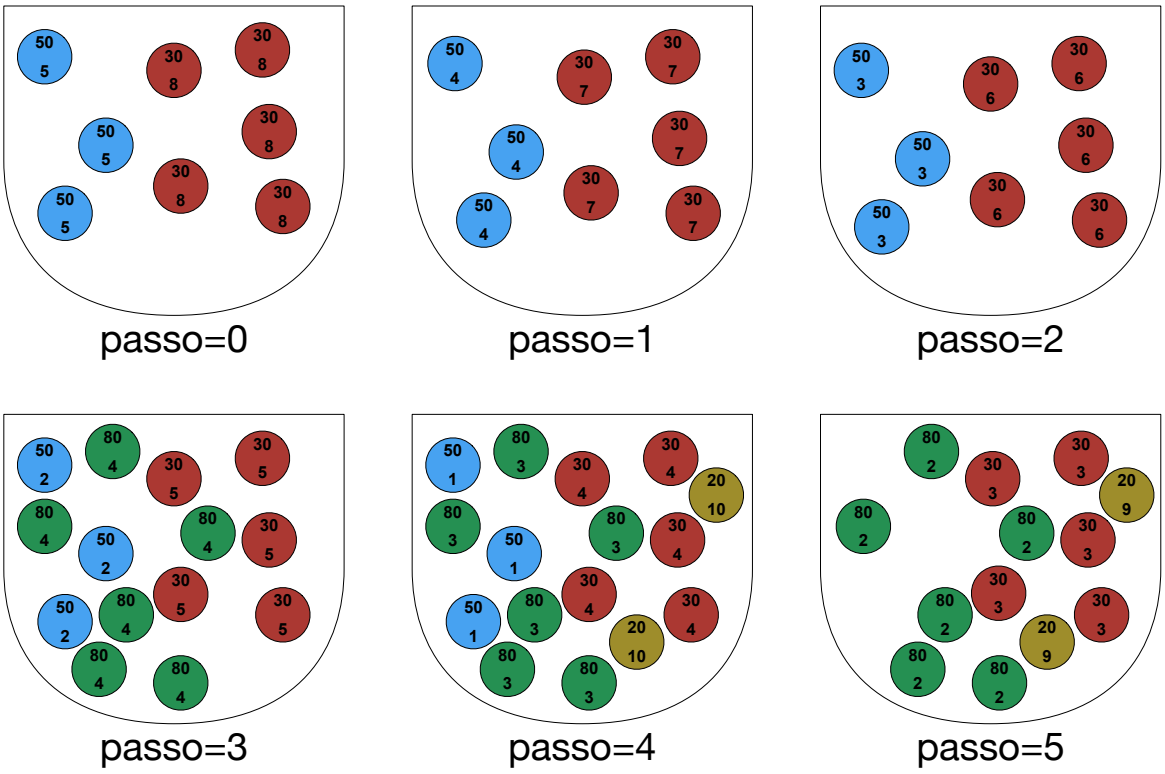


Figura 8 – Exemplo de aplicação *Bag of Tasks*.

5.2 Consolidação de Tarefas

A consolidação de tarefas consiste em uma etapa de escalonamento, realizada em nível aplicativo, que realiza o mapeamento das tarefas definidas em uma aplicação sobre um conjunto finito de recursos de processamento. Os critérios de escalonamento para tal consolidação levam em consideração unicamente aqueles atributos empregados para descrever aplicações BoT apresentados na Seção 5.1: data de chegada,

duração, número de tarefas, quantidade de processamento requerida e o número de processadores disponíveis.

Assume-se que as seguintes situações possam ser verdadeiras em um determinado instante de execução:

- O número total de tarefas é maior que o total de processadores disponíveis.
- A quantidade de processamento requerida pelo conjunto de *jobs* aptos a executar é superior àquela oferecida pelo conjunto de processadores disponíveis.

5.2.1 Discretização das tarefas

Embora em uma aplicação do tipo BoT inexistam relações de dependência, tampouco precedência entre tarefas, uma tarefa qualquer pode gerar novas tarefas para o bag. Assim, considere duas tarefas τ_k^i e τ_l^j, definidas em duas quádruplas q_i e q_j, de tal forma que $a_i < a_j$, existindo, portanto, um valor δ tal que $a_i + \delta = a_j$. Isto significa que, pelo menos uma tarefa em q_i pode ter criado q_j ao final da execução de um *job* $w_\delta^{\tau^i}$. Como o modelo de aplicação BoT adotado não distingue as tarefas pertencentes a uma mesma quádrupla, assume-se que todas as tarefas τ_l de q_j são inseridas no bag após o término do passo w_δ de todas as tarefas τ_k de q_i.

Define-se *passo global* g um instante de tempo discreto na evolução de uma aplicação BoT no modelo descrito em um ambiente com número não limitado de recursos de processamento e sem contenção. Uma aplicação qualquer, descrita por n quádruplas possui $|g| = \max_{i=1}^{k}(a_i + d_i)$ passos globais. Nesta situação, a passagem de g_i para g_{i+1} marca o término da execução de um *job* de cada tarefa presente no bag até g_i, inclusive.

Os atributos de um *job* w_j^i, criado a partir de uma tarefa τ_k^i especificada em uma quádrupla q_i, são descritos por uma quádrupla contendo $[g_j^i, r_j^i, f_j^i, c_j^i]$, onde g_j^i corresponde ao passo global em que o *job* encontra-se disponível para execução e c_j^i ao custo do *job* definido pela tarefa. Nesta quádrupla, os membros r_j^i e f_j^i, respectivamente, representam o número de *jobs* de τ_i que ainda estão no bag e os que já foram concluídos. Para um dado *job* w_j^i, $g_j^i = a_i + f_j^i$, bem como $r_j^i + f_j^i = b_i$ e $c_j^i = c_i$.

Para uma determinada tarefa τ_k^i, são instanciados d_i *jobs* na forma:

$$\{w_1^i, w_2^i, \ldots w_{d_i}^i\} = \{[g_1^i, r_1^i, f_1^i, c_1^i], [g_2^i, r_2^i, f_2^i, c_2^i], [g_3^i, r_3^i, f_3^i, c_3^i] \ldots [g_{d_i}^i, r_{d_i}^i, f_{d_i}^i, c_{d_i}^i]\}$$

Para efeito de consolidação das tarefas, assume-se como unidade de manipulação o *job*. Os *jobs* definidos por diferentes tarefas são independentes entre si. Os *jobs* definidos por uma tarefa, no entanto, são executados na ordem expressa pelos respectivos g_i.

5.2.2 Contingenciamento de execução

Em um ambiente real, a capacidade de processamento é limitada, tanto em termos de recursos, definidos pelo número de processadores ou quantidade de memória, como em termos de capacidade, exemplificada pela velocidade de comunicação. A aplicação em execução, em tais sistemas, é a principal fonte de consumo destes recursos, não sendo, no entanto, a única. A atuação de um sistema de gerenciamento, como o provido pelo mecanismo de escalonamento, também consome a capacidade de processamento oferecida. No modelo BoT apresentado, é considerado apenas o custo computacional de cada tarefa, o que implica em considerar a carga de processamento oferecida pela arquitetura disponível, assumindo, desta forma, ilimitada capacidade de processamento, armazenamento e nulos os custos de comunicação. Assume-se, também, que a influência do mecanismo de escalonamento não introduz sobrecustos à execução.

O número de processadores disponíveis na arquitetura real delimita, no modelo adotado, a capacidade de processamento disponível. Uma arquitetura M dedicada à execução de uma aplicação BoT A é composta por m processadores idênticos, $\{p_1, p_2 \dots p_m\}$, com ilimitada capacidade de memória por processador e custo 0 (zero) para comunicação de tarefas e operações de escalonamento entre eles. Em determinado passo global, a quantidade de processamento máximo disponível nesta máquina é dado por $C = m \times 100\%$.

O número de *jobs* que podem ser executados em um determinado passo global g é limitado pela quantidade de processamento oferecida. Assim, se em um determinado passo global g o somatório dos custos computacionais c_k^i dos *jobs* w_i aptos à execução suplantar a capacidade C de processamento oferecida, haverá contenção na execução. Deve também ser observado que a necessidade de processamento de um *job* não pode ser dividida entre processadores. Como consequência, pode ocorrer fragmentação de uso de processador quando, em um passo global, um novo *job* não pode ser adicionado aos demais por ultrapassar o limite de uso de 100% de CPU máximo previsto por passo. Assim, a capacidade real C_g explorada em um passo global g é limitada em $C_g \leq m \times 100\%$.

5.2.3 Normalização do passo global

Como existe contingenciamento na execução dos *jobs*, em função da carga computacional disponível, em um dado passo global g nem todos os *jobs* elegíveis para execução podem ser de fato executados. Assim, cada tarefa terá, no máximo um *job* em execução, havendo a possibilidade de tarefas não terem nenhum *job* executado em um determinado passo global.

Para cada tarefa τ_k^i, apenas um *job* é elegível para execução, sendo este o *job* que possuir o menor g_j^i, satisfazendo $g_j^i \leq g$. Considerando dois *jobs* w_j^i e w_k^i definidos por

uma mesma tarefa τ_i a relação de ordem temporal definida pelo atributo g_i de cada um deles deve ser preservada.

Considerando dois *jobs* w_j^i e w_k^h, definidos pelas tarefas τ_k^i e τ_l^h, respectivamente, ambas elegíveis para execução, não existe nenhuma ordem pré-definida de execução entre eles.

Em consequência do contingenciamento, em um dado passo g pode existir um conjunto não vazio de *jobs* em que se verifica que $g_i \leq g$. Uma parcela limitada das tarefas já iniciadas contendo estes *jobs* é elegível para execução em função da limitação dos recursos de processamento disponíveis. As demais, não elegíveis por não haver recursos de processamento suficientes, terão sua execução retardada.

Deste fato destaca-se a consequência da contingência no modelo de BoT adotado. Conforme previsto na Seção 5.1, o atributo a_i definido por uma quádrupla q_i representa a quantidade de processamento já realizada pela aplicação em uma arquitetura não contingenciada. Portanto, todo atraso na execução de qualquer *job* reflete no atraso, na mesma proporção, da chegada de um novo lote de tarefas (e seus *jobs* correspondentes) da mesma aplicação.

A Figura 9 apresenta um exemplo da situação. Supondo duas aplicações $A = \{[0, 5, 1, c_1^A], [3, d_1^A, 2, c_1^A]\}$ e $B = \{[3, d_1^B, 3, c_1^B]\}$, cujos atributos não identificados são irrelevantes para compreensão do exemplo. Nesta figura, *jobs* são representados por círculos, sendo envolvidos pelas tarefas que os descrevem. O inteiro informado no interior de cada *job* refere-se ao seu passo de execução. As setas tracejadas identificam a dependência de criação de tarefas anotada entre os *jobs*. A discretização do tempo (passos) também é apresentada.

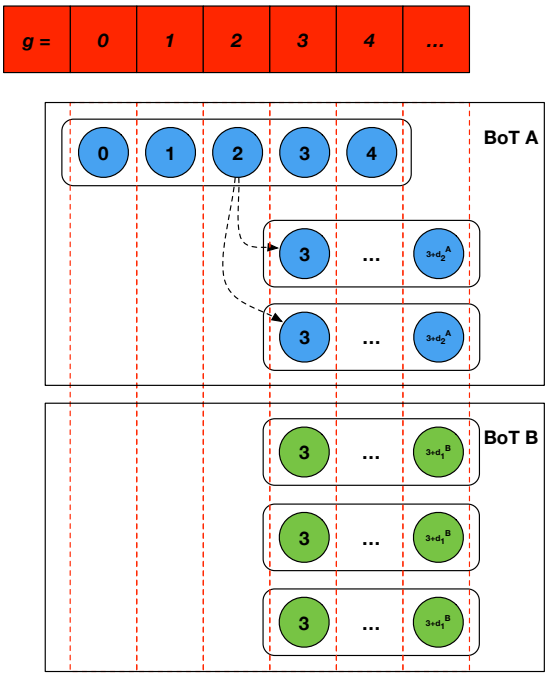


Figura 9 – Dependência na criação de tarefas entre *jobs*.

Na Figura 9, o BoT A possui dois grupos de tarefas, o primeiro é recebido no tempo 0 (zero), sendo composto por uma (1) tarefa com comprimento de cinco passos, ou seja, de cinco *jobs*. O segundo grupo é composto por duas tarefas, sendo recebido no tempo 3 em uma arquitetura sem contenção. A interpretação é a de que de todas as tarefas lançadas anteriormente, na mesma aplicação, devem ter tido o direito de executar até 3 passos, ou seja, até 3 de seus *jobs*, de forma que a quantidade de trabalho suficiente para criação deste segundo grupo tenha sido completada.

Nesta mesma figura, o BoT B, possui um único grupo de tarefas, composto por três tarefas, que é recebido no tempo 3. A execução destas tarefas não depende de trabalhos realizados em outra aplicação, uma vez que trata-se de um novo BoT. Seus *jobs* não são atrasados, portanto, em decorrência de atrasos de execução de *jobs* descritos em outro BoT, no caso da figura, das tarefas do BoT A.

5.3 Escalonamento em Nível Aplicativo

O escalonamento em nível aplicativo consiste no mapeamento dos *jobs* gerados por uma aplicação BoT em um conjunto de recursos de processamento limitados.

5.3.1 Operação básica

O processo de escalonamento em nível aplicativo consiste em alocar os *jobs* que descrevem uma aplicação BoT descrita pelo modelo apresentado na Seção 5.1 sobre um conjunto limitado de processadores. O escalonador será considerado correto se:

- Respeitar as dependências temporais expressas entre os *jobs*;
- Executar cada um dos *jobs* apenas, e no máximo, uma única vez;
- Finalizar a execução de todos os *jobs* em tempo finito;
- Não alocar uma quantidade de *jobs* a um processador superior àquela que o processador pode suportar;
- Não deixar em espera um *job* elegível para execução caso exista pelo menos um processador com capacidade de processamento disponível suficiente para executá-lo.

A cada passo global da execução de uma aplicação BoT, o escalonador pode executar as seguintes operações:

- Seleção de um *job* para execução;
- Atribuição de *job* para processador;

- Preempção de tarefa;
- Migração de tarefa.

A operação de seleção deve respeitar a eligibilidade dos *jobs*. Esta operação pode aplicar uma função-critério, na qual algum atributo, ou alguma combinação de atributos, indica qual o *job* mais apto para a seleção em um determinado momento.

A operação de atribuição de um *job* a um processador corresponde a adicionar a carga de processamento de um processador, em um determinado passo. Como todos os processadores são idênticos, e inexistem custos de comunicação ou limite de memória, a atribuição a um processador específico considera apenas a capacidade ociosa de processamento de cada um dos processadores. Esta atribuição pode ser realizada de duas formas: por lote ou cíclica. Na alocação por lote, é selecionado para cada processador um conjunto de *jobs* tal que sua capacidade de processamento seja ocupada, de forma a não ultrapassar 100% da capacidade oferecida. Na alocação cíclica, cada processador recebe um *job* por vez, repetindo o ciclo até que não seja possível alocar nenhum outro *job* a nenhum processador.

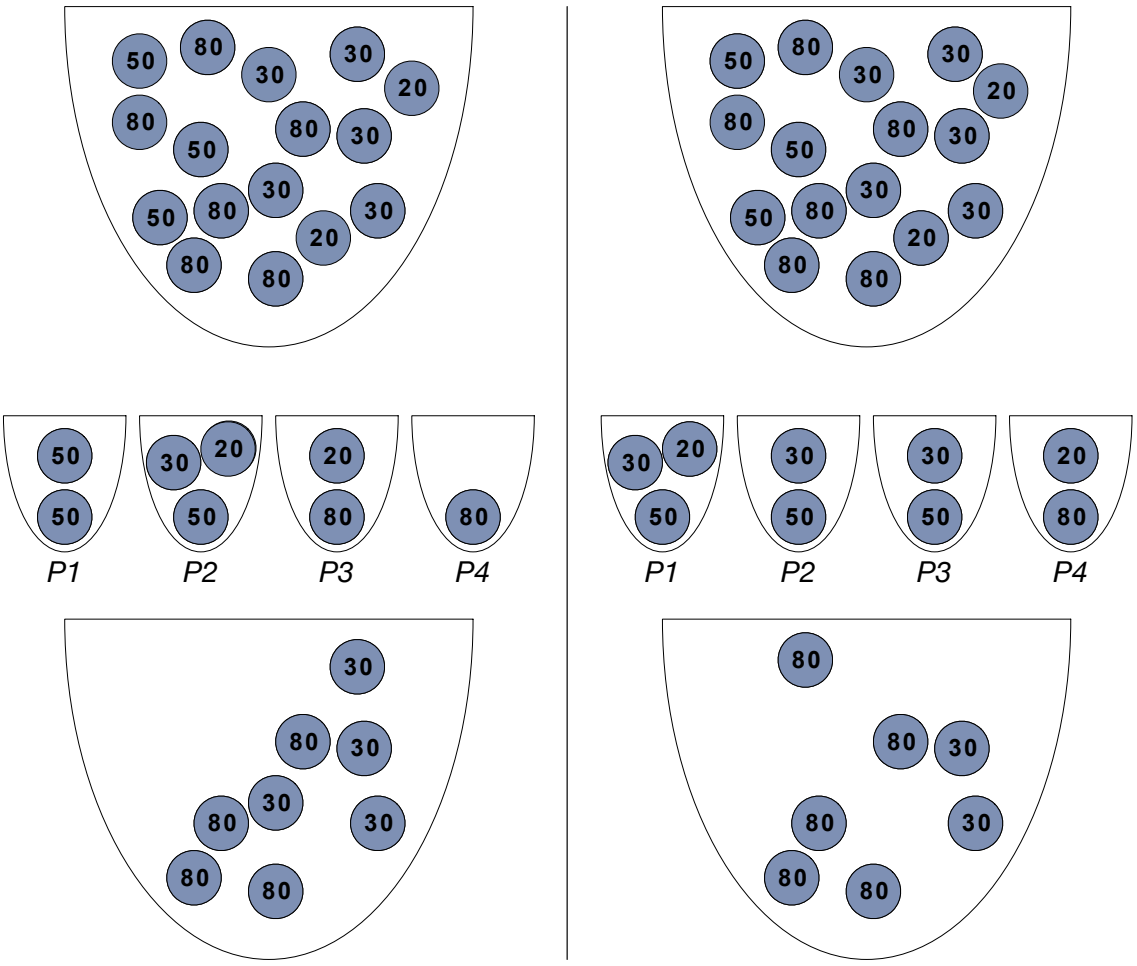
A Figura 10 representa os dois mecanismos de atribuição, considerando a existência de 4 processadores. Nesta figura, os *jobs* são representamos por círculos, sendo anotado seu custo computacional. A grande sacola contém os *jobs* aptos a executar no passo global considerado (passo g) e aqueles remanescentes para o passo $g + 1$ inicialmente disponíveis e sua configuração após a atribuição. Para cada mecanismo é considerada e mesma sacola inicial. As pequenas sacolas representam a alocação realizada a cada processador. A Figura 10.a representa a alocação por lote e a Figura 10.b a alocação cíclica. Nota-se que a aplicação dos diferentes mecanismos de alocação resulta em diferentes configurações de carga dos processadores e da quantidade de carga remanescente na sacola.

Os *jobs* não são migrados, pois requerem apenas um passo para execução. No entanto, tarefas podem ser preemptadas quando a sequência de execução de seus *jobs* pode ser interrompida em favor da sequência de *jobs* determinada, segundo os critérios de seleção adotada, por outra tarefa.

A migração de tarefas também é possível, pois, considerando a homogeneidade dos recursos de processamento e a inexistência de custos de comunicação, não é realizada associação de afinidade de *jobs* pertencentes a uma tarefa a um determinado processador.

5.3.2 Algoritmo de lista

O conjunto de tarefas descrito por uma aplicação BoT permite a construção de uma lista de *jobs*. Estes *jobs* são ordenados, segundo alguma função-critério, de forma a determinar a prioridade de execução entre eles. Esta lista de *jobs* é criada de forma



(a) Alocação por lote (b) Alocação cíclica

Figura 10 – Mecanismos de alocação de *jobs* a processadores.

estática, considerando que todas as informações de tempo correspondem a passos globais em uma arquitetura não contingenciada.

O escalonamento $\mathbb{S}'$ consiste em uma função $A^{m'} = \mathbb{S}' < F > (A, m')$, onde:

- A: Representa uma aplicação BoT;
- m': É o número de processadores disponíveis para execução da aplicação;
- $A^{m'}$: Representa o resultado do escalonamento na forma de uma matriz $t \times m'$;
- F: Representa o critério de priorização das tarefas, sendo aplicado na ordenação da lista de *jobs*. Consiste em uma função-critério.

O resultado do escalonamento $A^{m'}$ é apresentado na forma de uma matriz $t \times m'$. Cada uma das m' colunas representa, linha a linha, a variação da carga alocada a um processador. As linhas representam a passagem do tempo, sendo t o comprimento do escalonamento obtido.

O escalonamento $\mathbb{S}'$ aplica, de forma iterativa, o Algoritmo 1, interrompendo sua execução quando toda a carga gerada por A tiver sido alocada. No Algoritmo 1 é realizada a consolidação de tarefas de uma aplicação BoT sobre um conjunto finito de processadores. Neste algoritmo, os seguintes dados de entrada são considerados:

- L: Consiste na lista de *jobs* a ser escalonada. Esta lista é construída a partir das tarefas definidas em uma aplicação A descrita na forma $A = \{q_1, ..., q_n\}$.
- m': É o número de processadores disponíveis para execução da aplicação.
- P: Pode conter apenas um de dois valores: *Lote* ou *Cíclico*, indicando qual método de alocação de *jobs* aos processadores deve ser aplicado, se por lote de tarefas ao processador ou se de forma cíclica, tarefa a tarefa, entre os processadores.
- g: Informa o passo global da execução que deve ser considerado para identificação dos *jobs* elegíveis.

A saída do algoritmo corresponde à alocação dos *jobs* em um determinado passo global e a lista de *jobs* atualizada pela remoção dos *jobs* escalonados no passo considerado. Estas saídas são dadas por:

- L: Lista de *jobs* em que os *jobs* escalonados no passo considerado foram removidos.
- M': Carga computacional dos m' processadores após a alocação dos *jobs*.

No Algoritmo 1, os dados locais são os seguintes:

Algoritmo 1 Consolidação de uma lista de tarefas em um conjunto de processadores.

```
1: procedure CONSOLIDATAREFAS< F >(L, m', P, g)
2:     L' ← ∀w_i ∈ L|g_i = g
3:     L ← L − L'
4:     L' ← Sort < F > (L')
5:     if P = Lote then                                   ▷ Alocação por lote
6:         for i ← 1 to m' do
7:             |P_i| ← 0
8:             for j ← L'.begin to L'.end do
9:                 if |P_i| + c_j ≤ 100 then
10:                    |P_i| ← |P_i| + c_j
11:                    L' ← L'.remove(j)
12:                end if
13:            end for
14:        end for
15:    else                                               ▷ Alocação cíclica
16:        while ∃w_k ∈ L', 1 ≤ i ≤ m'|c_k + |P_i| ≤ 100 do
17:            for j ← L'.begin to L'.end do
18:                for i ← 1 to m' when j ≠ L'.end do
19:                    if |P_i| + c_j ≤ 100 then
20:                        |P_i| ← |P_i| + c_j
21:                        aux ← j.next
22:                        L' ← L'.remove(j)              ▷ Um job alocado
23:                        j ← aux
24:                    end if
25:                end for
26:            end for
27:        end while
28:    end if
29:    L ← L ∪ L'
30:    ∀w_i ∈ L, g_i ← g_i + 1
31:    M' ← {|P_1|, ..., |P'_m|}
32:    return L, M'
33: end procedure
```

- $|P_i|$: Refere-se a carga do processador P_i.
- w_i: Descrição de um *job* na forma $w_i = [g_i, r_i, f_i, c_i]$.
- L': Lista de *jobs* cujo passo é igual ao passo considerado no escalonamento ($g_i = g$).
- i: Iterador sobre o número de processadores disponíveis, representando um índice sobre cada processador.
- j: Iterador sobre a L', representando um *job* w desta lista por vez.

Neste algoritmo, as instruções nas linhas 2 e 3 movem da lista de *jobs* de entrada todos os *jobs* que possuem um passo de execução igual ao passo global a ser considerado. Na linha 4, a lista de *jobs* a serem considerados neste passo de escalonamento é ordenada conforme o critério de prioridade considerado. A ordem de prioridade é decrescente, sendo o primeiro *job* da lista o de maior prioridade. Então segue-se a alocação de *jobs* aos processadores.

Caso a alocação seja realizada por lote, os processadores são visitados (linha 6) e removidos da lista de *jobs* (linha 11), tantos *jobs* quantos forem necessários para cobrir a capacidade de processamento disponível pelo processador (linha 09), sendo 100% a carga total disponível inicialmente. A seleção de *jobs* considera a prioridade dada pela ordem de *jobs* na lista, sendo atribuídos a um processador i tantos *jobs* quantos necessários para atingir o limite definido. Assim, um *job* de menor prioridade pode ser alocado a um determinado processador caso pelo menos um *job* de maior prioridade não possa a ele ser atribuído por extrapolar a capacidade de processamento ainda disponível. Esta situação é descrita pelo laço da linha 8.

Na alocação cíclica (linha 16), enquanto houver pelo menos um processador que possa prover a necessidade computacional de pelo menos um dos *jobs* na lista, a lista é percorrida (linha 17) de forma a alocar o primeiro *job* encontrado, ou seja, o mais prioritário, no processador de menor índice em que sua carga computacional possa ser suprida (linha 19). No caso da alocação ter sido bem sucedida, o *job* é removido da cabeça da lista (linha 22) e a nova cabeça de lista será considerada para alocação ao processador $i+1$ (linha 23). Caso negativo, um novo *job* da lista é buscado de tal forma que seu custo computacional seja compatível com aquele disponível no processador considerado.

As linhas 29 a 31 do algoritmo compõem o resultado final do algoritmo, que consiste no resultado da alocação realizada no passo global considerado. São retornados à lista de *jobs* aqueles *jobs* considerados para alocação na rodada realizada mas que, por questões de prioridade, não foram alocados. Na lista resultante (linha 30), o passo de cada *job* é incrementado de 1 (uma) unidade, de forma a preservar a distância entre *jobs* definidas em termos de uma execução não contingenciada. Assim não ocorre

que, terminado um passo global, permaneça na lista algum *job* com passo de execução inferior ou igual ao passo considerado. O vetor M', por sua vez, contém a carga acumulada para cada processador da máquina explorada.

5.3.3 Critério de prioridade entre *jobs*

Os *jobs* de uma aplicação BoT, descritos por quádruplas na forma $[g_i, r_i, f_i, c_i]$ possuem uma relação de ordem temporal especificada em termos da data especificada pelo seu passo. Esta relação de ordem, mais do que uma prioridade a ser considerada entre os *jobs*, consiste em uma imposição de ordenamento. A linha 29 do Algoritmo 1 garante esta propriedade. Assim, a priorização de *jobs* pertencentes a um mesmo passo pode considerar qualquer um dos membros r_i, f_i e c_i que o descreve ou de uma combinação entre eles. O Algoritmo 2 apresenta o procedimento de ordenação.

Algoritmo 2 Atribuição de prioridades a *jobs* em um passo global.

1: **procedure** SORT$< F >(L)$
2: $\quad \forall w_i, w_j \in L \wedge w_i \neq w_j, i < j \iff F(w_i, w_j) \equiv true$
3: **end procedure**

No algoritmo de ordenação, a entrada é representada pela lista de *jobs* a ser considerada no passo e a saída é dada pela própria lista, garantindo o atendimento ao critério especificado pela função-critério F. Duas possíveis implementações para F são apresentadas nos algoritmos 3 e 4.

Algoritmo 3 Definição da prioridade de *jobs* em função do custo computacional.

1: **procedure** $ComparaMaiorCusto(w_i, w_j)$
2: $\quad$ **if** $c_i \geq c_j$ **then**
3: $\quad\quad$ **return** $true$
4: $\quad$ **else**
5: $\quad\quad$ **return** $false$
6: $\quad$ **end if**
7: **end procedure**

No Algoritmo 3, dados dois *jobs* w_i e w_j, a função-critério $ComparaMaiorCusto$ retorna $true$ caso o custo de w_i for maior que w_j e $false$ caso contrário.

No Algoritmo 4, dados dois *jobs* w_i e w_j, a função-critério $ComparaTarefaMaisAntiga$ retorna $true$ caso a tarefa que tenha originado w_i tenha completado um número maior de passos que w_j e $false$ caso contrário. A interpretação desta função critério, considerando a semântica do escalonamento aplicado, permite identificar o *job* pertencente a tarefa que foi iniciada há mais tempo.

Algoritmo 4 Definição da prioridade de *jobs* em função da data de criação da tarefa.

```
1: procedure ComparaTarefaMaisAntiga(w_i, w_j)
2:     if f_i − g_i ≥ f_j − g_j then
3:         return true
4:     else
5:         return false
6:     end if
7: end procedure
```

5.3.4 Resultado do escalonamento

O resultado do escalonamento em nível aplicativo consiste na discretização, em termos de passos globais de execução, da carga computacional de cada um dos processadores disponibilizados para computação da aplicação. Assume-se que estes processadores executarão a carga computacional especificada em cada um de seus passos de forma independente.

5.3.5 Exemplo

Um exemplo do processo de escalonamento em nível aplicativo é apresentado partindo da Figura 11. Nesta figura é apresentada uma aplicação BoT descrevendo 19 tarefas distribuídas em 6 quádruplas. O total gerado é de 46 *jobs*. A figura apresenta também o passo global, em uma arquitetura não contingenciada, de cada *job*.

Um exemplo de resultado do escalonamento em nível aplicativo é apresentado nas figuras 11, 12 e 13, refletindo o escalonamento obtido. Nesta sequência de figuras, a aplicação observada é dada por:

$$A = \{[0, 2, 3, 20], [1, 4, 3, 30], [1, 2, 5, 50], [2, 2, 2, 60], [2, 3, 4, 40], [3, 1, 2, 80]\}$$

Para a aplicação em questão são consideradas tanto alocação por lote quanto cíclica, sobre 4 processadores.

Na Figura 11 é destacada a relação entre os *jobs* criados e as tarefas das quais foram originados. Também é informado o passo global da execução em que cada *job* se encontra, em uma arquitetura não contingenciada.

A Figura 12 apresenta o mesmo conjunto de *jobs* após a realização da etapa em que a lista de *jobs* é ordenada por prioridades de *jobs*. A lista à esquerda refere-se a ordenação contemplando apenas o passo global de cada *job* e a lista a direita a aplicação de uma função-critério na ordenação dos *jobs* em um mesmo passo global. A função-critério neste exemplo prioriza a execução dos *jobs* com menor custo computacional.

O contingenciamento, em uma arquitetura dotada de 4 processadores, é apresen-

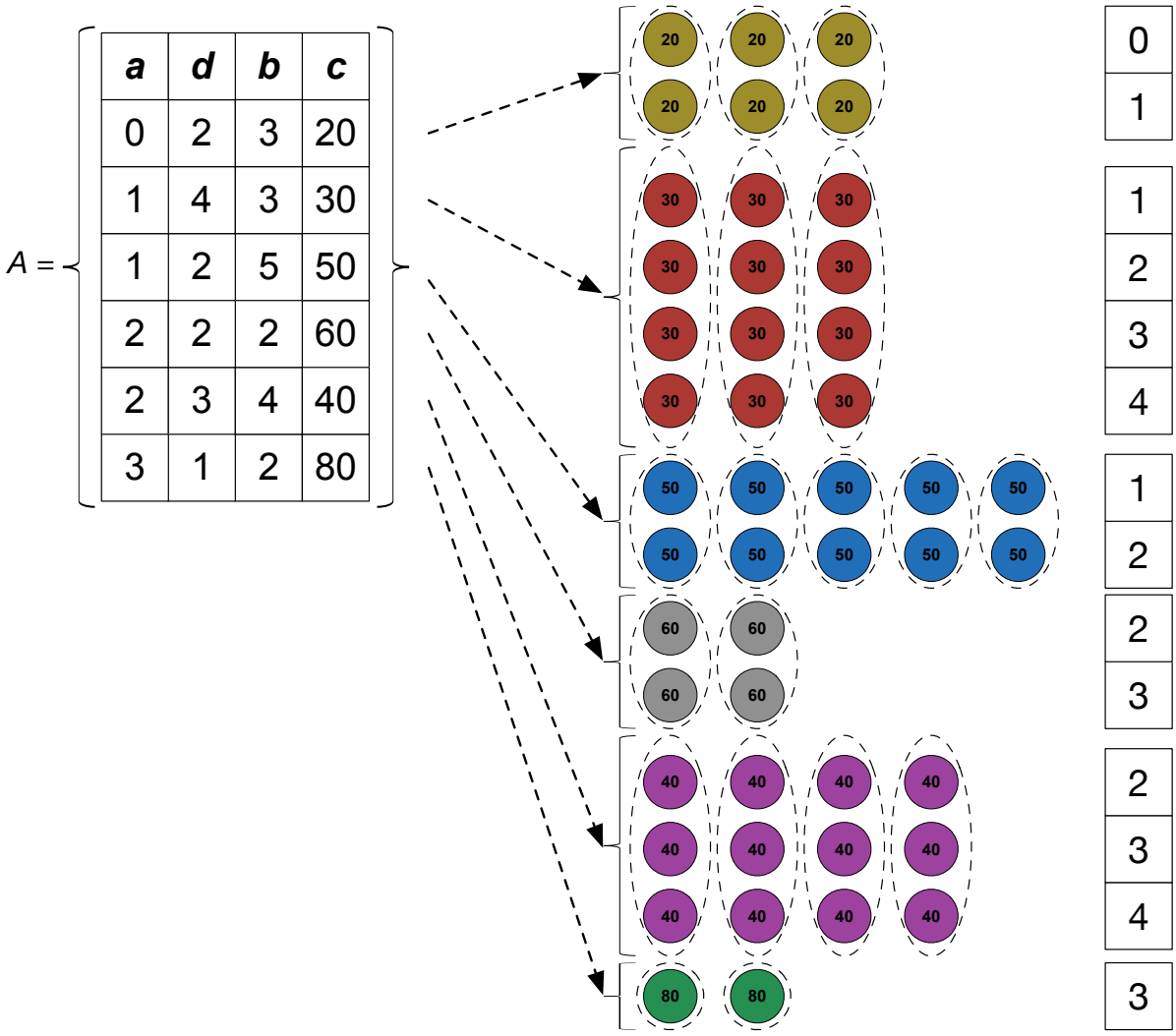
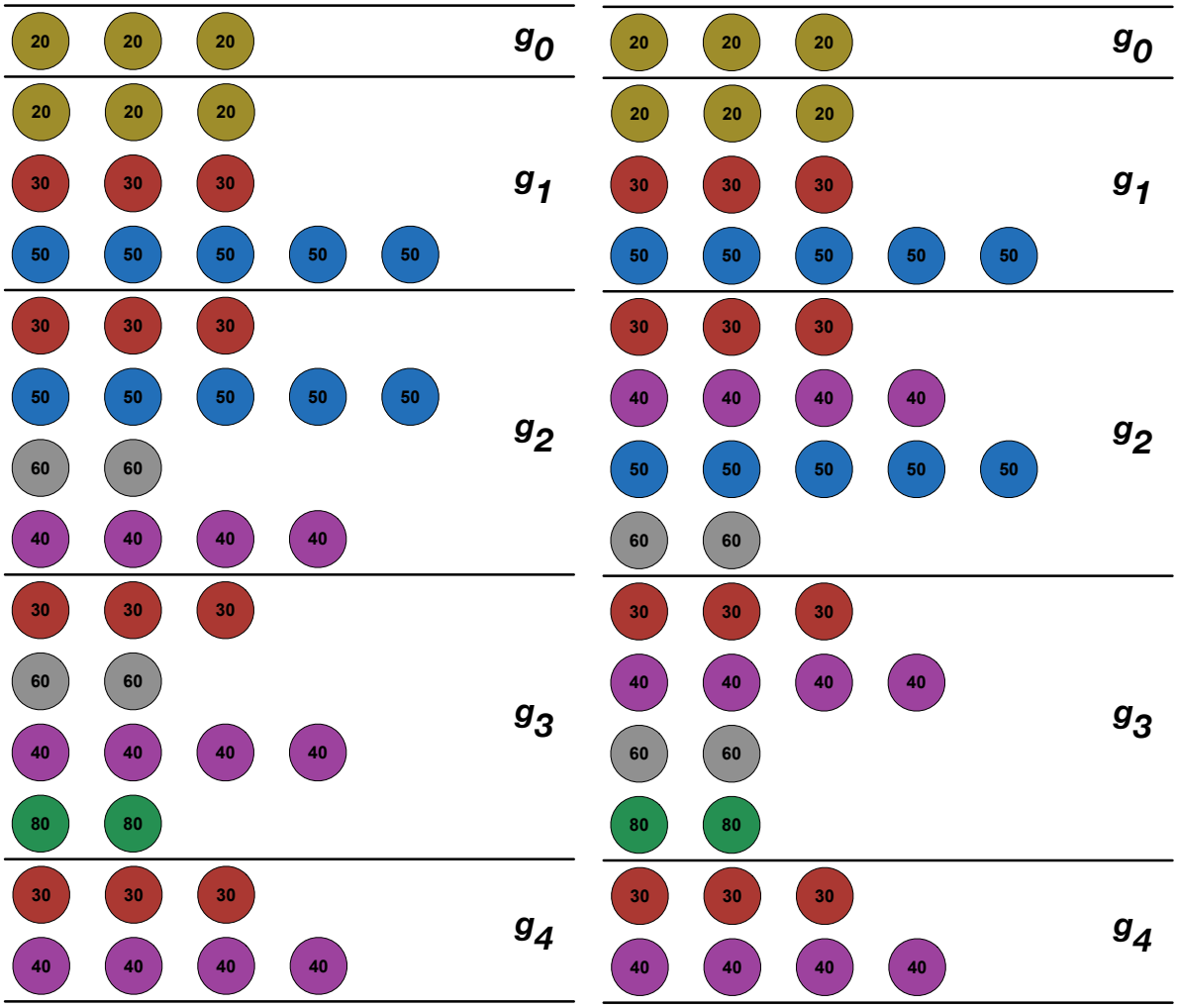


Figura 11 – Geração de *jobs* a partir da descrição de uma aplicação BoT.



(a) Ordenação por passo global (b) Ordenação por função-critério

Figura 12 – Ordenação dos *jobs* em função do passo global não contingenciado.

tado na Figura 13. Nesta figura são reproduzidos os escalonamentos obtidos pela alocação cíclica (Figura 13-a) e pela alocação por lote (Figura 13-b) quando aplicada a função-critério menor custo.

	P_0	P_1	P_2	P_3
g_0	20	20	20	0
$g1$	100	100	70	80
$g2$	50	0	0	0
$g3$	70	70	70	90
$g4$	50	50	50	50
$g5$	60	60	0	0
$g6$	70	70	70	100
$g7$	60	0	0	0
$g8$	70	70	70	40
$g9$	80	80	0	0

(a) Alocação cíclica

	P_0	P_1	P_2	P_3
g_0	60	0	0	0
g_1	90	60	100	100
g_2	50	0	0	0
g_3	90	80	80	100
g_4	100	50	60	60
g_5	90	80	80	80
g_6	60	0	0	0
g_7	90	80	80	80
g_8	80	0	0	0

(b) Alocação por lote

Figura 13 – Resultado final do escalonamento aplicativo.

5.4 Escalonamento em Nível de Sistema

Neste trabalho, o escalonamento em nível de sistema tem a função de explorar uma infraestrutura física, por exemplo um aglomerado de computadores ou uma grade computacional, para suportar o processamento gerado por uma aplicação BoT. No escalonamento projetado, deve ser atendida a demanda especificada, racionalizando o uso dos recursos computacionais.

5.4.1 Operação básica

O modelo de escalonamento adotado para gerenciamento de uma infraestrutura de nuvem prevê a existência de um conjunto finito de recursos de processamento. Assume-se, tendo como base o modelo de aplicação BoT em questão, que estes recursos de processamento possuem memória ilimitada e que o custo de comunicação entre eles é nulo. Assim, apenas a capacidade de processamento do processador é considerada, sendo assumido processadores idênticos. A infraestrutura é definida por:

$$M = \{n_1, n_2, \ldots n_m\}$$

Ao escalonador em nível de sistema é submetida uma aplicação $A^{m'}$ na forma $t \times m'$, na qual estão discretizadas em t passos de execução, as cargas computacionais geradas por uma aplicação BoT A sobre m' processadores idênticos.

Como premissas a serem atendidas pelo escalonamento em nível de sistema, deve ser observado que:

- Toda a carga de processamento submetida deve ser finalizada em tempo finito.
- Qualquer carga computacional especificada para um processador P_i no tempo j, não deve ser executada antes de ter sido concluído o processamento de P_i no tempo $j-1$, ou seja, $|P_i^j| \prec |P_i^{j-1}| \forall i,j | 1 \leq i \leq m' \wedge 2 \leq j \leq t$.

A cada passo da execução de $A^{m'}$, o escalonamento em nível de sistema pode realizar as seguintes operações:

- Identificar a carga de processamento de cada processador da infraestrutura.
- Selecionar cargas computacionais a serem migradas entre processadores da infraestrutura.
- Desativar processadores da infraestrutura.
- Ativar processadores da infraestrutura.

Ativar e desativar nós de processamento da infraestrutura permite racionalizar o uso, desligando processadores (máquinas da infraestrutura) quando não são necessários ao processamento e ligando-os quando a demanda se apresentar.

5.4.2 Modelo de escalonamento em nível de sistema

O escalonamento em nível sistema é realizado sobre uma aplicação $A^{m'}$, a qual é descrita por uma sequência de t m'-tuplas, ou seja, uma matriz $t \times m'$, na forma $A^{m'} = \{(l_1^1, l_2^1, \ldots l_{m'}^1), (l_1^2, l_2^2, \ldots l_{m'}^2), \ldots (l_1^t, l_2^t, \ldots l_{m'}^t)\}$, onde:

- m': grau de paralelismo requerido pela aplicação a cada unidade de tempo da execução. Corresponde ao número de processadores considerados para consolidação de tarefas no escalonamento aplicativo.
- t: tempo total de execução da aplicação, determinado de forma discreta, correspondendo a uma unidade de tempo de execução. Corresponde ao comprimento do escalonamento aplicativo obtido após a consolidação das tarefas em um número limitado de processadores.
- l_i^j: carga de processamento requerido para o processador, $i | 1 \leq i \leq m'$, no passo de execução j. Corresponde à carga do processador P_j no passo global i determinada pelo escalonamento aplicativo.

A função do escalonamento em nível de sistema, dada por $R = \mathbb{S}(A^{m'}, M)$. A função $\mathbb{S}$ é executada de forma iterativa para todos os t passos de execução definidos por $A^{m'}$. A cada iteração é realizada a consolidação das cargas computacionais descritas pela aplicação das operações básicas do escalonamento em nível aplicativo.

5.4.3 Consolidação de máquinas virtuais

Qualquer sequência de cargas $l_i^1, l_i^2, \ldots l_i^t, \forall 1 \leq i \leq m'$ descrita por uma aplicação $A^{m'}$ descreve a carga computacional de um processador P_i. Esta carga, em nível de sistema, é associada a uma *máquina virtual* (MV). Máquinas virtuais são instanciadas sobre nós físicos de uma infraestrutura real. O processo de consolidação de máquinas virtuais realiza a gestão de operação das MVs sobre os recursos de processamento reais disponibilizados na infraestrutura disponível.

O Algoritmo 5 apresenta o conjunto de passos básicos executados pelo escalonamento em nível de sistema. As entradas deste algoritmo são:

- G: Função implementando critério para seleção de nó de processamento que deverá originar a migração de carga, representada por uma ou por um conjunto de MVs, para um nó com carga de processamento disponível.
- H: Função implementando critério para seleção de carga, representado por uma ou por um conjunto de MVs, sobre um determinado nó sobrecarregado que deve ser migrada.
- I: Função implementando critério para seleção de nó de processamento com potencial de trabalho disponível para destino de carga representada por uma ou por um conjunto de MVs.
- J: Função implementando critério para seleção de nó de processamento que deverá receber toda a carga, representada por uma ou por um conjunto de MVs, associada a um nó que deva ser desligado.
- M^{on}: Lista dos nós da infraestrutura que se encontram operacionais.
- M^{off}: Lista dos nós da infraestrutura desligados.
- δ_h: Limite superior de carga médio dos nós, acima do qual novos nós devem ser acionados.
- δ'_h: Limite superior de carga de um nó; atingindo este valor, um processo de migração deve ser iniciado.
- δ_l: Limite inferior de carga de um nó; sendo atingido este limite, o nó deve ser desligado, sendo que as MVs que abriga migradas em consequência.

- δ'_l: Limite inferior de carga de um nó, o qual, sendo atingido deve solicitar que MVs sejam migradas a ele.

Para efeitos de simplificação do algoritmo, assume-se que os parâmetro δ_h, δ'_h, δ_l e δ'_l sejam adaptativos, de forma a não induzir a laços infinitos.

Algoritmo 5 Consolidação de máquinas virtuais.

1: **procedure** CONSOLIDAMVS$< G, H, I, J >(M^{on}, M^{off}, \delta_h, \delta'_h, \delta_l, \delta'_l)$
2: $|M^{on}| = \sum_{i=0}^{m} |n_i|, iff\ n_i \in M^{on}$
3: **while** $\frac{|M^{on}|}{M^{on}.size} > \delta_h \wedge \exists n_i \in M^{off}$ **do**
4: $n^{aux} = M^{off}.first$
5: $M^{off}.remove(n^{aux})$
6: $M^{on}.insert(n^{aux})$
7: **end while**
8: **for** $n_i \leftarrow M^{on}.begin$ **to** $M^{on}.end$ **when** $|n_i| < \delta'_l$ **do**
9: $v \leftarrow \delta'_l - |n_i|$
10: $n^{out} \leftarrow G(M^{on}, v)$
11: $|n_{out}| \leftarrow |n_{out}| + v$
12: $|n_i| \leftarrow |n_i| - v$
13: **end for**
14: **for** $n_i \leftarrow M^{on}.begin$ **to** $M^{on}.end$ **when** $|n_{aux}| > \delta'_h$ **do**
15: $v \leftarrow H(n_i)$
16: $n_{aux} \leftarrow I(M^{on}, v)$
17: $|n_i| \leftarrow |n_i| - v$
18: $|n_{aux}| \leftarrow |n_{aux}| + v$
19: **end for**
20: **for** $n_i \leftarrow M^{on}.begin$ **to** $M^{on}.end$ **when** $|n_i| < \delta_l$ **do**
21: $n^{in} \leftarrow J(M^{on}, |n_i|)$
22: $|n^{in}| \leftarrow |n^{in}| + |n_i|$
23: $M^{on}.remove(n_i)$
24: $M^{off}.insert(n_i)$
25: **end for**
26: **end procedure**

O processo de detecção de adição de nós físicos à infraestrutura ocorre entre as linhas 3 e 7 do algoritmo. A necessidade da adição de novos nós considera a carga total do sistema, obtida na linha 2, e a disponibilidade de nós não ativos.

Entre as linhas 8 e 13 do algoritmo, caso exista um nó cuja capacidade de processamento seja inferior a um determinado valor, é disparada uma operação de migração. Um nó origem é selecionado (linha 10) e a carga de trabalho, materializada na forma de uma ou mais MVs, é transferida (linhas 11 e 12).

Entre as linhas 14 e 19 é promovida a migração de MVs de nós considerados sobrecarregados. Para cada nó sobrecarregado, é identificada a MV a ser migrada (linha 15) e o nó destino da migração (linha 16). A migração é então efetivada nas linhas 17 e 18.

Entre as linhas 20 e 24 ocorre o processo de retirada de um nó subutilizado dentre os nós ativos. Caso exista um nó com carga inferior a um determinado limite, um nó ativo é selecionado (linha 21), então, para receber (linha 22) as MVs hospedadas no nó que será desativado. A retirada efetiva do nó ocorre nas linhas 23 e 24.

5.4.4 Critérios de seleção

As funções de seleção G, H, I e J, bem como os valores dados pelos limites δ_h, δ'_h, δ_l e δ'_l oferecem uma visão simplificada, porém suficiente para este trabalho, das questões que envolvem a distribuição de carga entre nós de uma infraestrutura de processamento em nível de sistema. A precisão estará associada a representatividade do índice de carga utilizado e sua frequência de atualização. Sua qualidade também será reflexo da frequência com que o algoritmo for ativado.

5.4.5 Resultado do escalonamento

O resultado do escalonamento em nível de sistema consiste na efetiva execução das cargas computacionais geradas por uma aplicação sobre um conjunto de recursos físicos disponibilizados em uma infraestrutura real. O tempo total de execução permite analisar sua qualidade.

Do ponto de vista tempo, entende-se que, quanto mais próximo o tempo total de execução for do tempo projetado pelo escalonamento em nível aplicativo, melhor qualidade foi apresentada no escalonamento realizado em nível de sistema.

5.5 Considerações Sobre o Capítulo

Aplicações BoT podem fazer uso dos recursos providos por estruturas administradas com o auxílio do OpenStack uma vez que suas tarefas podem ser submetidas no ambiente distribuído fornecido pela nuvem computacional. Por sua vez, o ambiente computacional em nuvem dará suporte às características de alta demanda de processamento e distribuição de carga necessárias para a execução das tarefas pertencentes às aplicações do tipo BoT.

6 EXPERIMENTAÇÃO E RESULTADOS

Este capítulo apresenta uma avaliação, por experimentação, da proposta deste trabalho. De início é apresentada a infraestrutura real utilizada nos experimentos e a metodologia aplicada aos estudos de caso. Na sequência são apresentadas as situações onde os experimentos foram realizados: análise da operabilidade do ambiente de testes (Seção 6.3), impacto para diferentes perfis de usuário do estado de uso da infraestrutura (Seção 6.4) e impacto de BoTs de diferentes classes sobre a infraestrutura (Seção 6.5).

6.1 Ambiente de Testes

Para execução dos experimentos, foi disponibilizado um cluster formado por 10 (dez) nós físicos: um nó controlador e nove nós de computação. Cada nó possui um processador Intel Core i5-2310 @ 2.90GHz com 04 (quatro) núcleos físicos, 16GB de memória RAM e uma interface de rede 1GbE, com exceção do nó controlador que possui duas interfaces de rede.

A versão do OpenStack utilizada para os testes foi a Kilo (OPENSTACK, 2016), com os seguintes módulos instalados: *Keystone* (serviço de identidade), *Glance* (serviço de imagens), *Compute* (serviço de computação), *Nova-network* (serviço de rede) e *Horizon* (interface web para administração). O OpenStack foi configurado para realizar o escalonamento padrão, *Filter*, que atribui a nós físicos as MVs no momento em que estas são criadas considerando a carga dos nós.

Foi implementada uma estratégia de escalonamento em *shell script* baseada no Algoritmo 5. Esta implementação considera um nó sobrecarregado quando for observada uma carga média entre os processadores maior ou igual 80%.

6.2 Metodologia dos Experimentos

Os experimentos realizados permitiram, pelos dados de desempenho coletados, relacionar os resultados do escalonamento realizado em nível aplicativo, conforme proposto na Seção 5.3, com o executado em nível de sistema sobre o OpenStack.

Para tanto, a sequência de passos a seguir foi seguida para execução de cada experimento.

1. Geração de um BoT ou de um conjunto de BoTs;
2. Definição do número de processadores para suportar a execução da aplicação;
3. Submissão da aplicação(ções) gerada(s) sobre o mecanismo de escalonamento aplicativo;
4. Lançamento de máquinas virtuais (MVs) sobre o OpenStack para realizar a carga, cada MV, de cada um dos processadores considerados;
5. Após o final da execução, coleta dos dados informando o tempo total de execução.

Para geração dos BoTs foi adotado o modelo de aplicação apresentado em (IOSUP et al., 2008) e (IOSUP; EPEMA, 2011). A aplicação deste modelo se deu da seguinte forma:

- **Número de quádruplas por BoT:** Conforme a bibliografia, segue a distribuição Weibull, com a mesma semântica dada para o número de tarefas por quádrupla. Nos experimentos conduzidos neste trabalho assumiu-se apenas três valores para este número, indicando BoTs pequenos, médios e grandes. A Tabela 1 apresenta as classes para o número de quádruplas adotadas para realização dos experimentos.
- **Número de tarefas por quádrupla:** Segue a distribuição de Weibull. O parâmetro *escala* indica o número de tarefas com maior probabilidade para ser observado em cada quádrupla descrevendo um conjunto de tarefas de um BoT. O parâmetro *forma* indica como a probabilidade se distribui entre os valores vizinhos à *escala* e se a tendência é de observar maior probabilidade em valores superiores ou inferiores à *escala* (coeficiente de assimetria positivo ou negativo, respectivamente). A grandeza deste valor é dada em unidades de tarefas. A Tabela 2 apresenta as classes para o número de tarefas adotadas para realização dos experimentos.
- **Comprimento de tarefas:** Segue uma distribuição Normal, apresentado em termos de unidades de processamento. O parâmetro *média* indica o tempo médio, entre as quádruplas, requerido para processamento de cada tarefa. O parâmetro *desvio padrão* indica o quanto de dispersão é esperado nos tempos requeridos pelos grupos para processamento de suas tarefas em relação à *média* informada. A Tabela 3 apresenta as classes adotadas para representar o comprimento das tarefas.

- **Carga de cada tarefa:** Segue uma distribuição Normal, apresentada em valores percentuais. O parâmetro *média* indica a média a ser observada entre as tarefas das diferentes quádruplas de um BoT. O parâmetro *desvio padrão* indica o quanto de dispersão é esperado dos custos computacionais de cada grupo de tarefas em relação à *média*. A Tabela 4 apresenta as classes adotadas para representar as cargas de trabalho das tarefas.
- **Tempo entre submissões de quádruplas:** Segue uma distribuição de Weibull. O parâmetro *escala* indica o tempo com maior probabilidade de transcorrer entre a submissão de quádruplas de um mesmo BoT e o parâmetro *forma* indica se distribui entre os valores vizinhos à *escala*. A grandeza deste valor é dada em unidades de tempo. A Tabela 5 apresenta as classes adotadas para representar o tempo entre submissão de quádruplas de BoTs.

Tabela 1 – Classes de aplicações em função do número de quádruplas do BoT.

Classe	**Característica**
Q_5	BoT pequeno, com 5 quádruplas
Q_{10}	BoT médio, com 10 quádruplas
Q_{20}	BoT grande, com 20 quádruplas

Considerando as tabelas 1, 2, 3, 4 e 5, um BoT pode ser descrito por: $Q_{10}, N_{15}^{75}, S_{10\%}^{8}, L_{20\%}^{30}$ e I_{-15}^{15}. Este BoT possui 10 quádruplas, cada uma com 75 tarefas, havendo pouca variação do número de tarefas por quádrupla, sendo maior a probabilidade de quádruplas com um número inferior de tarefas. O comprimento médio das tarefas é de 8 unidades de tempo e o consumo médio de CPU é de 30%, ocorrendo variação de 10% entre o comprimento e de 20% do custo computacional entre tarefas definidas por diferentes quádruplas. O intervalo de tempo entre duas submissões de tuplas tem sua mais alta prioridade em 15 unidades de tempo, havendo alta variabilidade nos valores, tendendo a ocorrência de intervalos de tempo superiores.

É importante salientar dois pontos em relação às classes concebidas. O primeiro ponto diz respeito aos valores utilizados como referência para cada caso. Estes valores foram concebidos tendo como base os trabalhos relacionados, citados no Capítulo 3 e seu reagrupamento em classes tem como único objetivo simplificar o processo de análise e discussão de resultados. Casos considerados extremos, como o citado em (IOSUP; EPEMA, 2011) em que BoTs podem ter mais de 2 mil tarefas e consumir mais do que 100 anos de CPU não foram tratados neste trabalho. O segundo ponto diz respeito à unidade de tempo utilizada para representar o tempo de processamento de uma tarefa e o tempo entre submissões de grupos de tarefas. A unidade utilizada

Tabela 2 – Classes de aplicações em função do número de tarefas.

Classe	Escala	Forma	Característica
$N^s_{/h}$	$[10;50]$	$(0;5]$	Poucas tarefas, alta distribuição da probabilidade em valores próximos à *escala*, tendendo a gerar quádruplas com menos tarefas que a *escala* informada.
$N^s_{h/}$	$[10;50]$	$[-5;0)$	Poucas tarefas, alta distribuição da probabilidade em valores próximos à *escala*, tendendo a gerar quádruplas com mais tarefas que a *escala* informada.
$N^s_{/m}$	$[10;50]$	$(5;20]$	Poucas tarefas, média distribuição da probabilidade em valores próximos à *escala*, tendendo a gerar quádruplas com menos tarefas que a *escala* informada.
$N^s_{m/}$	$[10;50]$	$[-20;-5)$	Poucas tarefas, média distribuição da probabilidade em valores próximos à *escala*, tendendo a gerar quádruplas com mais tarefas que a *escala* informada.
$N^s_{/c}$	$[10;50]$	$(20;50]$	Poucas tarefas, alta concentração da probabilidade em valores próximos à *escala*, tendendo a gerar quádruplas com menos tarefas que a *escala* informada.
$N^s_{c/}$	$[10;50]$	$[-50;-20)$	Poucas tarefas, alta concentração da probabilidade em valores próximos à *escala*, tendendo a gerar quádruplas com mais tarefas que a *escala* informada.
$N^m_{/h}$	$[51;100]$	$(0;5]$	Idem à $N^s_{/h}$, mas com número médio de tarefas.
$N^m_{h/}$	$[51;100]$	$[-5;0)$	Idem à $N^s_{h/}$, mas com número médio de tarefas.
$N^m_{/m}$	$[51;100]$	$(5;20]$	Idem à $N^s_{/m}$, mas com número médio de tarefas.
$N^m_{m/}$	$[51;100]$	$[-20;-5)$	Idem à $N^s_{m/}$, mas com número médio de tarefas.
$N^m_{/c}$	$[51;100]$	$(20;50]$	Idem à $N^s_{/c}$, mas com número médio de tarefas.
$N^m_{c/}$	$[51;100]$	$[-50;-20)$	Idem à $N^s_{c/}$, mas com número médio de tarefas.
$N^l_{/h}$	$[101;200]$	$(0;5]$	Idem à $N^s_{/h}$, mas com muitas tarefas.
$N^l_{h/}$	$[101;200]$	$[-5;0)$	Idem à $N^s_{h/}$, mas com muitas tarefas.
$N^l_{/m}$	$[101;200]$	$(5;20]$	Idem à $N^s_{/m}$, mas com muitas tarefas.
$N^l_{m/}$	$[101;200]$	$[-20;-5)$	Idem à $N^s_{m/}$, mas com muitas tarefas.
$N^l_{/c}$	$[101;200]$	$(20;50]$	Idem à $N^s_{/c}$, mas com muitas tarefas.
$N^l_{c/}$	$[101;200]$	$[-50;-20)$	Idem à $N^s_{c/}$, mas com muitas tarefas.

Tabela 3 – Classes de aplicações em função do tempo de processamento das tarefas.

Classe	Média	Desvio Padrão	Característica
S_c^s	(0;3]	5%	Tarefas curtas, com pouca variação de tamanho entre tarefas de diferentes grupos.
S_m^s	(0;3]	10%	Tarefas curtas, com média variação de tamanho entre tarefas de diferentes grupos.
S_h^s	(0;3]	20%	Tarefas curtas, com muita variação de tamanho entre tarefas de diferentes grupos.
S_c^m	(3;10]	5%	Idem à S_c^s, com tarefas de comprimento médio.
S_m^m	(3;10]	10%	Idem à S_m^s, com tarefas de comprimento médio.
S_h^m	(3;10]	20%	Idem à S_h^s, com tarefas de comprimento médio.
S_c^m	(10;20]	5%	Idem à S_c^s, em um BoT com muitas tarefas.
S_m^m	(10;20]	10%	Idem à S_m^s, em um BoT com muitas tarefas.
S_h^m	(10;20]	20%	Idem à S_h^s, em um BoT com muitas tarefas.

representa um passo na evolução da execução da aplicação, podendo ser adequado proporcionalmente ao tempo real adotado[1].

Tabela 4 – Classes de aplicações em função da carga computacional das tarefas.

Classe	Média	Desvio Padrão	Característica
L_c^s	[10;50]	5%	Tarefas com percentual de ocupação de CPU baixo e pouca variação entre si.
L_m^s	[10;50]	10%	Tarefas com percentual de ocupação de CPU baixo e média variação entre si.
L_h^s	[10;50]	20%	Tarefas com percentual de ocupação de CPU baixo e alta variação entre si.
L_c^m	[51;80]	5%	Idem à L_c^s, mas com carga de ocupação média da CPU.
L_m^m	[51;80]	10%	Idem à L_m^s, mas com carga de ocupação média da CPU.
L_h^m	[51;80]	20%	Idem à L_h^s, mas com carga de ocupação média da CPU.
L_c^m	[81;100]	5%	Idem à L_c^s, mas com carga de ocupação alta da CPU.
L_m^m	[81;100]	10%	Idem à L_m^s, mas com carga de ocupação alta da CPU.
L_h^m	[81;100]	20%	Idem à L_h^s, mas com carga de ocupação alta da CPU.

Uma vez gerado o BoT, ou uma coleção de BoTs, para avaliação, é computado o tempo necessário para sua execução em uma arquitetura não contigenciada. Neste caso, o tempo de execução reflete a exploração de todo o paralelismo descrito pelo conjunto de tarefas em uma nuvem com número suficiente de processadores para suportá-lo.

Em seguida, o conjunto de tarefas é submetido ao escalonamento aplicativo. Como entrada, este escalonamento aplicativo recebe, além do conjunto de tarefas, o número

[1] Este tempo foi fixado em 300 segundos de processamento neste trabalho.

Tabela 5 – Classes de aplicações em função do tempo entre submissões de tarefas.

Classe	Escala	Forma	Característica
$I^s_{\prime h}$	[0;3]	(0;3]	Alta frequência de chegada de grupos de tarefas, com alta variabilidade no tempo, tendendo a gerar chegadas mais frequentes que a *escala* informada.
$I^s_{h\prime}$	[0;3]	[-3;0)	Alta frequência de chegada de grupos de tarefas, com alta variabilidade no tempo, tendendo a gerar chegadas mais espaçadas que a *escala* informada.
$I^m_{h\prime}$	(3;10]	[7;10]	Idem à $I^s_{\prime h}$, mas com média frequência de chegada de grupos de tarefas.
$I^m_{\prime h}$	(3;10]	[-10;-7]	Idem à $I^s_{h\prime}$, mas com média frequência de chegada de grupos de tarefas.
$I^l_{h\prime}$	(10;20]	[10;20]	Idem à $I^s_{\prime h}$, mas com baixa frequência de chegada de grupos de tarefas.
$I^l_{\prime h}$	(10;20]	[-20;-10]	Idem à $I^s_{h\prime}$, mas com baixa frequência de chegada de grupos de tarefas.

de processadores disponíveis e a política de priorização de alocação de tarefas. O resultado da execução desta etapa provê a consolidação dos custos computacionais das tarefas sobre os processadores disponibilizados.

Lembrando que a busca no *bag* se dá por lote ou de forma cíclica (cf. Figura 10), a priorização de tarefas pertencentes a cada passo pode ser feita segundo os critérios apresentados na Tabela 6. Todos os critérios, exceto None, possuem também a estratégia invertida. Por conta da grande variedade de estratégias de priorização que podem ser consideradas, neste trabalho os dados de desempenho correspondem à priorização das tarefas em função do seu custo, sendo utilizada a estratégia de alocação cíclica dos *jobs*.

De posse dos custos consolidados das tarefas sobre MVs, o OpenStack é inicializado sendo as máquinas virtuais distribuídas, principalmente, de forma cíclica entre os *cores* físicos da infraestrutura. O número de nós desta arquitetura real não é necessariamente igual ao número de MVs considerado, podendo ser maior ou menor.

A carga de cada MV indicada em cada passo é efetivamente produzida sobre uma CPU durante um período de tempo fixo pré-definido. Neste trabalho, estipulou-se que cada passo de execução corresponde a 300 segundos para todos os casos de estudo realizados.

Uma vez submetidas à execução, as máquinas virtuais são manipuladas pelo mecanismo de escalonamento sistema responsável pelo balanceamento de carga nos nós físicos da infraestrutura. A despeito da existência de módulos desenvolvidos

[2]Neste trabalho, o critério de prioridade aplicado a todos os experimentos considera a ordenação dos *jobs* em função de seu custo, sendo prioritário os com menor custo computacional.

Tabela 6 – Critérios para priorização das tarefas

Estratégia	Critério	Descrição
None	Sem critério	A prioridade é dada pela ordem de ocorrência da tarefa no passo.
TaskArrivalTime	Data de lançamento da tarefa	A tarefa criada há mais tempo é prioritária.
Duration	Tempo total de duração da tarefa	A tarefa mais longa é prioritária.
Cost[2]	Consumo de CPU da tarefa	A tarefa com menor custo computacional é prioritária.
Completed	Número de passos completados	A tarefa que executou mais passos é prioritária.
Remaining	Número de passos restantes	A tarefa mais próxima do término é prioritária.

para o OpenStack implementando tais funcionalidades, como o Neat (BELOGLAZOV; BUYYA, 2014) e Terracotta[3], optou-se por implementar um mecanismo próprio para ter mais controle sobre sua operação. Na estratégia adotada, que consiste em uma versão do Algoritmo 5, são coletadas as cargas de todos os *cores* físicos disponíveis na arquitetura e considerados dois *thresholds*: nós físicos com carga média igual ou inferior a 20% são considerados ociosos, com carga média igual ou superior à 80% são considerados sobrecarregados. Em um primeiro momento MVs de nós físicos sobrecarregados são migrados para nós com carga intermediária ou ociosa e, então, MVs de nós remanescentes como ociosos são também migrados para nós intermediários.

6.3 Casos Triviais

Para aferir se o ambiente de testes se encontra operacional, dois BoTs foram concebidos de forma a ser possível prever seu resultado, em termos de tempo de execução. Cada um destes BoTs reflete classes de aplicações (tabelas 2 à 5) com características extremas, uma considerando uma aplicação leve sobre uma infraestrutura superdimensionada, outra considerando uma aplicação pesada sobre uma infraestrutura aquém das necessidades. Foram então consideradas duas hipóteses sobre o desempenho de execução destas aplicações, executado o teste correspondente e verificada a validade da hipóteses.

Nos experimentos realizados nesta seção, a carga de processamento de cada máquina virtual foi gerada com o uso de um gerador de cargas sintéticas chamado lookbusy[4]. Este aplicativo permitiu que cada máquina virtual definisse sua taxa de

[3]Disponível em: <http://terracotta.readthedocs.io>. Acessado em: 27 de junho 2016.
[4]Disponível em: <https://www.devin.com/lookbusy/>. Acessado em: 27 de junho 2016.

ocupação de uma CPU (um core físico) durante um passo global.

6.3.1 Tarefas leves e muitos recursos de processamento

A primeira hipótese considerada é a seguinte:

> **Hipótese 1** *Dado um BoT descrito por tarefas com baixo consumo de CPU, havendo uma quantidade suficiente de recursos físicos de processamento, o tempo total de execução real deve ser igual ao tempo de execução em uma arquitetura não contingenciada.*

Os BoTs gerados para validar esta hipótese estão apresentados na sequência.

H1-1 = $\{[0, 5, 3, 24], [3, 6, 7, 30]\}$
H1-2 = $\{[0, 9, 4, 38], [1, 14, 3, 23], [4, 8, 5, 19]\}$
H1-3 = $\{[0, 11, 5, 18], [2, 4, 8, 15], [5, 6, 4, 29], [6, 4, 2, 25]\}$

Estes BoTs foram escalonados de forma a consolidar as cargas de trabalho das tarefas sobre 10 máquinas virtuais. O lançamento destas máquinas virtuais foi realizado sobre o cluster, estando disponível para processamento todos os 9 nós (36 cores físicos). No estudo de caso realizado, a prioridade das tarefas é dada em função de seu custo computacional: maior o custo, maior a prioridade de execução.

A Tabela 7 apresenta os resultados de desempenho obtidos para comprovar a hipótese. A tabela apresenta o tempo, em minutos[5], estimado para execução dos BoTs considerando tanto o mapeamento cíclico na consolidação de tarefas. Também em minutos é apresentado o tempo de processamento real obtido. Neste caso, o tempo apresentado corresponde a uma média de dez execuções do problema, não tendo sido observado desvio padrão maior do que 5%.

Tabela 7 – Tempos de execução para validar hipótese do Caso Trivial 1 (tempos em minutos).

BoT	Consolidado	Real	Diferença (%)
H1-1	45	52	15,5%
H1-2	75	87	16%
H1-3	55	64	16,4%

Em função dos resultados anotados na Tabela 7, é possível confirmar a Hipótese 1. A diferença entre o tempo projetado e o anotado pela execução real reflete os custos operacionais de gerir as máquinas virtuais (sobrecusto), não tendo sido observada nenhuma migração de MV entre os nós físicos disponíveis.

[5]Lembrando que um passo de execução de um BoT corresponde a 300 segundos na arquitetura real.

6.3.2 Tarefas pesadas e poucos recursos de processamento

A segunda hipótese considerada é a seguinte:

> **Hipótese 2** *Dado um BoT descrito por tarefas com alto consumo de CPU, havendo uma quantidade insuficiente de recursos físicos de processamento, o tempo total de execução real será maior que o tempo de execução em uma arquitetura não contingenciada.*

Foram geradados três BoTs para validar esta hipótese. Estes Bots são dados por:

H2-1 = $\{[0, 5, 3, 84], [3, 6, 7, 90]\}$
H2-2 = $\{[0, 3, 5, 78], [2, 4, 5, 75], [3, 2, 4, 80], [3, 3, 8, 60], [5, 5, 4, 69], [6, 4, 2, 65]\}$
H2-3 = $\{[0, 5, 10, 68], [1, 4, 5, 73], [2, 6, 6, 83], [2, 2, 3, 65], [4, 3, 5, 65]\}$

Os dados para comprovação da validade da segunda hipótese encontram-se na Tabela 8. Nesta tabela são apresentados os tempos de processamento, em minutos, estimados após a consolidação de tarefas em um número suficiente de MVs para ser explorado o máximo de paralelismo de cada BoT. O tempo de execução real foi anotado pela execução destas máquinas virtuais sobre dois nós, apenas, da arquitetura disponível, totalizando oito cores para processamento. O tempo da execução real, apresentado em minutos, corresponde a uma média de dez execuções do problema, não tendo sido observado desvio padrão maior do que 5% e corresponde àquele previsto sobre a arquitetura com menor capacidade daquela desejada.

Tabela 8 – Tempos de execução para validar hipótese do Caso Trivial 2 (tempos em minutos).

BoT	MVs	Consolidado	Real	Diferença (%)	Migrações
H2-1	10	45	55	22,22%	21
H2-2	17	50	67	34%	11
H2-3	24	45	83	84,44%	3

Em função dos resultados de desempenho anotados na Tabela 8, é possível confirmar a Hipótese 2. Pela Hipótese 1 foi possível identificar o sobrecusto da operação do ambiente. Nos resultados da Tabela 8 verifica-se que o tempo real de execução, além do sobrecusto, inclui os atrasos na execução de *jobs* dos diferentes BoTs gerados. Considerando ainda que existem oito *cores* disponíveis para processamento, observamos que quanto maior as necessidades de máquinas virtuais identificada pela aplicação (coluna MVs na tabela), maior é a proporção de atraso observado.

Na Tabela 8 o número de migrações de MVs também é anotado. Verifica-se que este número é maior quando o número de MVs ideais da aplicação é próximo do número de *cores* disponíveis. Quanto maior o número de MVs necessárias, maior a saturação dos recursos disponíveis, portanto menos migrações ocorrem.

6.4 Impacto em Perfis de Usuário

Nesta seção é avaliado o comportamento da execução de três BoTs distintos sobre uma infraestrutura de nuvem, diferenciados por suas cargas computacionais, conforme a Tabela 9. Neste experimento, a política de mapeamento de *jobs* (cíclica), o critério de prioridade (menor custo), as classes Q, N, S e I são fixas, variando apenas o custo computacional dado por L_2^{10}, L_{10}^{50} e L_{20}^{75}, caracterizando um perfil de BoT com carga leve, mediana e pesada, respectivamente. Nesta tabela também é informado o número de tarefas de cada BoT e o número de MVs necessárias para exploração máxima de paralelismo da respectiva aplicação.

A exemplo do experimento documentado na Seção 6.5, tanto as cargas de uso dos *cores* da infraestrutura, como as cargas de processamento de cada passo global de cada MV é gerado com o uso do aplicativo stress-ng. Os experimentos foram executados primeiro sem utilizar o mecanismo de escalonamento aplicativo e então com o escalonamento aplicativo ativado. Estes resultados estão apresentados, respectivamente, nas tabelas 10 e 11. Nestas tabelas é apresentado o tempo previsto para execução de cada BoT pelo mapeamento cíclico e a diferença, percentual, do tempo previsto para execução para o tempo real observado.

Tabela 9 – Configuração de BoTs com diferentes perfis de carga de processamento: Leve, Mediano e Pesado.

BoT	**Classe**					**# Tarefas**	**MVs**
	Q	N	S	L	I		
Leve	Q_{10}	N_4^{10}	S_1^3	L_2^{10}	I_1^2	74	3
Mediano	Q_{10}	N_4^{10}	S_1^3	L_{10}^{50}	I_1^2	74	24
Pesado	Q_{10}	N_4^{10}	S_1^3	L_{20}^{75}	I_1^2	74	24

Pelos dados de desempenho observados nas tabela 10 e 11, observamos que a estratégia de escalonamento sistema não teve influência quando o experimento não saturou o ambiente, no caso, quando o cluster encontrava-se ocioso e para o BoT de perfil leve. Nos casos intermediários, BoT de perfil mediano em infraestrutura estressada e BoT pesado sobre arquitetura à 50%, sua influência foi negativa. Já no caso em que houve ocupação extrema, BoT de perfil pesado sobre infraestrutura à 75%, houve ganho em balancear a carga em nível sistema.

A título de informação, a Tabela 12 apresenta o número de migrações registradas durante a execução dos BoTs. Pode-se observar que o número de operações de migração observados refletem os tempos de execução anotados.

Tabela 10 – Tempo real, em minutos, de execução de BoTs com diferentes perfis sobre a infraestrutura disponível com uso dedicado, considerando diferentes taxas de sobrecarga da infraestrutura.

Perfil	**Consolidado**	**Ocupação da Infraestrutura**			**Diferença**		
		Ocioso	**50%**	**75%**	**Ocioso**	**50%**	**75%**
Leve	80	93	86	86	16,25%	7,5%	7,5%
Mediano	65	75	74	78	15,4%	13,8%	20%
Pesado	75	83	89	96	10,7%	18,7%	28%

Tabela 11 – Tempo real, em minutos, de execução de BoTs com diferentes perfis sobre a infraestrutura disponível sujeitas a cargas externas, considerando diferentes taxas de sobrecarga da infraestrutura e executando escalonamento sistema.

Perfil	**Consolidado**	**Ocupação da Infraestrutura**			**Diferença**		
		Ocioso	**50%**	**75%**	**Ocioso**	**50%**	**75%**
Leve	80	92	86	86	15%	7,5%	7,5%
Mediano	65	74	77	86	13,8%	18,5%	32,3%
Pesado	75	84	94	90	12%	25,3%	20%

Tabela 12 – Número de migrações de MVs durante a execução de BoTs de diferentes perfis.

Perfil	**Ocupação da Infraestrutura**		
	Ocioso	**50%**	**75%**
Leve	0	0	0
Mediano	51	97	182
Pesado	49	152	55

6.5 Uso em Produção

Nesta seção é avaliado o uso da infraestrutura de teste simulando um ambiente de produção. Para tanto foram concebidos dez conjuntos de BoTs, identificados por UP-1 a UP-10, representando, cada, um grupo de BoTs reais submetidos de forma sobreposta no tempo. Tais conjuntos de BoTs foram concebidos de acordo com o modelo apresentado em (IOSUP et al., 2008) e suas características encontram-se descritas na Tabela 13.

Tabela 13 – Classes de diferentes BoTs gerados para uso em produção.

UP	# Quádruplas	# Tarefas/quádruplas	Duração	Custo	Intervalo
UP-1	Q_5	N_2^4	S_1^3	L_2^{10}	I_1^2
UP-2	Q_5	N_3^4	S_1^5	L_5^{25}	I_1^2
UP-3	Q_5	N_2^6	S_1^3	L_{10}^{50}	I_2^4
UP-4	Q_5	N_3^6	S_1^5	L_{10}^{50}	I_2^4
UP-5	Q_{10}	N_2^4	S_1^3	L_2^{10}	I_1^2
UP-6	Q_{10}	N_3^4	S_1^5	L_5^{25}	I_1^2
UP-7	Q_{10}	N_2^6	S_1^3	L_{10}^{50}	I_2^4
UP-8	Q_{10}	N_3^6	S_1^5	L_{10}^{50}	I_2^4
UP-9	Q_{20}	N_2^4	S_1^3	L_2^{10}	I_1^2
UP-10	Q_{20}	N_3^6	S_1^5	L_{10}^{50}	I_2^4

Para gerar as cargas consumidas a cada passo global, cada uma das máquinas virtuais utiliza o aplicativo stress-ng[6]. Nas máquinas virtuais, o aplicativo foi configurado para executar o número de operações[7] suficientes para manter ocupado um core físico durante um passo global (5 minutos, nos experimentos) considerando a taxa de ocupação determinada para a MV naquele passo[8].

O detalhamento dos BoTs descritos na Tabela 13 não é apresentado, uma vez que estes são gerados de forma pseudo-aleatória, respeitando as diferentes distribuições identificadas para cada classe. Também de forma pseudo-aleatória foi gerada a ordem de chegada destes BoTs, bem como o intervalo entre eles, sendo este intervalo determinado por I_4^8. Algumas características dos BoTs gerados, notadamente o número de tarefas a data de chegada e a previsão de término dada pelo processo de consolidação, são apresentadas na Tabela 14. Nesta tabela, a ordem de apresentação dos BoTs é determinada pela data de chegada destes. Também é informado o número de

[6]Disponível em: `<http://kernel.ubuntu.com/~cking/stress-ng/>`. Acessado em: 30 junho 2016.

[7]Bogo operações.

[8]Na calibragem deste experimento, foi aferido que, na arquitetura disponível, 65 mil bogo operações são executas em 5 minutos (aproximadamente) em um core a 100% de sua capacidade. Portanto, se a taxa de utilização do core for estabelecido em 50%, 32,5 mil bogo operações serão executadas nos mesmos 5 minutos.

MVs utilizado para a consolidação das tarefas de cada BoT. Foi considerado que para as aplicações descritas por BoTs das classes Q_5, Q_{10} e Q_{20} executam em, respectivamente, 4, 8 e 10 MVs. A coluna Previsão de Término desta tabela, portanto, informa a data, em minutos, prevista para término da execução pelo escalonamento aplicativo de cada BoT.

Tabela 14 – Número de tarefas e projeção dos tempos, em minutos, de execução de BoTs em produção considerando diferente número de MVs.

BoT	# Tarefas	Data de Chegada	# MVs	Previsão de Término	Término Real	Diferença (%)
UP-10	98	0	10	355	394	10,99%
UP-8	41	65	8	240	262	9,17%
UP-9	56	110	10	215	225	4,65%
UP-5	23	165	8	225	231	2,67%
UP-3	10	200	4	245	252	2,86%
UP-7	39	230	8	345	359	4,06%
UP-1	8	270	4	275	297	8%
UP-6	26	360	8	460	472	2,61%
UP-4	16	400	4	480	489	1,86%
UP-2	10	530	4	570	576	1,05%

Na Tabela 14, a coluna Término Real informa a data, em minutos, anotada ao final da execução de cada BoT. A diferença entre a data de chegada do BoT e a sua data de término real reflete a passagem de tempo ocorrida durante execução. A coluna Diferença destaca o percentual de atraso da execução real daquela esperada pela etapa de consolidação. Neste experimento fixou-se os seguintes parâmetros: política de mapeamento cíclica e prioridade de tarefas pelo custo (menor custo, maior prioridade). O escalonamento sistema foi ativado uma vez a cada passo, migrando MVs de nós sobrecarregadas, com nível de ocupação dos processadores médio igual ou superior à 80%, para nós com índices médios de ocupação inferiores.

Ainda na Tabela 14, destaca-se que a data real de término anotado para o UP-2, o minuto 576, representa a data em que todas as aplicações submetidas foram encerradas. A coluna que representa a diferença entre as datas com a expectativa de término das execuções e o término efetivo merece uma análise mais detalhada. Embora um maior número de experimentos pudesse auxiliar a fundamentar o estudo das questões envolventes, pode-se dizer que o número de máquinas virtuais definidas para cada classe de problema foi suficiente para atender os requisitos de execução a contento na infraestrutura disponível. Constata-se este fato pois a diferença não atingiu os índices anotados quando da hipótese da submissão de BoTs em infraestruturas oferecendo menos recursos que os demandados (conforme Hipótese 2).

Um novo experimento, então, foi concebido para avaliar o comportamento do mesmo conjunto de BoTs quando submetido a uma infraestrutura contingenciada e sujeita a cargas externas. Para tanto, todos os BoTs descritos (UP-1 a UP-10) foram agrupados em um único BoT e submetidos à consolidação conjunta, empregando as mesmas políticas do experimento anterior, considerando 16, 20 e 36 MVs. As cargas geradas foram executadas sobre a infraestrutura completa (9 nós, 36 *cores*), considerando diferentes níveis de ocupação com cargas externas (ociosa, 50 e 75%), com escalonamento sistema ativado. As cargas externas (sobrecargas) foram lançadas diretamente sobre os *cores* físicos com o aplicativo stress-ng. Os tempos de execução anotados para execução deste BoT único nas diferentes configurações de consolidação e de infraestrutura são apresentados na Tabela 15.

Tabela 15 – Tempo real, em minutos, de execução de BoTs em produção considerando diferente número de MVs em uma infraestrutura não dedicada.

	16 MVs	**20 MVs**	**36 MVs**
Sem sobrecarga na infraestrutura	883	733	553
Infraestrutura com sobrecarga de 50%	819	684	527
Infraestrutura com sobrecarga de 75%	825	691	540

Observando a Tabela 15, verificamos que o tempo de execução total deste BoT único quando MVs = 36 é inferior aquele observado para o término do UP-2, cuja data é o minuto 576, no experimento anterior. Uma vez que o número de MVs disponíveis para a consolidação neste experimento é maior que o empregado no experimento anterior (4, 8 ou 10 MVs conforme a classe), este comportamento reflete uma melhor distribuição do trabalho, ao mesmo tempo em que a consolidação das tarefas mostra-se resiliente às cargas externas. E possível ainda verificar que a execução do BoT único sobre a arquitetura ociosa resultou em pior desempenho quando comparado aos casos onde foi considerada a existência de cargas externas ao BoT na infraestrutura. Atribuímos este fato ao mecanismo de escalonamento *Filter* aplicado pelo OpenStack. Para chegar a tal conclusão, foi considerado que:

- O mecanismo de carga sintética de 50 e 75%, gerado pelo stress-ng, foi lançado antes do lançamento.
- Em cada nó há 4 *cores* disponíveis.
- Quando da distribuição das MVs sobre os nós com a infraestrutura ociosa, cada nó recebeu 4 MVs para executar, concentrando as MVs da aplicação sobre menos nós físicos.
- Quando da distribuição das MVs sobre os nós com a infraestrutura ocupada, a distribuição de carga realizada pelo *Filter* considerou as carga atuais dos nós

físicos. O resultado é que as MVs responsáveis pela execução da aplicação foi distribuída em um número maior de nós físicos.

O impacto desta observação é refletido no aumento de máquinas virtuais utilizadas. Quanto maior o número de máquinas empregada pela aplicação, melhor distribuição é observada, portanto, um melhor balanceamento de carga é obtido.

6.6 Síntese dos Resultados

Neste capítulo foi realizado um conjunto de experimentos para avaliar o impacto da estratégia de escalonamento aplicativo proposta, denominada Consolidação de Tarefas, para aplicações do tipo *Bag of Tasks*. Os experimentos foram realizados sobre uma infraestrutura privada de nuvem, composta por dez nós, sendo um nó controlador e os demais para processamento. Esta infraestrutura física foi gerenciada por OpenStack, com apoio de um mecanismo de escalonamento sistema externo rudimentar, construído para efeitos de validação da proposta. Devido ao tempo necessário para o processamento de cada um dos casos de teste, o número de repetições realizadas foi limitado, sendo de apenas uma execução para a avaliação dos BoTs em produção (Seção 6.5).

Considerando a grande variedade de experimentos ao qual a proposta poderia ser submetida e ao tempo necessário para processar alguns dos experimentos, foram tomadas duas medidas voltadas a permitir um conjunto mínimo de experimentos a serem realizados. Em um primeiro momento, delimitou-se dois parâmetros: em todos experimentos, a consolidação das tarefas considerada foi aquela realizada utilizando o mecanismo de mapeamento cíclico e priorizando as tarefas, entre si, pelo critério de custo, sendo as tarefas de menor custo prioritárias. Então, em um segundo momento, foram identificadas classes para descrever aplicações do tipo *Bag of Tasks*. As diferentes classes propostas, relacionadas ao número de quádruplas (Q), número de tarefas por quádruplas (N), duração (S) e custo das tarefas (L) e ao intervalo entre submissões de quádruplas (I), embora concebidas para refletir os diferentes modelos de distribuição apresentados em (IOSUP; EPEMA, 2011), foram descritas considerando valores arbitrários suficientes para comparação entre BoTs neste trabalho.

Pela análise dos desempenhos apresentada, foi possível comprovar que a estratégia proposta permite dimensionar a utilização de uma arquitetura física para execução de uma determinada aplicação. Desta forma, comprovou-se a constatação apresentada em (GOKILAVANI; SELVI; UDHAYAKUMAR, 2013), na qual os autores identificam que o tempo total de execução pode ser reduzido agrupando tarefas em uma etapa prévia à execução efetiva.

7 CONCLUSÃO E TRABALHOS FUTUROS

Aplicações do tipo *Bag of Tasks* merecem destaque pelo fato de representarem um número significativo de aplicações que demandam um alto poder de processamento. Este trabalho buscou ampliar os horizontes de aplicação do escalonamento de tarefas aplicativo, com base no modelo proposto.

Com o crescente uso de tecnologias de computação em nuvem tanto em ambientes de pesquisa e desenvolvimento quanto de mercado, os desafios para gerenciamento de recursos nestes ambientes tem se tornado cada dia mais relevantes para os administradores de rede dos provedores de serviço.

Nuvens computacionais podem ser utilizadas como suporte à computação de tarefas por meio de virtualização de hardware, que dá aos usuários a ideia de recursos ilimitados para processamento de suas demandas. Sendo assim, é necessário o uso de tecnologias e modelos capazes de oferecer possibilidades de tratamento para as demandas de uso submetidas às infraestruturas.

Não raro, os recursos físicos de uma nuvem computacional podem não ser suficientes para atender todas as demandas de seus usuários, assim como, também podem haver momentos em que o hardware da nuvem não esteja sendo utilizado em sua plenitude. As técnicas de virtualização permitem a otimização dos recursos, provendo flexibilidade e economia na distribuição das tarefas de processamento. É importante para o usuário de uma nuvem computacional ter a possibilidade de dimensionar, com a maior precisão possível, os recursos que estão sendo contratados e, assim, economizar nos custos operacionais.

O escalonamento de tarefas em uma nuvem computacional deve ser capaz de agendar recursos para que as máquinas virtuais, que carregam a demanda de processamento dos usuários, possam evoluir de modo satisfatório, atendendo às solicitações iniciais de processamento.

Com o cenário proposto neste trabalho, no qual a especificação de aplicações do tipo *Bag of Tasks* e a quantidade de máquinas virtuais necessárias para a execução destas aplicações servem de entrada para o escalonamento aplicativo desenvolvido, foi proposta uma estratégia de consolidação de tarefas para a qual se espera como

saída a distribuição da carga computacional prevista para a execução dos BoTs nas máquinas virtuais, no decorrer do tempo.

O modelo para aplicações BoT proposto neste trabalho é descrito por um conjunto de quádruplas que possuem os elementos: instante de tempo no qual o conjunto de tarefas chega para processamento, o tempo de duração do grupo de tarefas, a quantidade de tarefas descritas por quádrupla correspondente e a taxa de utilização de CPU para cada quádrupla. Para o conjunto de quádruplas, as informações de tempo assumem um número ilimitado de processadores e um ambiente livre de contenção.

Dentre as etapas do modelo de escalonamento proposto, a consolidação de tarefas realiza o mapeamento das tarefas definidas em uma aplicação BoT levando em consideração um conjunto finito de recursos de processamento representado por um número finito de máquinas virtuais.

O escalonamento aplicativo trata da discretização da carga computacional de cada um dos processadores disponibilizados para computação da aplicação e é assumido que estes processadores executarão a carga computacional especificada em cada um de seus passos de forma independente. Quanto mais próximo o tempo total de execução for do tempo projetado pelo escalonamento em nível aplicativo, melhor qualidade foi apresentada no escalonamento realizado em nível de sistema.

O escalonamento em nível de sistema consiste na efetiva execução das cargas computacionais geradas por uma aplicação sobre um conjunto de recursos físicos disponibilizados em uma infraestrutura real.

Para avaliação do modelo de escalonamento proposto neste trabalho, foram realizados experimentos em uma infraestrutura real, constituída por um *cluster*, formado por dez nós físicos. Neste *cluster* foram submetidas cargas de trabalho produzidas a partir de apicações BoT, geradas com base no modelo de aplicação sugerido e executadas para coleta de informação de tempo total de execução.

A aferição do ambiente de testes se deu com a concepção de duas aplicações BoT, com resultado previsível em termos de tempo de execução: Hipótese 1, com tarefas leves e muitos recursos de processamento; Hipótese 2, com tarefas pesadas e poucos recursos de processamento. Nos dois casos, as hipóteses foram confirmadas. Para avaliação da simulação de um ambiente de produção foram concebidos dez conjuntos de BoTs, e submetidos de forma sobreposta no tempo. Neste estudo de caso foi possível verificar que a concentração de máquinas virtuais sobre nós físicos, mesmo que observando o limite de uma MV por core físico, é prejudicado em termos de desempenho

Por fim, foram executados testes para avaliar o comportamento da execução de três aplicações BoT distintas, variando apenas o custo computacional. O objetivo foi o de simular diferentes perfis de usuários, submetendo BoTs com cargas de processamento leve, mediana e pesada. Ao final dos experimentos, verificou-se que o escalo-

namento sistema não influenciou o ambiente quando este não estava sobrecarregado, no caso, perfil leve. Para o caso BoT mediano com sistema a 75% de carga e BoT pesado com sistema a 50% de carga, o escalonador teve influência negativa. Para o caso BoT pesado com sistema a 75% de carga, obteve-se ganho no balanceamento de carga em nível de sistema.

Como continuidade do trabalho, deve-se estender a quantidade de experimentos de forma a permitir a análise de diferentes políticas de priorização de tarefas. Também deve ser considerado avaliar o uso de alguma ferramenta de simulação, como SimGrid (CASANOVA, 2001) ou CloudSim (CALHEIROS et al., 2011). Embora o presente trabalho tenha como um de seus ponto fortes a experimentação em uma infraestrutura real, o limite dos experimentos se dá na proporção dos recursos disponíveis. Projeta-se, portanto, a especificação de uma série de experimentos com vistas a determinar a escalabilidade da proposta de escalonamento aplicativo sobre padrões de aplicações, primeiro avaliando seu comportamento em escalas suportadas pela infraestrutura real e então especificando um modelo para análise apoiada em simulação.

Em relação à estratégia de escalonamento, sua evolução se dará pela incorporação, no modelo especificado, do tratamento de custos de comunicação e da possível heterogeneidade de infraestruturas em termos de capacidades de processamento e/ou memória dos nós de processamento. Por fim, considera-se a extensão da propostas a outros modelos de aplicações, como a BSP (VALIANT, 1990), representativo de uma importante classe de aplicações e bastante utilizado para modelar programas paralelos em diferentes arquiteturas paralelas com desempenho garantido (GOLDCHLEGER et al., 2005; CAMARGO et al., 2006).

REFERÊNCIAS

AL NUAIMI, K.; MOHAMED, N.; AL NUAIMI, M.; AL-JAROODI, J. A survey of load balancing in Cloud Computing: Challenges and algorithms. **Proceedings - IEEE 2nd Symposium on Network Cloud Computing and Applications, NCCA 2012**, [S.l.], p.137–142, Dec. 2012.

ANNETTE, J. R.; BANU, W. A.; SHRIRAM. A Taxonomy and Survey of Scheduling Algorithms in Cloud: Based on task dependency. **International Journal of Computer Applications**, [S.l.], v.82, n.15, p.20–26, Nov. 2013.

BELOGLAZOV, A.; BUYYA, R. OpenStack Neat: A framework for dynamic and energy-efficient consolidation of virtual machines in OpenStack clouds. **Concurrency Computation Practice and Experience**, [S.l.], p.n/a–n/a, June 2014.

CALHEIROS, R. N.; RANJAN, R.; BELOGLAZOV, A.; DE ROSE, C. A. F.; BUYYA, R. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. **Softw. Pract. Exper.**, New York, NY, USA, v.41, n.1, p.23–50, Jan. 2011.

CALVEY, J. **Cloud Computing principles and paradigms**. Disponivel em: <http://bodhost.com/blog/cloud-computing-principles-and-paradigms/>. Accesso em: 18 fevereiro 2015.

CAMARGO, R. Y. de; GOLDCHLEGER, A.; KON, F.; GOLDMAN, A. Checkpointing BSP parallel applications on the InteGrade Grid middleware. **Concurrency and Computation: Practice and Experience**, [S.l.], v.18, n.6, p.567–579, 2006.

CASANOVA, H. Simgrid: a toolkit for the simulation of application scheduling. In: CLUSTER COMPUTING AND THE GRID, 2001. PROCEEDINGS. FIRST IEEE/ACM INTERNATIONAL SYMPOSIUM ON, 2001. **Anais...** [S.l.: s.n.], 2001. p.430–437.

CASAVANT, T. L.; KUHL, J. G. Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. **IEEE Transactions on Software Engineering**, [S.l.], v.14, n.2, p.141–154, 1988.

CIRNE, W.; BRASILEIRO, F.; SAUVÉ, J. Grid computing for bag of tasks applications. **In Proc. of the 3rd IFIP . . .** , [S.l.], 2003.

FEITELSON, D. **A Survey of Scheduling in Multiprogrammed Parallel Systems**. [S.l.]: IBM T.J. Watson Research Center, 1994. (Research report).

GOEL, H.; CHAMOLI, N. Job Scheduling Algorithms in Cloud Computing : A Survey. **International Journal of Computer Applications**, [S.l.], v.95, n.23, p.19–22, 2014.

GOKILAVANI, M.; SELVI, S.; UDHAYAKUMAR. A Survey on Resource Allocation and Task Scheduling Algorithms in Cloud Environment. **International Journal of Engineering and Innovative Technology (IJEIT)**, [S.l.], v.3, n.4, Oct. 2013.

GOLDCHLEGER, A.; GOLDMAN, A.; HAYASHIDA, U.; KON, F. The implementation of the BSP parallel computing model on the InteGrade Grid middleware. In: MIDDLEWARE FOR GRID COMPUTING, MGC 2005, GRENOBLE, FRANCE, NOVEMBER 28 - DECEMBER 2, 2005, 3., 2005. **Proceedings. . .** [S.l.: s.n.], 2005. p.5:1–5:6.

GUTIERREZ-GARCIA, J. O.; SIM, K. M. A Family of Heuristics for Agent-based Elastic Cloud Bag-of-tasks Concurrent Scheduling. **Future Gener. Comput. Syst.**, Amsterdam, The Netherlands, The Netherlands, v.29, n.7, p.1682–1699, Sept. 2013.

HU, B.; YU, H. Research of scheduling strategy on openstack. **Proceedings - 2013 International Conference on Cloud Computing and Big Data, CLOUDCOM-ASIA 2013**, [S.l.], p.191–196, Dec. 2013.

HUANLE, Y.; WEIFENG, S.; TINGTING, B. An OpenStack-Based Resource Optimization Scheduling Framework. **2013 Sixth International Symposium on Computational Intelligence and Design**, [S.l.], p.261–264, Oct. 2013.

IOSUP, A.; EPEMA, D. Grid Computing Workloads. **IEEE Internet Computing**, Los Alamitos, CA, USA, v.15, n.2, p.19–26, 2011.

IOSUP, A.; SONMEZ, O.; ANOEP, S.; EPEMA, D. The performance of bags-of-tasks in large-scale distributed systems. **Proceedings of the 17th international symposium on High performance distributed computing - HPDC '08**, New York, New York, USA, p.97, 2008.

LITVINSKI, O.; GHERBI, A. Openstack Scheduler Evaluation using Design of Experiment Approach. **16th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2013)**, [S.l.], p.1–7, June 2013.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology. **Nist Special Publication**, [S.l.], v.145, p.7, 2011.

MOHAMMADI, F.; JAMALI, S.; BEKRAVI, M. Survey on Job Scheduling algorithms in Cloud Computing. **International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)**, [S.l.], v.3, n.2, p.151–154, 2014.

NETTO, M. A. S.; BUYYA, R. Offer-based Scheduling of Deadline-constrained Bag-of-Tasks Applications for Utility Computing Systems. In: IEEE INTERNATIONAL SYMPOSIUM ON PARALLEL&DISTRIBUTED PROCESSING, 2009., 2009, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2009. p.1–11. (IPDPS '09).

OPENSTACK. **OpenStack: The Open Source Cloud Operating System**. Disponivel em: <http://www.openstack.org/>. Accesso em: 18 fevereiro 2015.

OPENSTACK. **OpenStack Installation Guide for Ubuntu 14.04**. Disponivel em: <http://docs.openstack.org/kilo/install-guide/install/apt/content/>. Accesso em: 03 fevereiro 2016.

OPRESCU, A.-M.; KIELMANN, T. Bag-of-Tasks Scheduling Under Budget Constraints. In: IEEE SECOND INTERNATIONAL CONFERENCE ON CLOUD COMPUTING TECHNOLOGY AND SCIENCE, 2010., 2010, Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2010. p.351–359. (CLOUDCOM '10).

RIMAL, B. P.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. **NCM 2009 - 5th International Joint Conference on INC, IMS, and IDC**, [S.l.], p.44–51, 2009.

SOMASUNDARAM, T. S.; GOVINDARAJAN, K. CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud. **Future Generation Computer Systems**, [S.l.], v.34, p.47–65, May 2014.

SUN, H.; CHEN, S.-p.; JIN, C.; GUO, K. Research and Simulation of Task Scheduling Algorithm in Cloud Computing. **TELKOMNIKA Indonesian Journal of Electrical Engineering**, [S.l.], v.11, n.11, p.6664–6672, 2013.

TIDMARSH, M. Cloud Computing. **http://itknowledgeexchange.techtarget.com/bookworm/book-excerpt-cloud-computing/**, [S.l.], 2013.

VAITHIYANATHAN, V. A Review on Different Scheduling Algorithms in Cloud environment. **International Journal of Applied Engineering Research**, [S.l.], v.8, n.20, p.2689–2692, 2013.

VALIANT, L. G. A Bridging Model for Parallel Computation. **Commun. ACM**, New York, NY, USA, v.33, n.8, p.103–111, Aug. 1990.

WILLEBEEK-LEMAIR, M.; REEVES a.P. Strategies for dynamic load balancing on highly parallel computers. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.], v.4, n.9, 1993.

UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Dissertação

Modelo de Escalonamento Aplicativo para Bag of Tasks em Ambientes de Nuvem Computacional

MAICON ANÇA DOS SANTOS

Pelotas, 2017