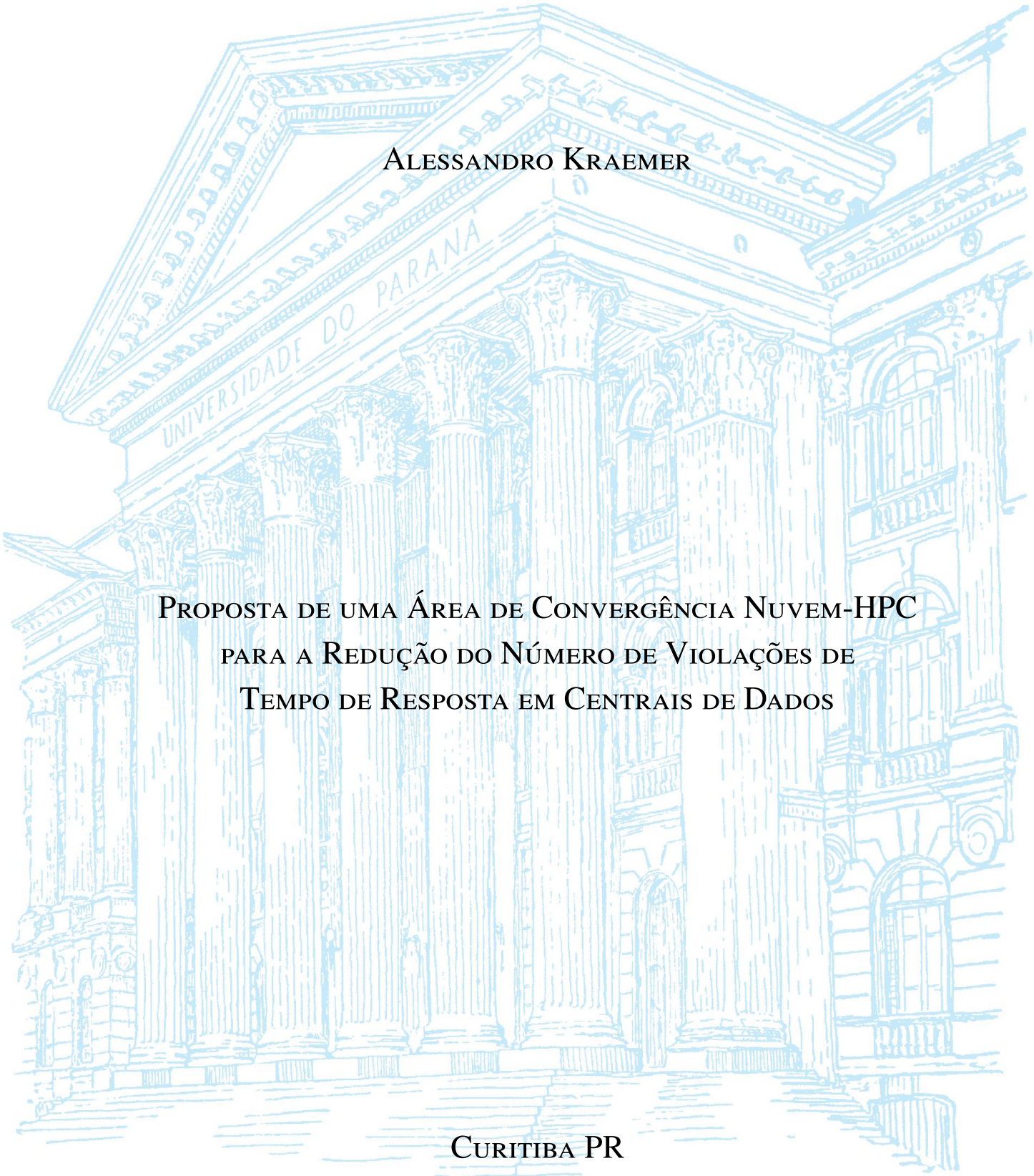
Universidade Federal do Paraná

Alessandro Kraemer



Proposta de uma Área de Convergência Nuvem-HPC para a Redução do Número de Violações de Tempo de Resposta em Centrais de Dados

Curitiba PR
2017

Alessandro Kraemer

Proposta de uma Área de Convergência Nuvem-HPC para a Redução do Número de Violações de Tempo de Resposta em Centrais de Dados

Tese apresentada como requisito parcial à obtenção do grau de Doutor em Informática no Programa de Pós-Graduação em Informática, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Carlos Alberto Maziero.

Curitiba PR
2017

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

K89p

Kraemer, Alessandro
Proposta de uma área de convergência Nuvem-HPC para a redução do número de violações de tempo de resposta em centrais de dados / Alessandro Kraemer. – Curitiba, 2017.
128 f. : il. color. ; 30 cm.

Tese - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2017.

Orientador: Carlos Alberto Maziero .
Bibliografia: p. 108-116.

1. Computação de alto desempenho. 2. Computação em nuvem. 3. Centro de processamento de dados. I. Universidade Federal do Paraná. II.Maziero, Carlos Alberto. III. Título.

CDD: 004.6782

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

MINISTÉRIO DA EDUCAÇÃO
SETOR CIÊNCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da tese de Doutorado de **ALESSANDRO KRAEMER** intitulada: **Uma proposta de área de convergência nuvem-HPC para a redução do número de violações de tempo de resposta em centrais de dados**, após terem inquirido o aluno e realizado a avaliação do trabalho, são de parecer pela sua aprovação no rito de defesa.

A outorga do título de doutor está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 11 de Dezembro de 2017.

CARLOS ALBERTO MAZIERO
Presidente da Banca Examinadora (UFPR)

ALCIDES CALSAVARA
Avaliador Externo (PUC/PR)

BRUNO RICHARD SCHULZE
Avaliador Interno (UFPR)

ANDREY MONTEIRO BRITO
Avaliador Externo (UFCG)

MARCO ANTONIO ZANATA ALVES
Avaliador Interno (UFPR)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

Há aqueles que lutam um dia, e por isso são bons;
há aqueles que lutam muitos dias, e por isso são melhores;
há aqueles que lutam anos, e são muito bons;
porém, há aqueles que lutam toda a vida, esses são imprescindíveis.

Bertolt Brecht

Agradecimentos

Agradeço aos meus pais, por terem me concedido o dom da vida.

Agradeço à minha família, por estar ao meu lado, sobretudo nos momentos de maior dificuldade.

Agradeço à Universidade Tecnológica Federal do Paraná – UTFPR, campus Campo Mourão, por ter investido em minha formação intelectual e profissional.

Agradeço ao Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela oportunidade concedida para cursar o Doutorado.

Agradeço ao prof. Carlos Maziero, pela orientação e acompanhamento de meus estudos e realizações ao longo do doutoramento.

Agradeço aos professores Denis Trystram, do Instituto Politécnico de Grenoble, e Olivier Richard, da Universidade Grenoble-Alpes, pela atenção dispensada durante meus períodos de pesquisa na França.

Finalmente, agradeço aos membros da banca examinadora, pela disposição em analisar e contribuir para a qualidade desta tese.

Resumo

A computação de alto desempenho vem permitindo à ciência avançar rapidamente em muitas áreas do conhecimento. Por sua vez, as tecnologias de computação em nuvem proporcionam o acesso a recursos computacionais a muitos usuários. Ambas as tecnologias proporcionam ambientes de computação de larga escala, embora sejam usadas para propósitos distintos. Enquanto tecnologias de nuvem proporcionam computação diversa ao público como meio de negócio, tecnologias puramente HPC têm objetivo científico e seus usuários são especializados. Em centrais de dados é frequente a implantação separada de plataformas HPC e de nuvem em um mesmo supercomputador. Em geral, cada plataforma gerencia uma fila de entrada de aplicações, tem seu próprio domínio de processadores, memória e rede de comunicação. A nuvem é propícia à migração de aplicações de plataformas HPC, sendo vista como extensão de *hardware* virtual e *software*. Por outro lado, a migração de aplicações de nuvem para plataformas HPC é um tópico muito menos explorado. Não obstante, isso pode ser útil em alguns casos, em particular quando a plataforma HPC tem baixo nível de utilização de recursos e a utilização de recursos na nuvem é alta. Em nuvem, a alta utilização de seus recursos faz com que aplicações de menor prioridade sejam despejadas devido ao escalonamento com *overbooking*. Com base na literatura, identificou-se que o processo de despejo aumenta o número de violações de tempo de resposta das aplicações despejadas. A fim de reduzir o número de violações de tempo de resposta, propõe-se uma área de convergência nuvem-HPC em centrais de dados, abrangendo funções de previsão de violação, estratégias para escalonadores e escalonamento de aplicações. A proposta está formalmente definida, implementada em um simulador e avaliada em diferentes experimentos. Para tal, uma carga de trabalho HPC com 11K aplicações foi extraída do Grid de produção do *Potsdam Institute for Climate Impact Research*. A quantidade e as características das aplicações de nuvem com potencial de despejo foram determinadas com base em documentos públicos do Google. O escalonador HPC proposto foi bem sucedido ao injetar 267K aplicações de nuvem com zero número de violação de tempo de resposta, considerando uma área de convergência com 320 processadores. Os resultados mostram que existe um número significativo de áreas de tempo ocioso no plano de escalonamento HPC, o que também tem sido observado em outros grids. Isso propícia a alocação por *backfilling* de um número elevado de aplicações como as de nuvem consideradas. No cenário com 10 processadores na área de convergência, o pior percentual de violações obtido foi de 0,002%. Não houve impacto relevante sobre o *makespan* da carga de trabalho HPC. Por fim, a estratégia de previsão mostrou-se eficiente com 0,96 de precisão média. Os resultados dos experimentos mostram que esta proposta é factível de ser implantada em centrais de dados e tem capacidade para reduzir o número de violações de tempo de resposta de aplicações de nuvem, com baixo impacto sobre o escalonamento HPC.

Palavras-chave: escalonamento de tarefas, computação de alto desempenho, computação em nuvem, central de dados.

Abstract

Science is enhancing rapidly in many knowledge areas due to high-performance computing. On the other hand, cloud computing technologies provide access to computating resources by many users. Both technologies provide large-scale computing environments, although they are used for different purposes: while cloud technologies provide diverse computing to the public users as a business, pure HPC technologies have a scientific purpose, and their users are specialized. In data centers, HPC and cloud platforms often are separately deployed on the same supercomputer. In general, each platform manages a job input queue, has its processors, memory, and communication network domains. The cloud is suitable to receive jobs migrated from HPC platforms, working as a virtual hardware and software extension to them. However, migrating cloud jobs to HPC platforms is a much less explored topic. Nevertheless, that may be useful in some cases, mainly when the HPC platform has a low resource utilization, and resource utilization in the cloud is high. In the cloud, the high resource utilization causes lower priority jobs eviction, due to overbooked scheduling. Based on the literature, the eviction procedure increases the number of response time violations for evicted applications. A cloud-HPC convergence area in data centers is proposed here, to reduce the number of response time violations. The proposal includes violation prediction functions, strategies for schedulers, and job scheduling. The proposal is formally defined, implemented in a simulator, and evaluated in different experiments. For thata, an HPC workload with 11K jobs collected from the production Grid of the Potsdam Institute for Climate Impact Research was used to evaluate the proposal. The number and characteristics of cloud jobs with dump potential were determined based on Google's public documents. The proposed HPC scheduler obtained success in injecting 267K cloud jobs in the HPC platform, resulting in zero response time violations, considering a convergence area set to 320 processors. The results show that there is a significant number of idle time areas in the HPC scheduling plane, which has also been observed in other grids. That facilitates the allocation by backfilling for a large number of jobs such as the cloud considered. In the scenario set to 10 processors in the convergence area, the worst violation percentage obtained was 0.002%. There was no critical impact on the HPC workload makespan. Finally, the prediction strategy was efficient, resulting the average precision of 0.96. The results of the experiments show that this proposal is feasible to be deployed in data centers and can reduce the number of response time violations for cloud jobs, with low impact on the HPC scheduling.

Keywords: job scheduling, high-performance computing, cloud computing, data center.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

AC	Área de Convergência
ANR	*L'Agence Nationale de la Recherche* - Agência Nacional de Pesquisa
API	*Application Programming Interface* - Interface de Programação de Aplicativos
BATSIM	*Batch Scheduler Simulator* - Simulador de Escalonador em Lote
BL	*Bottom-Left* - Abaixo-Esquerda
BSD	*Berkeley Source Distribution* - Distribuição de Código-fonte de Berkeley
BSP	*Bulk Synchronous Parallel* - Sincronização Paralela em Massa
CARS	*Convergence Area Scheduler* - Escalonador de Área de Convergência
CPU	*Central Processing Unit* - Unidade Central de Processamento
C-SIG	*Cloud Select Industry Group* - Grupo da Industria Selecionado de Nuvem
DAG	*Directed Acyclic Graph* - Grafo Acíclico Direcionado
DBL	*Decreasing Bottom-Left* - Abaixo-Esquerda Decrescente
DVFS	*Dynamic Voltage and Frequency Scaling* - Voltagem Dinâmica e Frequência Escalar
EASY	*Extensible Argonne Scheduling sYstem* - Sistema de Escalonamento Argonne Extensível
FCFS	*First Come - First Served* - Primeiro a Chegar, Primeiro a ser Servido
FIFO	*First in - First Out* - Primeiro a entrar, Primeiro a sair
GP	*Grid Processor* - Processador de Grid
HPC	*High Performance Computing* - Computação de Alto Desempenho
IAAS	*Infrastructure as a Service* - Infraestrutura como um Serviço
IEC	*International Electrotechnical Commission* - Comissão Eletrotécnica Internacional
INRIA	*Institut National de Recherche en Informatique et en Automatique* - Instituto Nacional de Pesquisa em Computação e Automação
ISO	*International Organization for Standardization* - Organização Internacional para Normalização
JSON	*JavaScript Object Notation* - Notação de Objeto *JavaScript*
KVM	*Kernel-based Virtual Machine* - Máquina Virtual baseada em Núcleo
MCSP	*Multiple Cluster Scheduling Problem* - Problema de Escalonamento de Vários Grupos
L-GPL	*Lesser General Public License* - Licença Pública Geral Menor
MCT	*Minimum Completion Time* - Tempo de Completude Mínimo
MLB	*Min-Lower-Bound* - Limite Inferior Mínimo

MOAIS	*prograMming and scheduling design fOr Applications in Interactive Simulation* - Projeto de Escalonamento e Programação para Aplicações em Simulação Interativa
MPI	*Message Passing Interface* - Interface de Passagem de Mensagem
MSG	*Meta SimGrid*
NIST	*National Institute of Standards and Technology* - Instituto Nacional de Padrões e Tecnologia
PAAS	*Platform as a Service* - Plataforma como Serviço
PS	*Parallel Scheduling* - Escalonamento Paralelo
P2P	*Peer-to-Peer* - Ponto-a-Ponto
RAM	*Random Access Memory* - Memória de Acesso Aleatório
RJMS	*Resource and Job Management System* - Sistema Gerenciador de Tarefas e Recursos
SAAS	*Software as a Service* - *Software* como Serviço
SD	*Slowdown* - Tempo Dentro do Sistema
SIMGRID	*Simulation for Grid* - Simulação para Grid
SIMDAG	*Simulation for DAG* - Simulação para DAG
SLA	*Service Level Agreement* - Acordo de Nível de Serviço
SLO	*Service Level Objective* - Objetivo de Nível de Serviço
SWF	*Standard Workload Format* - Formato de Carga de Trabalho Padrão
SMPI	*Simulation for MPI* - Simulação para MPI
SRPT	*Shortest Remaining Processing Time* - Tempo de Processamento Restante mais Curto
SWCT	*Weighted Completion Time* - Tempo de Completude Ponderado
SURF	*Virtual Platform Simulator* - Simulador de Plataforma Virtual
TI	Tecnologia da Informação
VLAN	*Virtual Local Area Network* - Rede Local Virtual
VM	*Virtual Machine* - Máquina Virtual
WSLA	*Web Service Level Agreement* - Acordo de Nível de Serviço Web
WSDL	*Web Service Definition Language* - Linguagem de Definição de Serviço web
WS-agreement	*Web Services agreement* - Acordo de Serviços Web
XBT	*Core ToolBox* - Núcleo das Ferrramentas
XML	*Extensible Markup Language* - Linguagem de Marcação Extensível
YARN	*Yet Another Resource Negotiator* - Ainda Outro Negociador de Recursos

Lista de Símbolos

$\mathcal{SLO}$	conjunto de *tempos de resposta SLO* da Central de Dados
$\mathcal{J}$	conjunto de aplicações J
a_j	momento de chegada da aplicação J_j na Central de Dados
σ_j	momento de inicialização da aplicação J_j
r_j	tempo gasto desde a_j até a inicialização da aplicação em σ_j
wt_j	tempo de processamento da aplicação J_j
h_j	número de processadores requerido pela aplicação J_j
H_j	lista de processadores que inicializarão a aplicação J_j
$SLO_{j,i}$	máximo *tempo de reposta* SLO_i acordado para a aplicação J_j
C_j	momento de completude da aplicação J_j
$\mathcal{L}$	fila FCFS
$\mathcal{P}$	conjunto de processadores P
s_q	velocidade de processamento do processador P_q
Q_q	fila de escalonamento do processador P_q
$\mathcal{D}$	conjunto $\mathcal{P}$ ordenado por tempo de terminação mais curto primeiro
$\mathcal{E}$	conjunto de escalonadores E
A_e	endereço do escalonador E_e na Central de Dados
m	número total de processadores
m_{conv}	número de processadores na *área de convergência*
$\mathcal{W}_{imediato}$	conjunto de *slots* de tempo ocioso de momento corrente
$\mathcal{W}_{futuro}$	conjunto de *slots* de tempo ocioso de momento futuro
$\mathcal{S}$	mapeamento para *backfilling* no *estágio* preliminar do escalonamento proposto
λ	fator de risco de *overbooking*
α	fator de ajuste de tempo de processamento
τ	fator de ajuste de oferta de *tempo de resposta*
β	tempo *mágico* para *backfilling* a partir do momento $t_0^{futuro,e}$, onde é possível realizar *backfilling* de aplicação
γ	área de alocação sem violação de *tempo de resposta*
ζ	momento atual do ciclo de escalonamento
$J_{j,e}$	aplicação J_j no escalonador E_e
$J_j^{'}$	aplicação J_j com atributos que determinam o menor *tempo de resposta*
$F^{\mathcal{W}}(J_{j,e})$	função para determinar a área de alocação (processadores X tempo de processamento) da aplicação $J_{j,e}$
$F^{\gamma}(\mathcal{W}_{futuro,e}, J_{j,e})$	função para determinar a área de tempo ociosa antes que a aplicação $J_{j,e}$ seja violada
$F^{\sigma}(J_{j,e})$	função para previsão do momento de inicialização de aplicação $J_{j,e}$

$F^{C}(J_{j,e})$	função para previsão do momento de completude da aplicação $J_{j,e}$
$F^{J^{'}}(J_j)$	função para previsão do menor *tempo de resposta* para J_j na *área de convergência*
$F^{V}(J_j, J^{'}_{j,e})$	função para previsão de violação
$F^{SLO}(J^{'}_{j,e})$	função para oferta de *tempo de resposta* SLO
$V^{k}(J_j)$	função para verificação da adequabilidade da aplicação J_j no *estágio* de escalonamento k
$P^{k}(J_j)$	função para escalonamento da aplicação J_j no *estágio* k

Sumário

Capítulo 1

Introdução

Desde o surgimento dos supercomputadores, os aglomerados de processadores são vistos como recursos para solução de problemas avançados de computação, tais como aplicações empresariais robustas e aplicações de experimentação científica. O alto desempenho computacional desses recursos para essas aplicações remete ao termo HPC (*High-Performance Computing* - Computação de Alto Desempenho) [Emeras et al., 2013]. Há crescente demanda por recursos computacionais requisitados por essas aplicações, fazendo com que muitas empresas, universidades e governos implantem sua própria infraestrutura de computação.

Desde o início deste milênio, com o avanço da Tecnologia da Informação, populariza-se o uso da computação para diversos fins. Com isso, surge significativo número de aplicações auxiliando tarefas de vários tipos de usuário. Os computadores estão relacionados a todo tipo de tecnologia e os limites de processamento não são mais determinados localmente. O processamento computacional passou a ser também remoto, onde tarefas de aplicações são submetidas para outros computadores em rede. Do ponto de vista dos usuários, toda complexidade desse mecanismo é oculta, ou seja, uma nuvem. No atendimento remoto dessas aplicações estão as chamadas *plataformas de computação em nuvem*, onde existem várias camadas de *software* para habilitar a transparência do mecanismo e tornar isso um modelo de negócio [IBM, 2009].

Devido a essa crescente demanda vinda de diversos tipos de aplicação de nuvem e de HPC, surgiram as chamadas centrais de dados. Uma Central de Dados é formada por aglomerados de processadores onde plataformas de nuvem e de HPC são implantadas separadamente. A fim de gerenciar o sistema de uma maneira central, tem-se também um RJMS (*Resource and Job Management System* - Sistema Gerenciador de Tarefas e Recursos). O RJMS é composto por escalonadores de aplicações e sistemas de monitoramento de recursos. Um escalonador gerencia pelo menos uma fila de aplicações e aplica uma estratégia de escalonamento para determinar onde e quando as aplicações são executadas.

Centrais de dados podem oferecer serviços privados e/ou públicos. Como exemplo de serviço privado, cita-se o Grid'5000 do Governo Francês, cujo objetivo é a experimentação científica realizada por universidades e laboratórios parceiros. Em Centrais de Dados como Google e Amazon existem os dois contextos, o público relacionado com plataformas de nuvem e o privado com objetivo HPC para sistemas internos e experimentação científica.

Algumas aplicações estão sujeitas a cobrança pela Central de Dados, em especial as aplicações de nuvem. Nesse contexto, elabora-se um contrato entre as partes, onde explicitam-se os recursos computacionais que serão disponibilizados e o desempenho das aplicações nesses recursos. Esse contrato é chamado de SLA (*Service Level Agreement* - Acordo de Nível de Serviço) e cada cláusula de recurso é chamada de SLO (*Service Level Objective* - Objetivo de Nível de Serviço). O usuário também tem obrigação contratual, tal como o número máximo

de submissão de aplicações por determinada unidade de tempo. As penalidades por quebra de acordo são diversas. Para exemplificar, quando há quebra de acordo pela aplicação do usuário, cobra-se mais pelo serviço. Quando há quebra de acordo pela Central de Dados, o usuário recebe crédito, podendo submeter mais aplicações que o acordado. Aplicações podem, eventualmente, ter seus acordos violados.

Aplicações submetidas para nuvem ou para HPC possuem algumas características distintas. Na nuvem Google, por exemplo, aplicações dominantes em número correspondem à aplicações curtas em termos de tempo de processamento e número de processadores requisitados [Sheng et al., 2012]. Comparando-se essas aplicações com aplicações não dominantes em número, mas dominantes em uso de recursos, há grande variação de tamanho entre elas, sendo que as primeiras podem levar apenas alguns minutos e as dominantes em uso de recursos podem levar dias até o encerramento. A frequência de submissão dessas aplicações é regular [Sheng et al., 2012]. Essas mesmas características de aplicações de nuvem são observadas em outras Centrais de Dados, tais como Facebook e Microsoft [Zhang et al., 2014]. Por outro lado, aplicações HPC requisitam maior tempo de processamento e recursos e a frequência de submissão é esparsa [Sheng et al., 2012].

Plataformas de computação em nuvem usualmente têm alto nível de utilização de seus recursos se comparadas com a maioria das plataformas HPC. Para exemplificar, a Central de Dados do Google possui nível de utilização médio de 40% de processador e 60% de memória [Reiss et al., 2012], embora objetiva-se ocupar 100% dos processadores disponíveis e 80% da memória principal. Enquanto isso, o nível de utilização em grids HPC pode variar muito, entre 10% e 88% considerando as cargas de trabalho apresentadas em [Feitelson, 2016]. Evita-se expressar um valor médio para grids HPC devido à grande variação entre seus tipos.

O escalonamento de aplicações por *overbooking* é visto como uma razão para o alto nível de utilização de recursos em plataformas de nuvem [Caglar e Gokhale, 2014]. Em *overbooking*, aplicações são escalonadas sobrepondo o uso de recursos, pois é previsto que algumas delas terminem antes do seu tempo de processamento estimado, liberando os recursos para outras aplicações sobrepostas. Essa forma de escalonamento pode levar a uma alta taxa de despejo de aplicações menos prioritárias [Reiss et al., 2012]. As aplicações de menor prioridade despejadas são encerradas prematuramente e podem ser reescalonadas. Cerca de 2,2% dessas aplicações nunca são reescalonadas, podendo ser caracterizadas como violadas [Hemmat e Hafid, 2016]. Em caso de reescalonamento, aumenta-se o tempo total gasto pela aplicação dentro do sistema, o que potencialmente causa violações de tempo de resposta [Tomas e Tordsson, 2013]. O percentual médio de violações SLO em sistema de computação em larga escala é de 0,2% [Hemmat e Hafid, 2016].

O despejo de aplicações de menor prioridade ocorre eventualmente em nuvem devido a garantias vinculadas para as aplicações de maior prioridade. Quando ocorre o despejo de uma aplicação de maior prioridade, em geral realiza-se a migração dela com pouca ou nenhuma perda de dados já processados [Amani e Zamanifar, 2014]. Essa é uma forma de priorizar aplicações de usuários que pagam pelo serviço da Central de Dados. A migração de aplicações de menor prioridade considera que a aplicação deve ser reiniciada, pois o uso de recursos na migração mitiga a relação custo-benefício.

Em contrapartida ao *overbooking* em nuvens, plataformas HPC frequentemente utilizam estratégias de escalonamento em lote [Emeras et al., 2013], pois a observação da execução das aplicações científicas é melhor conduzida quando não há compartilhamento dos recursos. O OAR [Margery et al., 2014], RJMS utilizado no Grid'5000, é um exemplo de escalonador com essa estratégia de escalonamento.

Por meio de observação das características apresentadas por [Sheng et al., 2012], tem-se que plataformas HPC são em geral mais ociosas que plataformas de nuvem. A ociosidade é caracterizada pelos intervalos de tempo esporádicos onde não há aplicação escalonada. Observou-se também que em nuvem existe um conjunto de aplicações que são despejadas e eventualmente são reescalonadas. As aplicações reescalonadas continuam consumindo recursos computacionais e estão mais propícias a violações de tempo de resposta SLO. Por outro lado, existem plataformas HPC com nível de ociosidade considerável e que podem receber aplicações vindas de outras plataformas. Uma integração básica entre nuvem em HPC é possível de ser feita pela submissão de aplicações de uma plataforma para a fila de espera do escalonador da outra plataforma. Porém, o gerenciamento de tempo de resposta não é comumentemente implantado em plataformas HPC, haja visto que HPC não está inserido em modelos de negócio, tais como são os serviços de nuvem.

Neste trabalho propõe-se uma área de convergência nuvem-HPC onde habilita-se a migração de aplicações de nuvens para HPC e o gerenciamento de tempo de resposta de aplicação em plataformas HPC. O tamanho da área pode ser estático ou dinâmico em termos de número de processadores, podendo envolver processadores de partições de nuvem, partições de HPC e processadores fora dessas partições (processadores no domínio do escalonador principal da Central de Dados). Para a habilitação desse mecanismo, propõem-se também funções de previsão de violação de tempo de resposta, estratégias para escalonadores e estratégia de escalonamento nuvem-HPC. Nesse cenário, objetiva-se reduzir o número de violações de tempo de resposta SLO. Não foi encontrado trabalho similar na literatura.

A proposta está formalmente definida, implementada no programa CArS [Kraemer et al., 2017] e validada por meio de integração nos arcabouços BatSim [Dutot et al., 2016] e Simgrid [Margery et al., 2014]. Para experimentação científica foi utilizada uma carga de trabalho HPC de produção com 11K aplicações e nela foram inseridas 267K aplicações de nuvem. As aplicações HPC foram extraídas do histórico de execução do *Potsdam Institute for Climate Impact Research* [Feitelson, 2016] e as aplicações de nuvem foram definidas com base em documentos públicos do Google [Reiss et al., 2012]. Os resultados mostram que o número de violações de tempo de resposta SLO é significativamente baixo na área de convergência, sendo 0,002% no cenário com uma área de convergência com 10 processadores e zero com 320 processadores. A precisão da previsão de violação também mostrou-se satisfatória, com taxa 0,96 (ou 96% de previsões corretas). Esses resultados ocorrem principalmente pelas características das aplicações de nuvem selecionadas e pelos espaços esporádicos de tempo ocioso presentes em plataformas HPC, os quais são utilizados para o escalonamento dessas aplicações.

Este trabalho está organizado em 7 capítulos. No Capítulo 2 apresentam-se fundamentos sobre plataformas de computação em larga escala, tais como são as plataformas implantadas em centrais de dados. Ainda nesse capítulo comparam-se características de plataformas de computação em nuvem e de HPC e detalha-se o conceito de SLA. No Capítulo 3 apresentam-se estratégias de escalonamento, iniciando pela caracterização de aplicação e partição, concluindo com um levantamento sobre desafios em aberto na área. No Capítulo 4 apresenta-se a proposta de área de convergência nuvem-HPC e seus mecanismos. No Capítulo 5 apresentam-se propostas para escalonadores na área de convergência, seguida da descrição de uma estratégia de escalonamento. No Capítulo 6 apresentam-se a validação da proposta, resultados de experimentos e discussões. Finalmente, no Capítulo 7 conclui-se o trabalho.

Capítulo 2

Plataformas de Computação em Larga Escala

Existe uma demanda crescente por processamento e análise de dados em todas as áreas do conhecimento. Essa demanda tem sido atendida pelas plataformas de computação em larga escala, haja visto que elas são caracterizadas por possuirem uma ampla lista de recursos computacionais e capacidade de gerenciamento de aplicações diversas.

Plataformas de computação em larga escala são usualmente implantadas em centrais de dados. Nessas plataformas são executados sistemas de administração de negócios por meio de serviços de *software* em máquinas virtuais; serviços de Internet, como e-mail e páginas web para diversas finalidades; computação de alto desempenho para áreas como biológicas e engenharias, por meio de acesso a arcabouços e suas bibliotecas de programação; processamento de dados para aplicações de experimentação científica da ciência da computação, por meio de instâncias de cenários de sistemas distribuídos, entre outros.

Há um significativo número de trabalhos na literatura apresentando modelos de gerenciamento de aplicações em plataformas de computação. A eficiência desses modelos é determinada por objetivos como a maximização de uso dos recursos computacionais [Yarmolenko e Sakellariou, 2006], a maximização do número de aplicações aceitas [Barquet et al., 2013], entre outros. Essas otimizações são usualmente implantadas nos escalonadores de aplicação das plataformas de computação. Tendo em vista que o processamento de aplicações em plataformas pode ser pago à Central de Dados, essas otimizações são importantes, porque podem reduzir custos dos usuários e otimizar o lucro do provedor [Simarro et al., 2011, Mao e Humphrey, 2011]. Entretanto, muitas vezes o ganho não é mútuo.

O objetivo deste trabalho se situa no contexto de escalonamento de tarefas de aplicações em computação de alto desempenho. Nessa linha de pesquisa há diversos trabalhos discutidos na literatura. Neste capítulo são apresentadas características relativas a esse tema. As seções seguintes delineiam aplicações, arquiteturas e sistemas de escalonamento. Na Seção 2.1 são apresentadas características de centrais de dados, assunto central deste capítulo. Na Seção 2.2 é apresentada uma comparação entre plataformas HPC e de computação em nuvem em centrais de dados. Exemplos de RJMS comerciais são apresentados na Seção 2.3. Por fim, na Seção 2.4 apresenta-se o conceito de Acordo de Nível de Serviço (SLA - *Service Level Agreement*). O entendimento de SLA é importante no contexto deste trabalho, porque por meio dele é que são negociados os limites de tempo de processamento, que têm influência sobre diversas métricas de escalonamento.

2.1 Características de Centrais de Dados

Uma Central de Dados é caracterizada em alto nível como um sistema de computação em larga escala que visa executar uma diversidade de tipos de aplicações [Iglesias et al., 2014]. Centrais de dados podem conter milhares de servidores formando uma complexa rede de aglomerados de computadores [Liu e Cho, 2012b]. O uso e gerenciamento eficiente dos seus recursos computacionais para processamento de aplicações tem sido tratado como tarefa desafiadora pela literatura científica [Reiss et al., 2012] [IBM, 2009]. Aplicações em centrais de dados podem ser muito distintas, sendo grosseiramente divididas em dois grupos:

- **Serviços de nuvem**: este grupo consiste de aplicações distribuídas em larga escala para processamento de e-mail, editoração de documentos *online*, comércio eletrônico, armazenamento de dados, aluguel de máquinas virtuais e outras. Essas aplicações são usualmente executadas em ambientes virtualizados e/ou dentro de *containers*, onde recursos computacionais são disponibilizados conforme a necessidade de cada aplicação;

- **HPC**: este grupo consiste usualmente de aplicações de computação intensiva como processamento de imagens, análise de genomas, simulação numérica e outras similares. Uma aplicação HPC é usualmente executada em ambientes não virtualizados, para evitar a sobrecarga da virtualização e também para que sua execução seja mais previsível para análises científicas.

Centrais de dados frequentemente particionam sua infraestrutura para implantar plataformas separadas de computação em nuvem e de HPC, atendendo assim demandas por processamento vindas de diversos tipos de aplicações de usuários. Uma partição é formada por recursos físico-computacionais, como número de processadores, disco rígido e memória RAM. Dessa forma, aplicações podem ser submetidas para a partição que contém o arcabouço necessário e os recursos físicos mais apropriados. Nesse contexto, um único supercomputador pode ser utilizado para manter plataformas de computação em nuvem e de HPC em separado [Badia et al., 2013].

Centrais de Dados como do Google e do Grid'5000[1] possuem comportamento dinâmico em relação ao número de plataformas instanciadas. Do ponto de vista do escalonador dessas Centrais de Dados, uma plataforma de computação em nuvem ou de HPC é também uma aplicação com tempo de processamento pré-estabelecido, que muitas vezes pode ser também estendido após iniciado, cuja execução consiste em definir uma partição, implantar uma plataforma e implantar um ou mais arcabouços para tarefas de aplicações. A Figura 2.1 representa um exemplo de escalonamento de plataformas determinado pela Central de Dados.

Com o objetivo de detalhar uma arquitetura de Central de Dados onde as plataformas são instanciadas e recebem aplicações, apresenta-se a Figura 2.2. Essa arquitetura foi identificada com base nos documentos públicos do Google [Reiss et al., 2011] e do Grid'5000, os quais possuem mecanismos similares.

[1]Grid'5000 é uma plataforma de computação em larga escala do Governo da França para experimentação científica, que pode ser controlada e monitorada pelos seus utilizadores. Mais informações podem ser obtidas em https://www.grid5000.fr/.

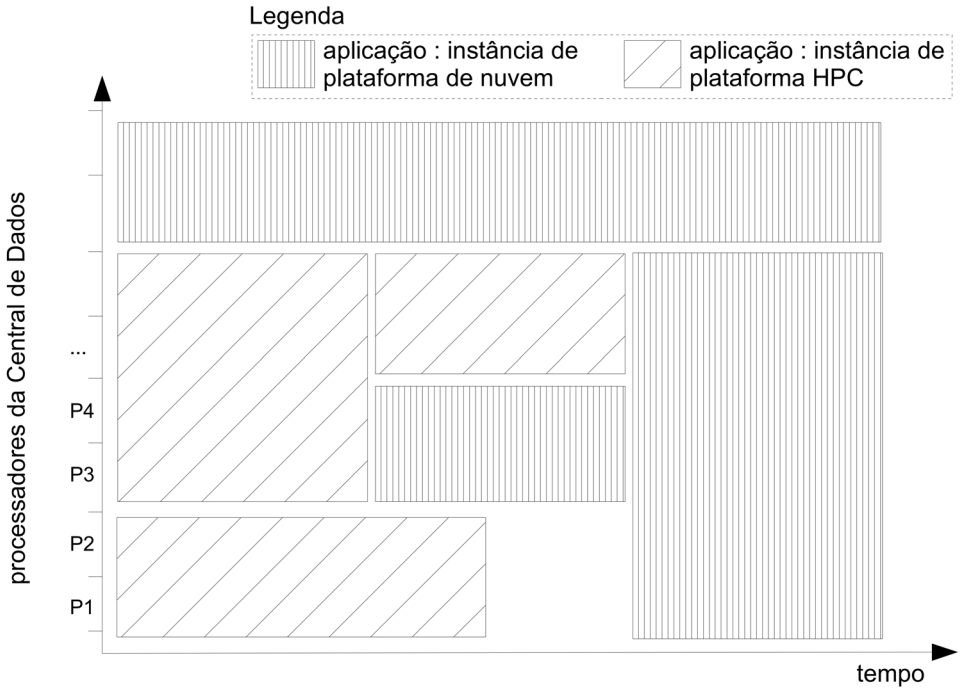


Figura 2.1: Gráfico Gantt com exemplo de escalonamento de aplicações de instâncias de plataformas em centrais de dados.

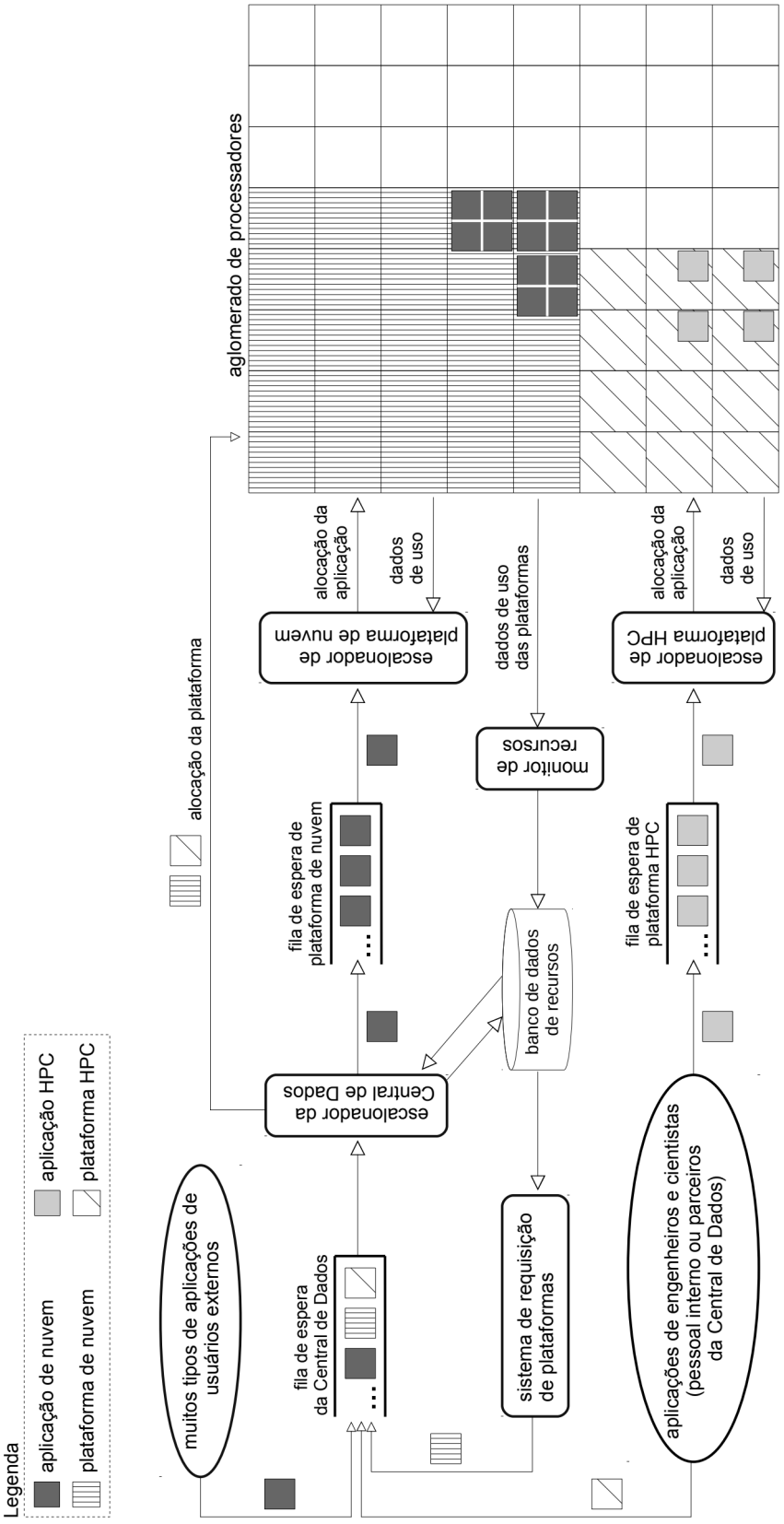


Figura 2.2: Fluxo de dados dentro da arquitetura de uma Central de Dados com plataformas HPC e de computação em nuvem.

A submissão de aplicações à Central de Dados é feita por três requisitores de recursos, representados na Figura 2.2 como *sistema de requisição de plataformas*; *aplicações de engenheiros e cientistas*; e *muitos tipos de aplicações de usuários externos*. Uma Central de Dados possui no mínimo uma fila de espera para atender aplicações desses requisitores.

Aplicações do tipo plataforma são requisitadas por sistemas internos da Central de Dados, tal como o sistema de requisição de plataformas, visando atender a demanda por processamento. Assim, os serviços de nuvem são tratados de maneira dinâmica e otimizada. Plataformas de nuvem que atendem muitos tipos de aplicações de usuários externos possuem longo tempo de duração e maior prioridade. Instâncias de plataformas HPC são requisitadas como sendo aplicações por engenheiros e cientistas parceiros da Central de Dados. Esses parceiros possuem acesso direto a plataformas que requisitaram, tendo também controle sobre os seus subsistemas, o que lhes permitem melhor observação científica e reconfiguração de seus subsistemas.

O escalonamento de aplicações realizado pelo escalonador da Central de Dados pode resultar na implantação de plataformas HPC e de nuvem em um mesmo supercomputador, como representado na figura pelo aglomerado de processadores. O papel desse escalonador também é redirecionar aplicações de nuvem a partir da sua fila de espera para outras filas de espera de plataformas de nuvem com capacidade para atender os requisitos de *software* e *hardware*. Usualmente, cada plataforma instanciada tem seu próprio escalonador e sua própria fila de espera. Um escalonador coleta aplicações da sua fila e determina o escalonamento dentro do seu domínio de processadores. Para tomada de decisão sobre alocação, o escalonador considera dados de uso desses processadores. Na figura, observa-se que aplicações de nuvem e de HPC podem ser organizadas de maneira diferente em cada plataforma.

Torna-se importante a sintetização dos elementos das plataformas de computação em larga escala para as otimizações de escalonamento propostas mais adiante neste trabalho. Para tal, apresenta-se na Figura 2.3 uma visão em alto nível desses elementos, cuja abstração foi desenvolvida com base nos trabalhos [Trystram et al., 2015] e [Georgiou, 2006].

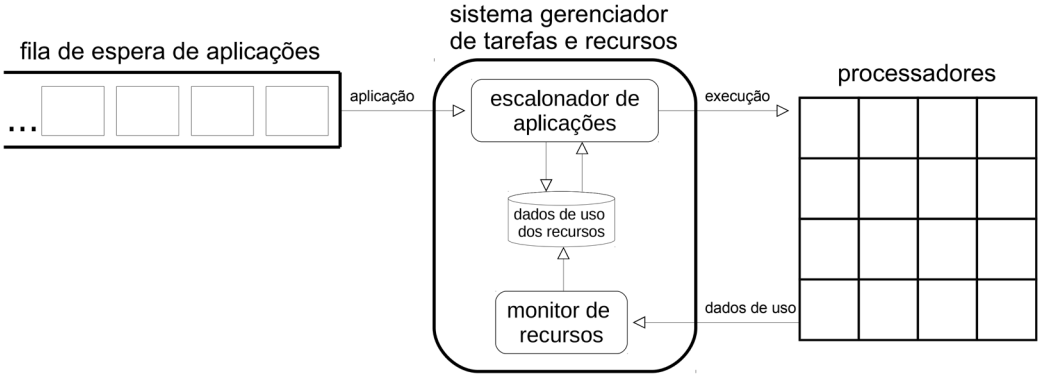


Figura 2.3: Principais elementos de uma plataforma de computação em larga escala.

Com base na Figura 2.3, aplicações submetidas são inicialmente armazenadas em uma fila de espera. Em seguida, o escalonador coleta aplicações dessa fila e processa uma estratégia de escalonamento para indicar onde (i.e. quais processadores) e quando as tarefas da aplicação devem ser executadas. Os recursos computacionais disponíveis e o estado atual das outras tarefas de aplicações já escalonadas são dados úteis à tomada de decisão do escalonador. O elemento responsável pelo monitoramento e armazenamento desses dados é o *monitor de recursos*.

Não é escopo deste trabalho abordar conexões entre centrais de dados, o que implicaria em abordar o tema *Federação*. Esse tópico é explorado em [Rochwerger et al., 2009, Elmroth et al., 2009].

2.2 Comparação entre plataformas HPC e de nuvem

Plataformas HPC e de computação em nuvem possuem objetivos distintos, embora possam utilizar a mesma infraestrutura computacional. Plataformas HPC são usualmente implantadas em computadores homogêneos. A homogeneidade reduz a complexidade da rígida parametrização exigida pelo sistema, onde usuários necessitam determinar os recursos requisitados e o tempo de execução nesses recursos. Enquanto isso, em plataformas de nuvem é usual o envolvimento de computadores heterogêneos de redes privadas e aglomerados de computadores, cada qual com sua homogeneidade, o que torna mais complexa a previsão de tempo de execução.

As principais características de plataformas de nuvem são definidas na norma ISO/IEC 17788 [JTC, 2014]. A seguir, comparam-se as características descritas nessa norma com características de plataformas HPC:

- **Serviços sob demanda**: nuvens são orientadas a serviços, onde o provedor de um serviço pode ser um *software* (*Software as a Service* - SaaS), uma plataforma (*Platform as a Service* - PaaS), ou uma infraestrutura (*Infrastructure as a Service* - IaaS). Um usuário negocia um serviço com o provedor e pode ampliar/reduzir as necessidades de recursos sob demanda. Interfaces de usuários são providas para facilitar o acesso e o gerenciamento dos serviços. Por outro lado, uma arquitetura HPC é orientada a aplicação, pode exigir uma complexa parametrização para uso dos seus recursos computacionais e existem poucas facilidades de utilização do sistema.

- **Múltiplos recursos para múltiplos usuários**: em nuvem, usuários podem utilizar recursos com base em uma longa lista de opções, como arcabouços, bibliotecas de programação, *hardware* virtual, entre outros. Recursos virtualizados de múltiplos usuários podem compartilhar um mesmo recurso físico, como ocorre com *threads* no modo de processamento *time-sharing*. Em HPC, recursos são reservados exclusivamente para cada usuário, com base em agendamento. O processamento em *time-sharing* também pode ser habilitado em HPC, mas esse mecanismo não é frequentemente utilizado, devido ao seu impacto na complexidade das análises matemáticas normalmente considerada por seus usuários.

- **Elasticidade rápida**: um provedor de nuvem pode redimensionar os recursos utilizados pelas aplicações, a fim de cumprir um ocordo SLA, tal como aumentar a fatia de tempo de uso da CPU para reduzir o tempo de resposta. Plataformas HPC também podem gerenciar SLAs e a elasticidade também pode ser aplicada. Entretanto, o limite de tempo para uma aplicação é frequentemente longo e pode ser estendido quando reconfigurado com antecedência. Nesse contexto, a elasticidade de recursos não é um mecanismo essencial.

- **Medição de serviços**: em nuvem, a utilização de recursos é medida e pode ser cobrada do usuário. O usuário também pode monitorar e limitar o uso dos recursos, controlando assim o que é cobrado. Em HPC, recursos de computação também podem ser mensurados, mas não existe frequentemente uma política que garanta a Qualidade de Serviço e o valor pago por isso. Ainda em HPC, todas as aplicações em execução são removidas quando a sessão do usuário alcança seu limite de tempo pré-estabelecido.

- **Multi-inquilinato**: em nuvem existem acordos visando isolação, governança, SLA e um modelo de negócios. Isso ocorre especialmente em nuvens públicas, as quais

habilitam acordos entre usuários e provedores. Multi-inquilinato garante o acordo em todos os níveis do serviço contratado. Em HPC, acordos também podem ser aplicados, mas a contabilização de créditos e garantias de reserva de recursos nas várias camadas de *software* não são práticas comuns.

Os autores [Sheng et al., 2012] observaram características de nuvem e de HPC sob o ponto de vista prático, envolvendo a Central de Dados do Google e alguns grids HPC. Esses autores observaram que 80% das aplicações em nuvem duram menos que 1000 segundos, enquanto em HPC esse tempo costuma ser maior; cerca de 94% das aplicações em nuvem possuem tempo de execução inferior a 3 horas; o Google possui uma frequência muito maior e regular de aplicações que chegam em comparação aos grids, sendo 552 submissões por hora em média, enquanto nos grids observados esse número varia entre 8 e 126; em nuvem, a ampla maioria das aplicações requer somente um processador e o seu conjunto de tarefas é organizado em sequência; a variação da carga da CPU no Google é cerca de 20 vezes maior que o observado nos grids, haja visto a diversidade de aplicações de múltiplos usuários sendo processadas em modo *time-sharing*; em nuvem o nível de utilização da CPU é em geral menor que o nível de utilização da memória RAM, enquanto em HPC foi observado o oposto disso.

2.3 Sistemas Gerenciadores de Tarefas e Recursos

Com o intuito de explorar implementações de RJMS, nesta seção são apresentadas algumas características de Centrais de Dados bem conhecidas. A plataforma para experimentação científica Grid'5000 utiliza o RJMS OAR [Margery et al., 2014]. No OAR, cada computador está associado a um conjunto de propriedades. Um escalonador global determina uma partição com os recursos computacionais e implanta o modelo de plataforma requisitado pelo usuário. A plataforma instanciada possui seu próprio escalonador, o qual é usualmente independente do escalonador global. Recursos podem estar distribuídos em rede e cada sessão de usuário possui sua própria VLAN (*Virtual Local Area Network* - Rede Local Virtual). Os usuários também podem escolher quando os recursos devem ser alocados e a experimentação inicia sem sua intervenção. Esse mecanismo é conhecido como *Reserva Antecipada de Recursos*. Por fim, a descrição e o monitoramento dos recursos computacionais utilizados ficam disponíveis em um repositório *git* para consulta do usuário.

Os engenheiros do Google desenvolveram o *Omega Management System* [Schwarzkopf et al., 2013], o qual possui um escalonador de dois níveis que distribui aplicações a partir de um escalonador global para escalonadores de outras plataformas. Cada instância de plataforma tem a visão global do estado de todas as demais instâncias do domínio. Esse modelo é conhecido como *Estado Compartilhado*. A maior parte dos recursos computacionais é reservada para executar aplicações de maior prioridade, enquanto os recursos restantes são utilizados pelas aplicações de menor prioridade.

O Yahoo e o eBay utilizam a plataforma Apache Hadoop YARN (*Yet Another Resource Negotiator* - Ainda Outro Negociador de Recursos) [Vavilapalli et al., 2013]. YARN é composto por dois componentes principais: um gerenciador de recursos e muitos gerenciadores locais. O gerenciador de recursos mantém uma fila de capacidades com aplicações executando e aplicações pendentes. Com base nesses dados, os recursos são negociados com os gerenciadores dos arcabouços locais. Os arcabouços lançam aplicações em BSP (*Bulk-Synchronous Parallel* - Sincronização Paralela em Massa) ou MapReduce, e mantêm o gerenciador de recursos informado sobre os detalhes da execução.

O Twitter utiliza a plataforma Mesos [Hindman et al., 2011]. Mesos contém um componente central chamado *Mestre*, o qual mantém um escalonador para cada tipo de arcabouço, como o Hadoop MapReduce e o MPI (*Message Passing Interface*). As aplicações são distribuídas para componentes chamados *Escravos*, usando como critério o tipo da aplicação e seus recursos requisitados. Ambos os arcabouços podem ser instalados no mesmo computador e são acionados de acordo com a decisão tomada pelo Mestre. Uma vez que a aplicação é transferida para seu domínio, cada escalonador local toma suas próprias decisões.

O Facebook utiliza a plataforma Corona [Facebook-team, 2012]. No Corona existem dois níveis de escalonadores: o escalonador do primeiro nível é chamado de *Gerenciador do Aglomerado*, o qual distribui aplicações para os chamados *Job Trackers* sem monitorar execuções; e o segundo nível é formado por *Job Trackers*. Um *Job Tracker* gerencia o seu próprio aglomerado de processadores e monitora o progresso das suas aplicações. Cada computador do aglomerado é conhecido como *Task Tracker*. O Grid do Facebook está dividido em aglomerado para aplicações longas e aglomerado para aplicações curtas.

O Alibaba utiliza o escalonador chamado Fuxi [Zhang et al., 2014]. As aplicações no Fuxi podem conter sua própria especificação de recursos. Existe um componente chamado *Fuxi Master*, que marca as aplicações como rígidas ou leves. O estado leve indica que a aplicação pode ser despejada/reiniciada para garantir recursos para outra aplicação. Também existem agentes de *software* para monitorar o funcionamento e o estado dos arcabouços instanciados em cada aglomerado. Uma aplicação pode utilizar mais de um aglomerado ao mesmo tempo. Por fim, a reserva de recursos feita por uma aplicação pode ser reaproveitada por outra aplicação, o que reduz o tempo total gasto por elas no sistema, pois não precisam aguardar pela preparação dos recursos.

Apresenta-se na Tabela 2.1 uma comparação envolvendo as principais características dos escalonadores apresentados.

A partir da revisão de literatura e da tabela apresentadas, observa-se que na arquitetura de centrais de dados tem-se um escalonador principal que distribui aplicações para escalonadores de plataformas. A decisão sobre qual plataforma recebe a aplicação ocorre por meio de análise das características da aplicação e da previsão dos recursos que ela consome. Com isso, o escalonador principal envia a aplicação para a plataforma que pode atender os requisitos de *hardware* e *software*. Por vezes, devido a circunstâncias de compartilhamento de recursos por várias aplicações e acordos entre usuário e provedor, o uso de recursos precisa ser priorizado para determinados tipos de aplicação, para que terminem no seu tempo previsto. Esse ajuste de uso de recursos é feito *online* em algumas centrais de dados, ou seja, durante a execução. Um exemplo de ajuste é o aumento de tempo de uso da CPU, o que acelera o processamento. Aplicações com menor prioridade, onde a relação custo-benefício não é relevante para justificar o ajuste *online*, são deixadas em segundo plano no aguardo de liberação de recursos ou são simplesmente despejadas e reinicializadas. Para evitar o ajuste *online*, a plataforma Grid'5000, por exemplo, atribui o dobro do tempo previsto para processamento de cada aplicação. Se de um lado isso evita o ajuste *online*, por outro, abrem-se intervalos de tempo ocioso no plano de escalonamento das aplicações, que pode ser interpretado como desperdício de recursos. Essas situações de escalonamento ocorrem frequentemente em centrais de dados.

2.4 Acordos e Objetivos de Nível de Serviço

O uso de um acordo SLA (*Service Level Agreement* - Acordo de Nível de Serviço) é uma prática comum para definir contrato entre duas partes em ambientes de computação em nuvem. Segundo [Wieder et al., 2011], SLA surgiu como derivação de padrões XML específicos para

Tabela 2.1: Comparação de características de escalonadores em Centrais de Dados

Central de Dados	**RJMS**	**Arquitetura**	**Método de previsão de recursos para uma aplicação**	**Característica(s) usada(s) para classificar aplicações**	**Faz ajuste *online* de recursos?**
Grid'5000	OAR	Monolítico	indicação pelo usuário em linha de comando	não há (zero prioridade)	não
Google	Omega	*Estado Compartilhado*	análise de histórico de aplicações similares	prioridades; curto ou longo tempo de processamento	sim
Yahoo, eBay	YARN	Dois Níveis	identificação em tempo de execução	-	sim
Twitter	Mesos	Dois Níveis	regressão lógica por aprendizado de máquina	prioridades	não
Facebook	Corona	Dois Níveis	-	prioridades; curto ou longo tempo de processamento	sim
Alibaba	Fuxi	Monolítico	indicação pelo sistema emissor da aplicação	prioridades; leve (fácil de reiniciar) ou rígida	sim

Web, tais como WS-Agreement (*Web Services agreement* - Acordo de Serviços Web), WSLA (*Web Service Level Agreement* - Nível de Acordo de Serviços Web) e WSDL (*Web Service Definition Language* - Linguagem de Definição de Serviço Web).

Institutos como ISO (*International Organization for Standardization* - Organização Internacional para Padronização), NIST (*National Institute of Standards and Technology* - Instituto Nacional de Padrões e Tecnologia) e C-SIG (*Cloud Select Industry Group* - Grupo da Indústria Selecionada de Nuvem) estão fazendo esforços no sentido de definir termos, mecanismos e métricas relativos a SLA.

A fim de apresentar conceitos relativos a SLA, apresenta-se uma abstração na Figura 2.4 envolvendo computação orientada a serviços. Nessa figura, a *1ª fase* é representada pelo processo de elaboração de uma oferta de SLA contendo os requisitos da aplicação. Cada requisito de recurso é chamado SLO (*Service Level Objectives* - Objetivos de Nível de Serviço).

Na *2ª fase* ocorre o processo de negociação entre o usuário e o provedor do serviço. Em geral, os principais SLOs da oferta SLA são definidos pelo *sistema propositor* do usuário e os demais são definidos pelo *negociador e criador de modelos de SLA* do provedor. Entendem-se como SLOs principais os referentes ao número de processadores e memória requisitados. Um exemplo de SLO definido pelo provedor consiste na limitação do número de submissão de aplicações por unidade de tempo. A 2ª fase encerra-se quando o provedor aceita a oferta. Em caso negativo, realiza-se uma contraoferta. Para esta decisão, o provedor toma como base os modelos de SLA existentes e a disponibilidade de tais recursos sem violação dos SLOs. Novos modelos de SLA são construídos pelo provedor a todo momento, que torna o processo de negociação mais ágil para novos acordos.

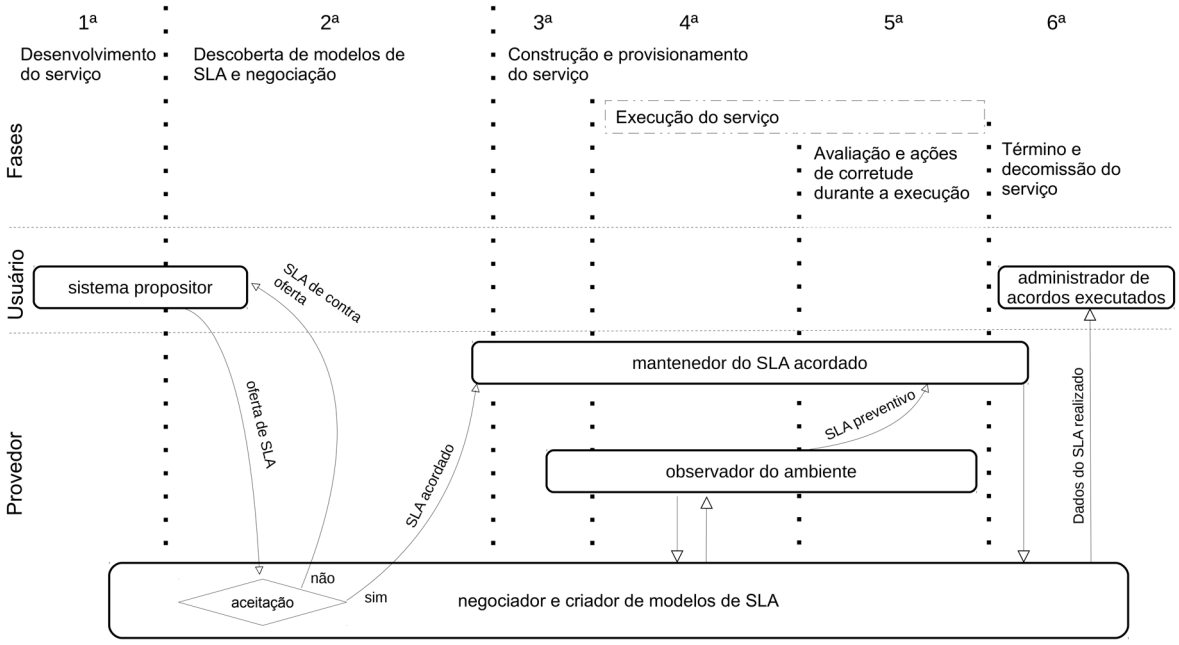


Figura 2.4: Fluxo de dados de SLA na computação orientada a serviços (adaptação de [Wieder et al., 2011]).

A *3ª fase* e a *4ª fase* consistem respectivamente no provedor disponibilizar os recursos e manter o serviço acordado. A partir da disponibilização dos recursos, a aplicação é executada pelo provedor e o cumprimento do SLA é monitorado pelo *observador do ambiente*.

Um *SLA preventivo* é determinado pelo provedor com o intuito de antecipar-se à violações. Na *5ª fase*, o *mantenedor do serviço SLA acordado* analisa dados do SLA preventivo para tomar ações de elasticidade. Isso ocorre com o objetivo de melhorar o desempenho da aplicação e evitar violações. Essas medidas são conhecidas no mecanismo como *ações de avaliação e correção*.

Na *6ª fase*, o serviço é finalizado e ocorrem as decomissões. Exemplos de decomissões são: incluir créditos na carteira do usuário devido ao descumprimento do SLA pelo provedor; cobrar mais do usuário porque a frequência de submissão de aplicações por unidade de tempo foi maior que o acordado, desde que essa cobrança também tenha sido previamente habilitada no acordo.

Um documento SLA contém termos *funcionais* e *não-funcionais* estabelecidos por um complexo processo de negociação [Koller, 2010]. Termos funcionais definem atributos sobre a infraestrutura computacional, tais como número de processadores, quantidade de memória que pode ser alocada e vários outros. Enquanto isso, termos não-funcionais envolvem desempenho, disponibilidade, segurança e outros. Mais detalhes sobre SLA podem ser observados nos apêndices. No Apêndice A.1 apresenta-se uma especificação geral de SLA desenvolvida por [Freitas et al., 2011]. No Apêndice A.2 apresenta-se um resumo dos termos definidos por [Wu et al., 2011] para SLAs em ambientes de computação em nuvem. Uma proposta de SLA para plataformas HPC foi proposta por [Koller, 2010]. A lista dos termos de Koller et al. e um resumo dos seus atributos são apresentados no Apêndice A.3.

2.5 Resumo

Neste capítulo foram apresentadas características de sistemas de computação em larga escala, exemplos de Centrais de Dados comerciais e o conceito de SLA. Por meio deste estudo pode-se obter uma visão arquitetural dos elementos que formam plataformas de computação em larga escala. Nessa visão arquitetural foi contextualizado o papel do escalonador. Observa-se que a estratégia de escalonamento de aplicações é um elemento de um sistema escalonador. Enquanto o escalonador gerencia a fila de aplicações e interage com os recursos do ambiente, a estratégia de escalonamento tem o objetivo de organizar as aplicações para uso dos recursos computacionais. O próximo capítulo detalha estratégias de escalonamento de aplicações, resume diversos trabalhos relevantes da literatura científica e apresenta desafios na área.

Capítulo 3

Escalonamento de Aplicações

Escalonamento é um procedimento que determina sequenciamento de elementos em vários contextos de produção [Tchernykh, 2014]. Estratégias de escalonamento são aplicadas, por exemplo, visando vazão máxima, minimização de custo de armazenamento de volumes de qualquer espécie (ex.: caixas, veículos, etc), redução do tempo de completude de operações e outras. Alguns exemplos específicos ocorrem em linhas de montagem, a fim de conseguir um design ótimo de impressão de peças em placas; na área de planejamento para determinação de rotas de veículos, a fim de reduzir o tráfego; organização de volumes de material radioativo em recipientes de diferentes tamanhos, a fim de reduzir o número de bandejas de imersões em usinas nucleares e maximizar a quantidade de material imersa; organização de volumes em caminhões de transporte de carga, a fim de reduzir o número de caminhões necessários; entre muitas outras. Em situações de tempo real, o escalonamento com base em limites de tempo pode ser crucial para a operação do ambiente, como é o caso do controle de tráfego aéreo, a fim de permitir maior uso do espaço em planejamento temporal [Tchernykh, 2014].

Em sistemas computacionais, o escalonamento determina onde e quando aplicações são executadas [Trystram et al., 2015]. Estratégias de escalonamento exploram formas de executar organizadamente aplicações em processadores de uma partição, visando, por exemplo, terminar o processamento delas em menor tempo. Detalhes sobre aplicações, partições e estratégias de escalonamento em computação são apresentadas nas seções que seguem. No Capítulo 3.1 constam a definição de aplicação e principais características abordadas na literatura científica. No Capítulo 3.2 são descritos os tipos de partição em centrais de dados, definição e características de seus processadores. No Capítulo 3.3 são discutidas formas de processamento e migração de aplicações. No Capítulo 3.4 são apresentadas classificações, mecanismos, métricas e formalizações matemáticas para escalonamento. No Capítulo 3.5 são apresentados trabalhos relacionados em estratégias de escalonamento. No Capítulo 3.6 são discutidos o escalonador e a estratégia de escalonamento de aplicações da nuvem Google. No Capítulo 3.7 são apresentados trabalhos relacionados em SLA. Por fim, no Capítulo 3.8 são apresentados desafios em aberto na área.

3.1 Características de aplicações

Uma aplicação é qualquer processo gerado por *software*, consistindo de uma ou mais tarefas do mesmo usuário. No contexto de Central de Dados, uma aplicação pode ter origem interna ou externa. Segundo [Reiss et al., 2012], com base em análise da Central de Dados do Google, aplicações internas são nomeadas de *produção*, tais como as que instanciam plataformas de nuvem e de HPC. Essas aplicações são dominantes em termos de uso de recursos. Platafomas

de nuvem são requisitadas pela própria Central de Dados a fim de habilitar e oferecer seus serviços. Plataformas HPC são requisitadas por usuários ditos *internos*, tais como engenheiros de computação e parceiros. Outras aplicações internas, não dominantes, são nomeadas como *monitoramento* e *infraestrutura* e estão relacionadas aos serviços administrativos da Central de Dados, tais como indexação de arquivos e atualização de banco de dados sobre seus clientes.

Aplicações de contexto externo são usualmente criadas por *softwares* que interagem com usuários finais, tais como os que utilizam serviços de nuvem ou que submetem aplicações HPC. Essas aplicações são originadas externamente mas processadas internamente na Central de Dados. Exemplos de aplicações de nuvem são envio/recepção de mensagens de e-mail, editoração de documentos, comércio eletrônico, armazenamento de dados, aluguel de máquinas virtuais, entre outras. Aplicações HPC possuem cunho científico, como experimentação de técnicas de sistemas distribuídos e processamento de algoritmos para várias áreas de conhecimento. No Google, aplicações de contexto externo são nomeadas como *gratis* e *outras* [Reiss et al., 2012]. O tipo *outras* está relacionado com serviços de maior prioridade.

Algumas propriedades de aplicação são intercambiáveis entre plataformas de nuvem e de HPC, sendo tratadas por ambos os escalonadores com pouca ou nenhuma adaptação. O que ocorre frequentemente em artigos científicos é a representação de uma aplicação contendo poucas variáveis. Isso ocorre como forma de diminuir a complexidade das análises matemáticas na elaboração de novas estratégias. Em termos de notação matemática, uma aplicação é usualmente definida como J_j e suas propriedades são representadas no formato de tupla. Apresentam-se a seguir notações com base nos trabalhos [Tchernykh, 2014] e [Trystram et al., 2015].

Um conjunto de aplicações independentes é descrito pela notação $\mathcal{J} = \{J_1, J_2, ..., J_j, ..., J_n\}$. Complementarmente, um exemplo com aplicações dependentes (ou com precedência) é descrito como $J_1 \prec J_j$, onde J_1 precede J_j. Na Figura 3.1 destacam-se as principais propriedades de uma aplicação sob a perspectiva de sua execução, como o número de processadores h_j, o tempo de processamento p_j, o tempo máximo de processamento d_j e a quantidade de memória requisitada b_j. A fim de representar o tempo máximo de processamento como um tempo rígido (ou tempo não flexível), pode-se utilizar $\tilde{d_j}$. A propriedade de quantidade de memória b_j é usualmente tratada de forma independente do número de processadores h_j e, portanto, sendo muitas vezes abstraída das ilustrações do tipo Gantt, pois as estratégias de escalonamento não conseguem boas soluções para esses dois critérios ao mesmo tempo. Assim, diversos trabalhos científicos focam apenas em h_j, p_j e d_j como principais características de uma aplicação [Sheng et al., 2012] [Tchernykh, 2014]. Em relação à demais propriedades da Figura 3.1, o momento da chegada a_j e o momento da inicialização σ_j recebem valores atribuídos pelo escalonador, enquanto o tempo de completude C_j depende do momento da finalização da aplicação, valor este também atribuído pelo escalonador ao término da execução.

Durante a negociação com o usuário, como descrito na Seção 2.4, a Central de Dados utiliza um conjunto de valores de SLOs pré-determinados, tais como tempos de resposta de tipos de aplicação. Esses valores são atualizados periodicamente levando em consideração a execução de novas aplicações e a ociosidade das plataformas. Em notação matemática, representa-se o conjunto de SLOs pré-determinados como $\mathcal{SLO} = \{SLO_1, SLO_2, ..., SLO_i, ..., SLO_z\}$. Assim, dada uma aplicação J_j com SLO acordado, considere $J_j = (h_j, p_j, b_j, d_j, SLO_{j,i})$, onde $SLO_{j,i}$ é o i tempo de resposta SLO negociado para j.

Alguns SLOs são abstraídos para compor uma única propriedade de aplicação chamada *tempo de resposta*. Uma razão para isso é que o uso dos recursos de maneira compartilhada tornou mais complexo o entendimento de uma aplicação em execução, sendo melhor encapsular várias propriedades em apenas uma, como a velocidade de processamento e o desempenho de todos os recursos envolvidos na sua execução. O SLA também se torna mais intelígivel pelo

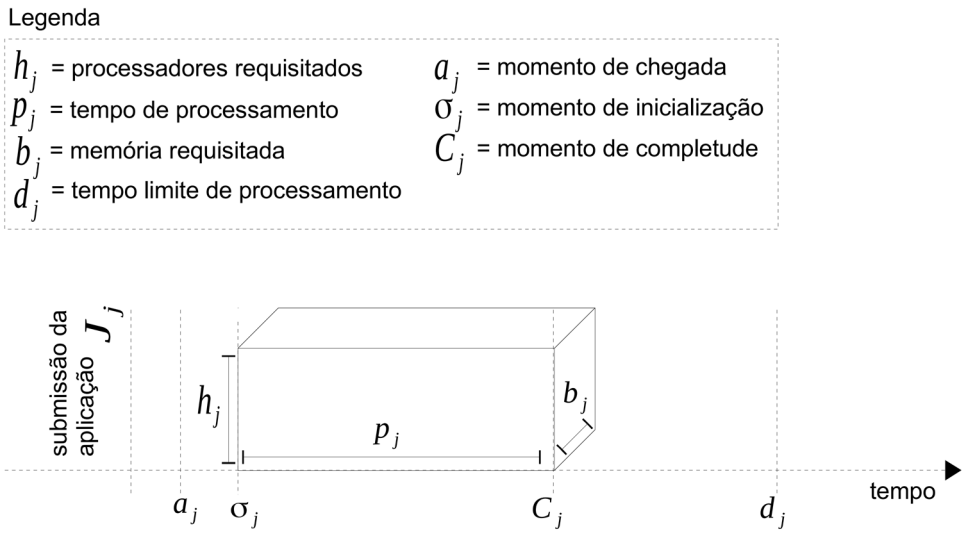


Figura 3.1: Principais propriedades de uma aplicação.

usuário ao interpretar o termo tempo de resposta como SLO. Isto também reduz a complexidade das análises matemáticas.

Do ponto de vista de processamento, tarefas de aplicação são executadas de maneira sequencial ou paralela em processadores. A Figura 3.3 ilustra essa diferenciação. Aplicações com tarefas paralelas são classificadas de acordo com a flexibilidade do paralelismo [Feitelson et al., 1997]. Essa classificação é observada na Figura 3.2, onde constam:

- **aplicação rígida**: o número de processadores relacionados não pode ser alterado no escalonamento;
- **aplicação moldável**: o número de processadores relacionados pode ser ajustado no escalonamento desde que o trabalho total (número de processadores multiplicado pelo tempo de processamento) seja o mesmo;
- **aplicação maleável**: desde que o trabalho total seja mantido, o número de processadores relacionados pode ser ajustado no escalonamento e também durante a execução da aplicação;
- **aplicação evolutiva**: a aplicação passa por diferentes fases desde o seu escalonamento. Essas fases ajustam o número de processadores relacionados de acordo com as necessidades da aplicação.

Uma aplicação em Central de Dados é caracterizada como um conjunto de tarefas do mesmo usuário, tendo como destino um serviço de nuvem ou uma plataforma HPC. Uma aplicação também pode ser a própria instância dessas plataformas, como descrito na Seção 2.1. Os autores [Sheng et al., 2013] identificaram que na Central de Dados do Google as aplicações agrupam tarefas conforme a organização representada na Figura 3.4. Segue uma descrição da classificação proposta por esses autores:

- **aplicação *single-task***: aplicação com apenas uma tarefa. Como exemplo cita-se uma requisição de página *web*;

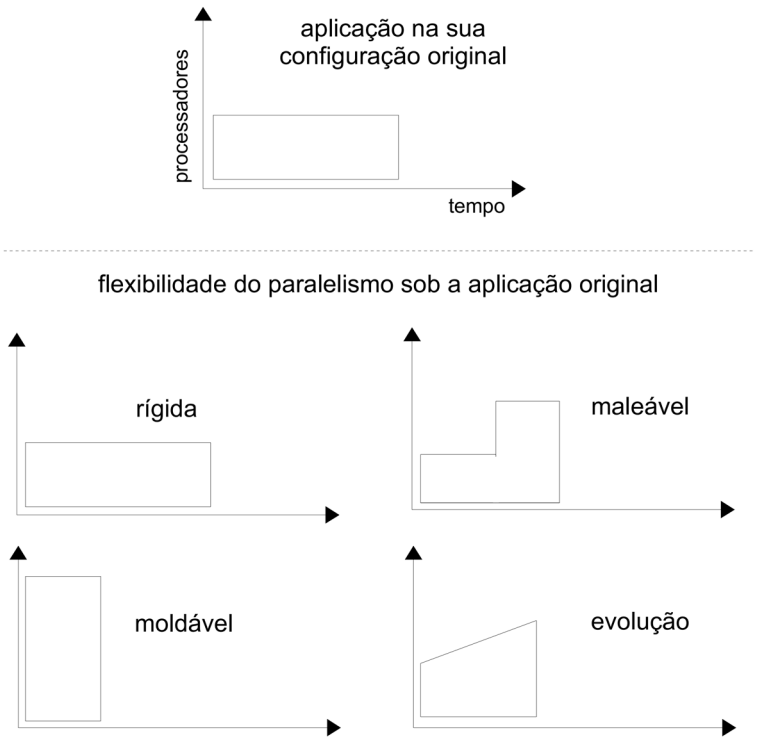


Figura 3.2: Classificação de aplicações de [Feitelson et al., 1997] de acordo com a flexibilidade do paralelismo.

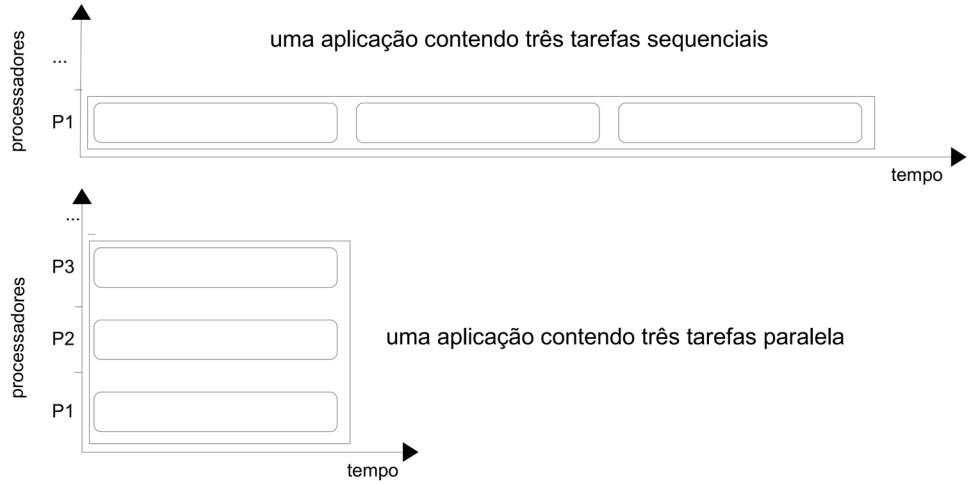


Figura 3.3: Diferenciação entre aplicação sequencial e aplicação paralela.

- **aplicação *sequential-task***: as tarefas estão organizadas em série. Como exemplo cita-se uma aplicação de transação bancária, onde é possível apenas debitar valor se houver saldo. A tarefa de verificação do saldo ocorre previamente. A organização em série também pode envolver tarefas sem dependências, visando processá-las em apenas uma unidade de processamento. Um exemplo disto é a organização de tarefas de um mesmo usuário, tais como o envio de uma mensagem de e-mail e uma requisição de página *web*, ambas na mesma aplicação;

- **aplicação *batch-task***: muitas tarefas pequenas estão conectadas umas nas outras, formando um grafo. Como exemplo cita-se o processamento de gênomas, no campo

científico da área de biológicas, onde há uma complexa rede de dados. Geralmente, esses dados possuem dependências, fazendo com que algumas tarefas aguardem o processamento de outras tarefas;

- **aplicação *mix-mode***: tipo misto de *sequential-task* e *batch-task*.

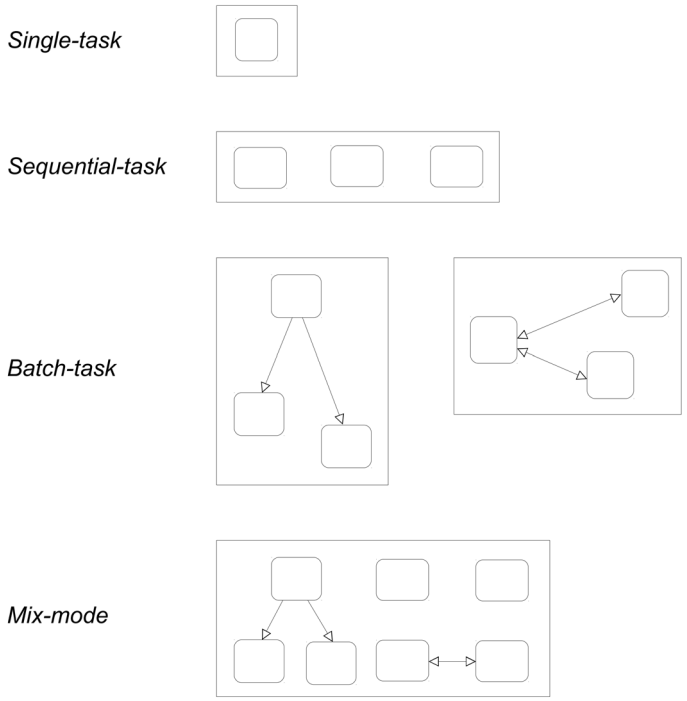


Figura 3.4: Tipos de aplicação identificadas por [Sheng et al., 2013] na Central de Dados do Google.

Essas classificações apresentadas são úteis para delimitar o escopo da estratégia de escalonamento e consequentemente para simplificar a sua descrição matemática.

3.2 Características de partições

Em centrais de dados, plataformas de computação em nuvem e de HPC são instaladas em partições distintas, as quais podem consistir de diferente número de processadores. Cada partição $\mathcal{P} = \{P_1, P_2, ..., P_q, ..., P_m\}$ está associada a uma ou mais filas de aplicação e seus processadores servem como *pool* de execução para essas filas. Uma classificação de partições de acordo com o seu tamanho foi feita por [Feitelson et al., 1997], como segue:

- **fixo**: o tamanho da partição é definido pelo sistema administrador;
- **variável**: o tamanho da partição é determinado no momento da submissão da aplicação, tendo como base os requisitos do usuário;
- **adaptativo**: o tamanho da partição é determinado pelo escalonador no momento que a aplicação é iniciada, tomando como base a carga do sistema e os requisitos do usuário;
- **dinâmica**: o tamanho da partição pode mudar durante a execução da aplicação.

Do ponto de vista tecnológico, processadores em partições possuem muitas características que os definem, como seus níveis de memória *cache* e registradores. Por outro lado, do ponto de vista de representação para análises matemáticas, como ocorre na descrição de estratégias de escalonamento, um processador é basicamente definido em termos de sua velocidade. Assim, na literatura, tem-se representação de conjunto de processadores homogêneos (ou com a mesma velocidade de processamento $P_1 = ... = P_q = P_m$), representação de conjunto de processadores heterogêneos (ou processadores de diferentes velocidades $P_{iq} = \frac{P_q}{s_i}, i = 1, ..., m$, onde s_i é o fator de velocidade de processamento de P_i) e representação de conjunto de processadores especializados para certas tarefas [Tchernykh, 2014]. Processadores heterogêneos são usualmente normalizados tendo como referência o processador de maior velocidade. Essas características são importantes para determinação do escalonamento, porque tem efeito no tempo de resposta das aplicações.

3.3 Processamento e Migração de Aplicações

Com base em [Silberschatz et al., 2015], nesta seção apresentam-se características de processamento e migração de aplicações. Objetiva-se relacionar os conceitos apresentados nesta seção com os conceitos de aplicação e partição apresentados nas seções anteriores. Esses conceitos são abordados frequentemente em estudos de sistemas operacionais. O contexto de Central de Dados é similar, distribuído e em larga escala. Consideram-se essas informações como fundamentais ao atendimento de escalonamento, apresentado com mais detalhes na próxima seção.

Embora um processador seja denominado como uma máquina em diversos trabalhos sobre escalonamento encontrados na literatura, na prática um processador é um núcleo de processamento capaz de executar uma *thread* de aplicação. Como visto na Seção 3.1, uma aplicação é um processo que pode consistir de várias tarefas. Por sua vez, uma tarefa é executada por meio de uma *thread* no processador. Uma aplicação pode requisitar várias *threads*, a fim de executar todas as suas tarefas. Assim, a execução de uma aplicação pode ser realizada em um ou mais processadores. Essas características permitem classificações de tipo de aplicação como as apresentadas na Seção 3.1: rígida, moldável, maleável e evolutiva. A execução dessas aplicações pode ser realizada em modo lote (ou *batch*), modo compartilhado (ou *time-sharing*), ou modo tempo-real. O paralelismo de processamento de aplicações pode ser tratado em camadas superiores da arquitetura computacional e, portanto, aplica-se para qualquer conjunto de processadores com esses modos.

No modo lote, o processador atende as aplicações de acordo com os seus tempos requisitados. Nesse modo, o processador tem foco no processamento sequencial, o que torna mais preciso o monitoramento de recursos e a previsão de desempenho. Devido a essa característica, plataformas HPC são usualmente configuradas para fazerem uso desta modalidade [Emeras et al., 2013].

No modo compartilhado, várias tarefas são intercaladas dentro de um intervalo de tempo bem definido no processador. No contexto de centrais de dados, a fatia de tempo de cada tarefa dentro de um intervalo de tempo bem definido pode ser redimensionada de acordo com o $SLO_{j,i}$ da aplicação, impactando na redução ou ampliação do tempo de completude C_j. Para esse mecanismo dá-se o nome de *elasticidade* de tempo de processamento, sendo compatível com tarefas do tipo maleável e evolução. Plataformas de nuvem fazem uso desta modalidade, pois do ponto de vista de modelo de negócio, conseguem atender muitas aplicações concorrentes e aproveitar melhor os recursos computacionais.

O escalonador Xen no Amazon utiliza a elasticidade durante a execução de máquinas virtuais, onde duas ou mais máquinas virtuais compartilham o mesmo processador, cada uma com um percentual bem definido de tempo de uso, cuja soma de todas não ultrapassa 100% [Cherkasova et al., 2007]. Quando amplia-se o percentual de tempo de uso para uma aplicação, diminui-se o mesmo valor de outra aplicação, mantendo assim sempre o total de 100% de uso. Por outro lado, nesse contexto a previsão de desempenho de aplicações pode tornar-se demasiadamente complexa. Devido a essa característica, esse mecanismo é raramente utilizado em plataformas HPC.

Por fim, dá-se o nome de modo *tempo-real* quando o controle do tempo de resposta da aplicação é rígido. Deseja-se nesse modo que aplicações terminem no seu tempo determinado. Entretanto, pequenos atrasos são toleráveis, dependendo do tipo de aplicação.

Um algoritmo de escalonamento considera usualmente as propriedades da aplicação e os dados do local de execução. No contexto do local de execução, destacam-se características como a velocidade de processamento, a contiguidade proporcionada à aplicação e a possibilidade de preempção das suas tarefas. A contiguidade consiste em aproximar a localização do conjunto de tarefas em processadores próximos um do outro, isso diminui custos de comunicação. A preempção é caracterizada pela troca de contexto de uma tarefa ou pela migração. Na troca de contexto, uma tarefa sendo executada é suspensa e mantida em segundo plano na memória principal enquanto outra tarefa assume seu lugar. O tempo de espera da tarefa suspensa pode ser rápido, com continuidade do processamento, ou pode ser seguido de migração para outro lugar, não necessariamente para outro processador. A migração pode ser encaminhada para outra fila de espera, um repositório de aplicações onde avalia-se a necessidade de reescalonamento, entre outros. Esse comportamento é comum em escalonamentos que consideram tarefas com diferentes prioridades, onde as tarefas de maior prioridade tomam o espaço das que possuem menor prioridade, fazendo com que algumas aplicações sejam realocadas no ambiente.

3.4 Estratégias de escalonamento

As estratégias de escalonamento são diferenciadas tendo como base o que é conhecido sobre a aplicação [Tchernykh, 2014], como segue:

- **estratégia determinística** (ou estática ou *off-fine*): assume-se que todos os dados requisitados para o desenvolvimento do escalonamento são conhecidos com antecedência. No ambiente fabril o escalonamento determinístico também é conhecido como *previsível*;
- **estratégia não-determinística**: assume-se que somente parte dos dados necessários é previamente conhecida, como ocorre em aplicações onde não se conhece o tempo de processamento.

O escalonamento ainda pode ser classificado em *online*, *non-clairvoyant* e estocástico. A forma *online* é caracterizada pela situação onde a natureza da aplicação é conhecida, mas no tempo na qual ela está aberta. O escalonamento *non-clairvoyant* considera problemas onde os requisitos de tempo de processamento não são especificados. Por fim, no escalonamento estocástico estão disponíveis informações probabilísticas sobre o comportamento do sistema, onde formas *online* ou *offline* são consideradas.

3.4.1 Métricas de escalonamento

Diversas métricas são utilizadas para determinar objetivos de estratégias de escalonamento. Em muitas delas o tempo de processamento tem papel essencial. Destacam-se algumas métricas a seguir:

- **tempo de resposta**: este dado isoladamente não determina o objetivo do escalonamento. Isto se deve porque ocorrem diversas etapas de tempo desde a inserção da aplicação na fila de espera até o seu encerramento. Assim, este dado é usado para determinar métricas como seguem:
  - *makespan* ou C_{max}: estes termos representam o tempo de completude de processamento de um conjunto de aplicações. Um exemplo de objetivo de escalonamento clássico é a minimização do máximo tempo de completude, ou minimização do C_{max}.
  - *turnaround*: determina todo o tempo gasto pela aplicação dentro do sistema. A minimização do *turnaround* é um exemplo de objetivo de escalonamento. O termo *flow time* também tem sido utilizado na literatura para representar o tempo de uma aplicação J_j no sistema. Assim, determina-se o *flow time* F_j por $(C_j - a_j)$, onde C_j é o tempo de completude e a_j é o momento de chegada da aplicação. Os termos *slowdown* e *stretch* são utilizados por diversos autores na literatura para determinar o *flow time* normalizado. Assim, tem-se o *slowdown*, ou *stretch*, como $(\frac{F_j}{p_j})$, onde p_j representa somente o tempo de processamento, não os demais atrasos, que são considerados em F_j. Como resultado deste cálculo, tem-se a medida de quanto tempo a mais a aplicação permaneceu no sistema além do tempo de processamento efetivo, ou seja, o tempo sem atrasos de fila e escalonamento. Um exemplo de objetivo de escalonamento é a minimização do *stretch* do sistema.
- **demora do escalonamento**: leva-se um intervalo de tempo entre a chegada de aplicações e a inicialização delas no sistema. Esse tempo pode ser utilizado pelo escalonador como uma métrica para, por exemplo, determinar a máquina com melhor tempo de inicialização e assim reduzir o *turnaround* das aplicações.
- **número de aplicações aceitas**: o escalonador pode rejeitar a aplicação caso não concorde com seus termos. Isso ocorre frequentemente em nuvem. O aumento do número de aplicações aceitas é um exemplo de métrica. Em HPC, aceita-se frequentemente todas as aplicações;
- **número de violações de SLO**: as violações podem ocorrer quando o tempo limite de processamento acordado no SLO é excedido, quando não estão disponíveis os processadores acordados e outras causas. Um exemplo de objetivo de escalonamento é a redução do número de violações. No contexto de violação por tempo limite de processamento, usa-se comumente o termo fator de folga (ou *slack factor*). O fator de folga de uma aplicação identifica o número de vezes que o seu tempo de processamento pode ser estendido sem que ultrapasse um tempo limite conhecido. Assim, determina-se o fator de folga por $(\frac{SLO_{j,i}}{p_j})$, onde $SLO_i \in \mathcal{SLO}$, sendo SLO_i o máximo tempo de resposta acordado para a aplicação J_j e p_j o tempo de processamento da aplicação.

3.4.2 Representação de estratégia de escalonamento

Uma sugestão de representação de estratégia de escalonamento chamada de *notação de três campos* foi proposta por [Graham et al., 1979] e consiste basicamente em determinar $\alpha|\beta|\gamma$. O símbolo α representa o ambiente de processamento, β representa as características da aplicação e γ representa o critério de otimização. Para exemplificar, um escalonamento em P processadores com preempção (abreviado como *pmnt*) onde o objetivo é minimizar o máximo tempo de completude de um conjunto de aplicações toma como forma a expressão $P|pmnt|C_{max}$.

Outra sugestão de representação foi introduzida por [Tchernykh et al., 2001], chamada *(a,b,c)-Scheme*. Essa é uma unificação de representação de aplicações sequenciais e aplicações que requisitam mais de um processador. A granularidade da aplicação é adaptada tendo como base as características dos três parâmetros do esquema: a para o sistema, b para a aplicação e c para as características da estratégia de escalonamento.

Em geral, há muitas variáveis em sistemas paralelos que podem impactar no desenvolvimento e representação de estratégias de escalonamento. Latências de comunicação, sincronização, preempção e outros fatores adicionam consumo de tempo extra para aplicações e podem ser difíceis de serem mensurados e representados matematicamente. Existe um desejo da comunidade científica em simplificar a complexidade da representação matemática dessas latências. Os autores [Tchernykh et al., 2009] comentam que nos casos onde há fatores díficeis de serem representados, eventualmente porque envolvem muitas variáveis ou são desconhecidos, procura-se definir o *fator de penalidade* do contexto. O fator de penalidade pode ser considerado como a taxa do *speedup* real sobre o descoberto analiticamente. O *speedup* sobre P processadores é o C_j da aplicação quando executado em P processadores dividido pelo C_j quando executado em apenas um processador. O *speedup* também pode ser estimado por meio de análise empírica ou *benchmarking*. Usualmente, o fator de penalidade é ignorado nas análises quando o *speedup* é linear ou menor que um. Quando as latências extras são bem menores que as variáveis conhecidas, elas podem, portanto, serem desconsideradas sem perda de representatividade. Segundo [Tchernykh et al., 2009], isso tem sido observado em vários sistemas de computação em larga escala .

Uma teoria amplamente abordada na literatura consiste em utilizar o modelo determinístico com análise de tempo de processamento de aplicações para obter a taxa de desempenho da estratégia. No escalonamento *offline* de tarefas sequenciais, a taxa de desempenho é conhecida por $\rho = \frac{caso_{pior}}{caso_{melhor}}$, onde $caso_{melhor}$ é o melhor caso obtido e $caso_{pior}$ é um resultado não ótimo obtido, já que pode não ser possível garantir que esses casos sejam o melhor ou pior entre todos os casos possíveis, muitas vezes desconhecidos. Dessa maneira, comparações entre escalonadores são feitas por um *fator de aproximação* utilizando a melhor taxa de desempenho obtida do melhor algoritmo conhecido e a taxa de desempenho da nova estratégia. O termo *aproximação* é substituído por *competitividade* quando o escalonamento é *online*.

3.4.3 Escalonamento em lista

O escalonamento em lista $P||C_{max}$ elaborado por [Graham, 1966] possui a melhor taxa de desempenho para escalonamentos *offline* de tarefas sequenciais conhecida na literatura, que é $\rho = 2 - \frac{1}{m}$, onde m é o número de processadores. Essa taxa tem sido utilizada como referência para novas estratégias que procuram minimizar o C_{max}. O pior caso de C_{max} (ou C_{pior}) é obtido quando a tarefa de maior tempo de processamento é escalonada após as demais tarefas. Considere W_{total} ser a soma do trabalho de todas as tarefas, W_{ocioso} ser o tempo ocioso e p_{max} o maior tempo de processamento de tarefa. Assim, tem-se $C_{pior} = \frac{W_{total}+W_{ocioso}}{m}$. O melhor caso de C_{max} (ou C_{melhor}) é obtido quando as tarefas maiores são escalonadas primeiro. Assim, tem-se

$C_{melhor} = max\{\frac{\sum_{i=1}^{n} p_j}{m}, p_{max}\}$. Por fim, por meio da resolução da equação $\rho = \frac{C_{pior}}{C_{melhor}}$, tem-se $\rho = 2 - \frac{1}{m}$.

Uma ilustração dos casos pior e melhor desse escalonamento é apresentada na Figura 3.5. Considera-se o conjunto de tarefas $\{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$, onde T_7 requisita 3 unidades de tempo de processamento e as demais requisitam apenas 1 unidade. Aplicando-se $\rho = \frac{C_{pior}}{C_{melhor}}$ com $m = 3$, tem-se $\rho = \frac{5}{3} = 1,66$. Este mesmo resultado é preferencialmente obtido por meio da equação universal $\rho = 2 - \frac{1}{m} = 1,66$. Entre outras palavras, obtem-se a taxa ρ do escalonamento em lista conhecendo apenas o número de processadores m.

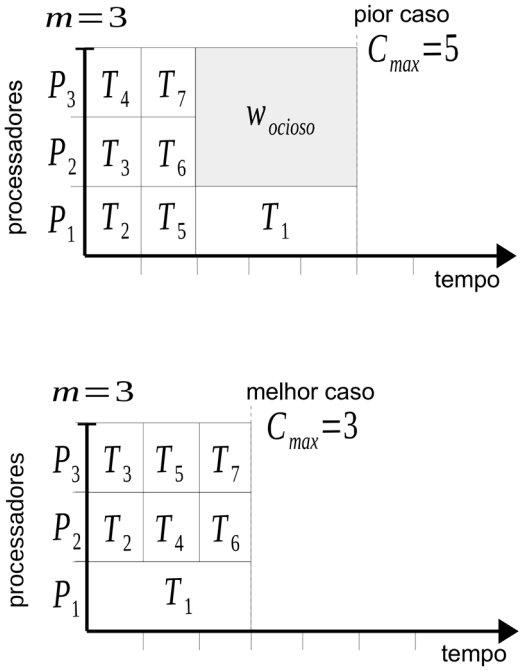


Figura 3.5: Casos pior e melhor do escalonamento em lista [Graham, 1966].

3.4.4 Estratégias clássicas de *backfilling*

Algumas estratégias de escalonamento não identificam casos pior e melhor e, portanto, não podem ser comparadas por taxa de desempenho. Essas estratégias frequentemente envolvem muitos mecanismos e são demasiadamente complexas para representação e análise matemática. Na literatura, essas estratégias são também chamadas de *heurísticas*. Exemplos de heurísticas são os mecanismos de *backfilling*.

O conceito fundamental de *backfilling* consiste em alocar tarefas nos espaços de tempo ocioso do escalonamento. A ociosidade pode ocorrer pela imperfeição do escalonamento ou pelo término de tarefas mais cedo que o previsto. Destacam-se dois algoritmos clássicos de *backfilling* na literatura, que são: *Extensible Argonne Scheduling sYstem* (EASY) [Lifka, 1995] e *Conservative Backfilling* [Gibbons, 1997]. No EASY *backfilling*, as aplicações são organizadas pelo menor tempo de término previsto. Em seguida, identificam-se espaços ociosos (ou *slots* de tempo ocioso) entre as aplicações escalonadas. Processadores extras são identificados quando encerra-se a busca por *slots* de tempo ocioso nos processadores com aplicações escalonadas. Finalmente, cada aplicação na fila de espera do escalonador é consultada sobre sua adequação de escalonamento nos *slots* de tempo ocioso ou nos processadores extras. Por outro lado, no

Conservative Backfilling, a estratégia consiste em encontrar o primeiro espaço ocioso para cada aplicação na fila de espera. As demais aplicações já escalonadas podem ser reescalonadas quando aparecem *slots* de tempo ocioso por ocorrência de término mais cedo que o previsto.

3.5 Trabalhos relacionados em escalonamento

Destacam-se nesta seção trabalhos relativos a estratégias de escalonamento em HPC, eventualmente com seus fatores de aproximação ou de competitividade. Esses trabalhos foram selecionados objetivando-se a identificação de características de aplicações, processadores, mecanismos e métricas adotadas. Um segundo passo consiste na formulação de uma proposta de escalonadores e estratégia de escalonamento, tendo como base alguns conceitos observados nesses trabalhos. A proposta é apresentada no próximo capítulo.

Torna-se relevante distinguir o mecanismo conhecido como *dois níveis* e o mecanismo chamado de *duas fases* no contexto de estratégias de escalonamento. Entende-se por mecanismo de dois níveis o modelo envolvendo controle de admissibilidade, que consiste basicamente em selecionar máquinas candidatas primeiro e depois aplicar a estratégia do escalonamento [Tchernykh et al., 2010]. O contexto completo do mecanismo dois níveis está representado na Figura 3.6; a Figura 3.7 representa um caso de admissibilidade. Enquanto isso, entende-se por duas fases a estratégia para aplicações dentro do algoritmo de escalonamento. Essa estratégia foi desenvolvida por [Phillips et al., 1997] e consiste em escalonar primeiro as aplicações com requisito de um processador.

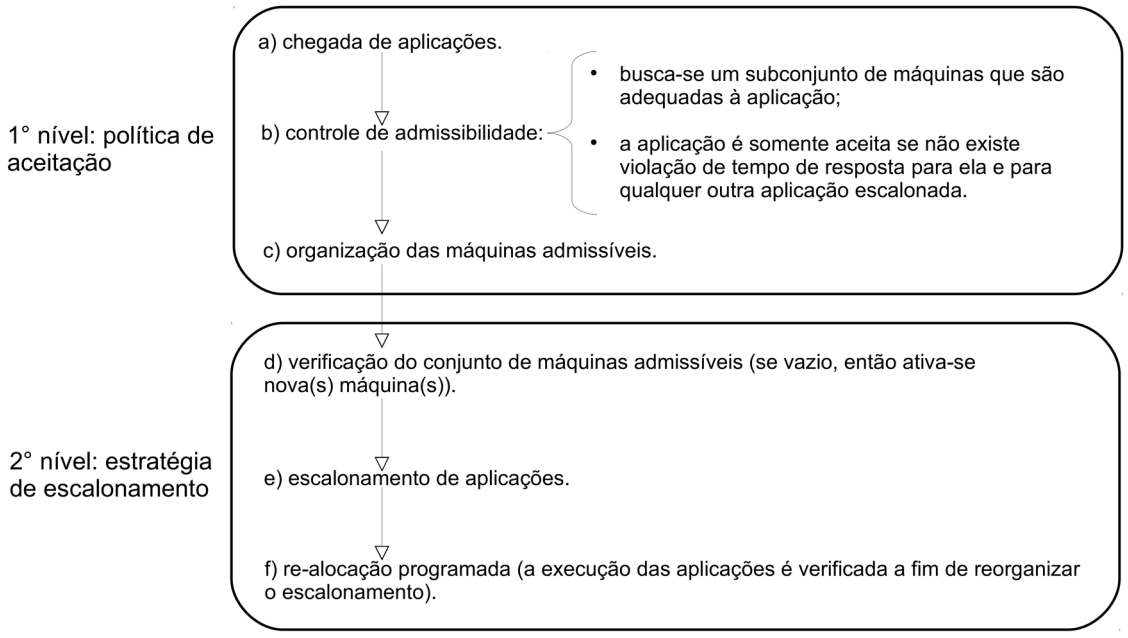


Figura 3.6: Mecanismo chamado de *dois níveis* no contexto de escalonadores.

Um escalonamento *online*, preemptivo sem migração, com regulação da ociosidade, usando estratégia de duas fases, foi apresentado por [Tchernykh et al., 2009] com o objetivo de minimizar o *makespan*. Essa estratégia consiste em escalonar aplicações paralelas rígidas em máquinas paralelas homogêneas. O princípio de escalonamento *online* considera que um processador não pode estar ocioso se existe alguma aplicação para ser executada. Os autores apresentam que a eficiência aumenta quando aplicações com muita exigência de processamento são alocadas separadamente. Assim, o *makespan* pode ser minimizado.

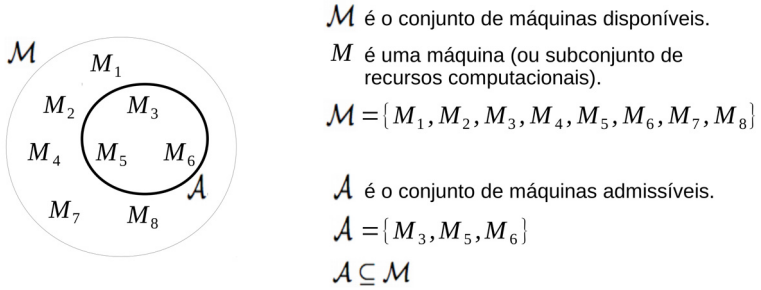


Figura 3.7: Uma sessão de admissibilidade de máquinas no nível de políticas de aceitação do mecanismo de dois níveis.

Uma estratégia de escalonamento *online* preemptiva sem migração foi apresentada por [Becchetti et al., 2000], onde o objetivo é minimizar a média do *stretch* para aplicações paralelas em máquinas paralelas homogêneas. Segundo esses autores, a taxa de competitividade de qualquer escalonamento *online* para minimizar a média do *stretch* em 2 processadores é pelo menos $1,19$, quando comparado com o melhor caso *offline* sem migração. Essa taxa passa para $1,26$ quando o escalonamento é *offline* com migração. A preempção foi destacada como algo essencial para esses escalonamentos.

Um escalonamento *online* não preemptivo para minimizar o *max-flow* foi apresentado por [Bender et al., 1998]. A taxa de desempenho desse escalonamento é $\frac{3-2}{P}$-competitividade, considerando que aplicações são escalonadas em máquinas homogêneas conforme sua ordem de chegada. Esses autores também apresentam várias outras estratégias envolvendo preempção.

Um escalonamento *offline* preemptivo para minimizar o *stretch* foi apresentado por [Bender et al., 2004]. Para qualquer constante $\epsilon > 0$ o algoritmo alcança uma taxa de $1 + \epsilon$-aproximação em máquinas homogêneas. Essa taxa de aproximação é melhorada obtendo fator 2 com uso de escalonamento *online*, considerando o SRPT (*Shortest Remaining Processing Time* - Tempo de Processamento Restante mais Curto).

Um escalonamento *offline* determinístico $P|precs, p_j = 1|C_{max}$ foi proposto por [Rodriguez et al., 2003], cuja taxa de desempenho $2 - (\frac{1}{m-1})$ com $m \geq 3$. Essa estratégia consiste em organizar os primeiros sucessores imediatos e o máximo número de primeiros sucessores imediatos. Consideram-se máquinas paralelas homogêneas e aplicações paralelas com dependência. Os autores [Braschi e Trystram, 1994] apresentaram um algoritmo com uma taxa um pouco melhor, que é $2 - (\frac{2}{m}) - (\frac{(m-3)}{m \times C_{max}})$.

O trabalho de [Dutot et al., 2004] aborda diversas estratégias de escalonamento para tarefas paralelas, sendo elas:

- tarefas moldáveis independentes, C_{max}, *offline*, cuja melhor taxa de desempenho tem sido alcançada por [Mounié, 2000] com $\frac{3}{2} + \epsilon$;
- tarefas moldáveis com precedência, C_{max}, *offline* com taxa de desempenho $\frac{3+\sqrt{5}}{2}$;
- tarefas moldáveis independentes, C_{max}, *online* com taxa de desempenho $3 + \epsilon$;
- tarefas maleáveis independentes, C_{max}, *online* com taxa de $1 + \phi \approx 2,16180$;
- tarefas moldáveis independentes, $\sum C_j$, *offline* com taxa de desempenho 8 em casos sem peso e 8.53 em casos com peso ($\sum w_j \times C_j$) alcançadas com o Algoritmo de [Schwiegelshohn et al., 1999] para minimizar a média do tempo de resposta.

- tarefas moldáveis independentes, com dois critérios, *online* com apresentação de um algoritmo genérico com garantias para ambos os critérios C_{max} e $\sum C_j$, embora uma solução ótima são seja possível para ambos. Utiliza-se o algoritmo de duas fases. A taxa de desempenho para o escalonamento com dois critérios é de $2\rho \sum C_j$ e $2\rho C_{max}$ ao mesmo tempo. Ou seja, a taxa de desempenho é o dobro da taxa dos algoritmos de cada critério.

Em caso de utilização de máquinas heterogêneas, os autores [Dutot et al., 2004] apresentam que o tempo de processamento de uma aplicação não depende do número de processadores alocados, mas sim das diferentes características dos processadores selecionados. A Tabela 3.1 resume as principais características das estratégias de escalonamento estudadas nesta seção.

Tabela 3.1: Principais características das estratégias de escalonamento estudadas

Autores	**Estratégia**	**Tipo de máquina**	**Tipo de aplicação**	**Objetivo**	**Aprox./Competitividade**
[Graham, 1966]	*offline*	paralelas homogêneas	rígidas	minimizar o C_{max}	$2 - \frac{1}{m}$
[Tchernykh et al., 2009]	*non-clairvoyant*, preemptivo, sem migração	paralelas homogêneas	rígidas	minimizar o C_{max}	-
[Becchetti et al., 2000]	*online*, preemptivo, sem migração	paralelas homogêneas	rígidas	minimizar a média do *stretch*	1, 19
[Bender et al., 1998]	*online*	paralelas homogêneas	sequenciais	minimizar o *max-flow*	$\frac{3-2}{P}$
[Bender et al., 2004]	*offline*, preemptivo	paralelas homogêneas	sequenciais	minimizar o *stretch*	$1 + \epsilon$
[Rodriguez et al., 2003]	*offline*	paralelas homogêneas	com precedências	minimizar o C_{max}	$2 - (\frac{1}{m-1})$
[Braschi e Trystram, 1994]	*offline*	paralelas homogêneas	com precedências	minimizar o C_{max}	$2 - (\frac{2}{m}) - (\frac{(m-3)}{m \times C_{max}})$
[Mounié, 2000]	*offline*	paralelas homogêneas	moldáveis e independentes	minimizar o C_{max}	$\frac{3}{2} + \epsilon$
[Dutot et al., 2004]	*offline*	paralelas homogêneas	moldáveis com precedências	minimizar o C_{max}	$\frac{3+\sqrt{5}}{2}$
[Dutot et al., 2004]	*online*	paralelas homogêneas	moldáveis independentes	minimizar o C_{max}	$3 + \epsilon$
[Dutot et al., 2004]	*online*	paralelas homogêneas	maleáveis independentes	minimizar o C_{max}	$1 + \phi \approx 2, 16$
[Dutot et al., 2004]	*offline*	paralelas homogêneas	moldáveis independentes	minimizar o C_{max}	8
[Dutot et al., 2004]	*online*	paralelas homogêneas	moldáveis independentes	minimizar o C_{max} e $\sum C_j$	$2\rho C_{max}$ e $2\rho \sum C_j$

Como base nessa tabela, observa-se que o maior número de trabalhos objetiva minimizar o C_{max}. Devido a complexidade das análises, esses trabalhos focam em processadores homogêneos e aplicações mais simples, tais como as aplicações com tarefas sequencias e independentes.

3.6 Trabalhos relacionados em escalonamento no Google

Nesta seção são apresentadas características do escalonador do Google, a estratégia de escalonamento utilizada e estatísticas sobre aplicações executadas. Devido a sua heurística, não foi possível determinar da taxa de desempenho da estratégia. A Central de Dados do Google possui 3 tipos de aglomerado de máquinas classificados de acordo com a capacidade de seus recursos, tais como capacidade de processamento e capacidade de memória principal [Liu e Cho, 2012a]. Cada aglomerado tem uma visão geral do estado dos outros aglomerados. Para esse mecanismo dá-se o nome de *visão compartilhada* [Schwarzkopf et al., 2013]. O escalonador de um aglomerado separa as aplicações por prioridade e envia-as para escalonadores de máquinas mais adequadas aos requisitos de cada aplicação. Em geral, existem muitas aplicações de baixa prioridade e menor número aplicações de alta prioridade. Quanto maior a prioridade, mais importante é a aplicação. Em cada aglomerado existe uma partição com maior número de processadores para aplicações de maior prioridade e uma partição menor para aplicações de menor prioridade. No mais baixo nível da arquitetura do sistema operacional que recebe as aplicações, cada aplicação tem seu próprio *container* Linux onde é permitido criar multiplos subprocessos [Reiss et al., 2011]. Cada um desses subprocessos atende uma tarefa de aplicação. O *container* permite a isolação da aplicação em relação a demais e com isso é possível também contabilizar o uso dos recursos. Habilita-se dessa forma um modelo de negócio entre a Central e seus usuários.

O fluxo de controle de aplicações do Google está ilustrado na Figura 3.8, tendo como fonte os trabalhos [Reiss et al., 2011] e [Liu e Cho, 2012a]. Observa-se nessa figura que o escalonador administra aplicações de 2 filas, existem 4 estados de aplicação e 9 eventos envolvidos. A fila principal (chamada de *fila de aplicações pendentes*) é do tipo FCFS (*First-Come-First-Served* - Primeiro que Chega - Primeiro a ser Atendido) e mantém aplicações em espera de escalonamento.

O primeiro estado de uma aplicação é *não-submetida*, ou seja, a aplicação ainda não está inserida no ambiente do escalonador. Após a submissão (ou evento *submit*) à Central de Dados, a aplicação é armazenada na fila principal. O escalonador coleta aplicações dessa fila e elabora o escalonamento (ou evento *schedule*). As tarefas das aplicações são processadas (ou estado *executando*) no tempo programado e podem ser preemptadas com troca de contexto ou migração. Cada máquina mantém uma lista das tarefas de aplicações em execução e com isso o gerenciador de carga de trabalho pode atualizar a configuração (ou *update-running*) dos recursos que estão sendo utilizados. Em outras palavras, esse é o momento onde aplica-se a elasticidade. Por fim, a aplicação passa para o estado *encerrada*, quando um dos seguintes eventos ocorre:

- ***evict***: ocorre quando uma preempção é necessária;
- ***kill***: ocorre devido à impossibilidade de preempção;
- ***fail***: ocorre por falhas do código da aplicação durante a sua inicialização ou durante a sua execução;
- ***lost***: ocorre quando os dados necessários ao processamento não puderam ser obtidos;
- ***finish***: ocorre quando a aplicação foi encerrada.

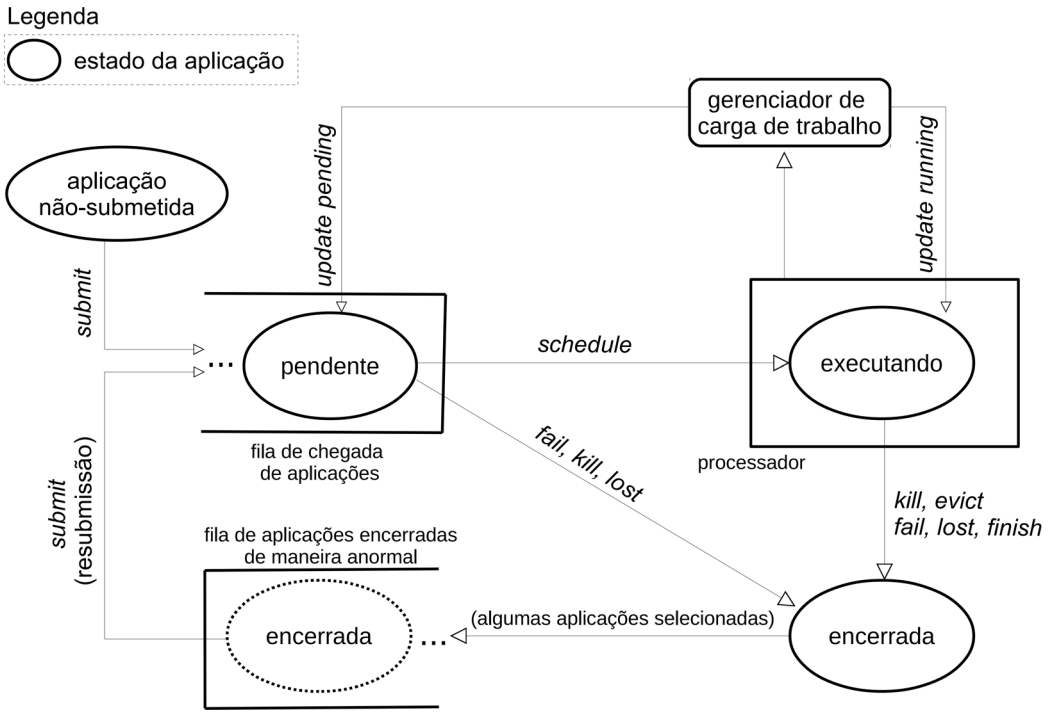


Figura 3.8: Fluxos de controle do escalonador de aplicações do Google, com base nos estudos de [Reiss et al., 2011] e [Liu e Cho, 2012a].

Os recursos requisitados pela aplicação podem ser reconfigurados (evento *update-pending*) enquanto ocorre a espera em fila. Uma aplicação pode ser encerrada devido à anormalidades previstas nos eventos *fail*, *evict*, *kill* e *lost*. No evento *evict* ocorre o despejo da aplicação, devido a sua prioridade. De acordo com [Reiss et al., 2011], no Google existem 5 níveis de prioridade (*pri*). Destaca-se cada um desses níveis a seguir, juntamente com informações sobre as aplicações:

- ***grátis*** ($0 \leq pri \leq 1$): aplicações com esta prioridade geralmente possuem curto tempo de processamento (menor que 3 minutos); não há alteração de requisitos durante sua execução; são do tipo *single-task* ou *sequential-task* e são geralmente independentes. Essas aplicações são as mais encerradas pela troca de contexto da preempção e devido a isso elas são frequentemente reescalonadas.

- ***outras*** ($2 \leq pri \leq 8$): aplicações com esta prioridade são do tipo *batch-task*, criadas apartir de aplicações científicas e *WebApps*. Em máquinas virtuais elas têm tempo de vida médio de 24 horas, sendo permita a migração. Essas aplicações são as que mais falham.

- ***produção*** ($p = 9$): as aplicações aqui são longas e representam serviços de longa duração com arcabouços instanciados para atender as demais aplicações.

- ***monitoramento*** ($pri = 10$): aplicações com esta prioridade são criadas para monitorar outras aplicações.

- ***infraestrutura*** ($pri = 11$): essas aplicações representam serviços de armazenamento e indexação de documentos.

Na Central de Dados do Google, 50% das tarefas (não aplicações) têm baixa prioridade e 30% de todo o conjunto de tarefas é encerrado anormalmente pelo menos uma vez [Reiss et al., 2012]. As aplicações *grátis* são mais despejadas que as aplicações de *produção*, devido à prioridade atribuída. As razões para o encerramento considerado anormal incluem a preempção para privilegiar tarefas de maior prioridade, esgotamento da quantidade de memória RAM no momento da execução, falhas de *hardware*, desistência dos usuários e interrupções para manutenção de máquinas [Sheng et al., 2013]. Considerando as aplicações despejadas que não foram reescalonadas, os autores [Hemmat e Hafid, 2016] estimaram que o número de violações no Google é de 2,2%. Entretanto, neste número estão inclusos todos os tipos de interrupção de execução, tal como a desistência do usuário, sendo, portanto, um número possivelmente maior que o real. Em contextos analisando-se apenas violações por de falta de disponibilidade de recursos, esses mesmos autores afirmam que o percentual de violações em plataformas de computação é em geral 0,2%.

A fim de apresentar dados estatísticos relativos a aplicações no Google, classificam-se a seguir alguns trabalhos de acordo com:

- *resultados sobre execução de aplicações*: 41% das aplicações possuem prioridade *grátis*, 52% possuem prioridade *não-produção* ($2 \leq pri \leq 8$) e 7% possuem prioridade *produção* ($9 \leq pri \leq 11$) [Abdul-Rahman e Aida, 2014]; 50% das aplicações têm baixa prioridade [Reiss et al., 2012].

- *resultados sobre uso dos recursos por aplicações*: menos de 6% de todas as aplicações contabilizadas utilizam cerca de 95% das CPUs e 90% da memória total disponível, o uso dos recursos é dominado pelas aplicações de prioridade *produção* [Abdul-Rahman e Aida, 2014]; 75% das aplicações têm uma tarefa e são curtas, consumindo recursos por cerca de 3 minutos, 2% das aplicações utilizam 80% dos recursos e $\frac{2}{3}$ dessas aplicações são longas [Reiss et al., 2012]; 17,4% das tarefas requisitam menos CPU e 1,9% requisitam menos memória RAM que o necessário, o que caracteriza uma boa previsão de recursos [Iglesias et al., 2014]; o número de núcleos de processamento requisitados está entre 1 e 4, o uso da memória RAM está entre 0,1 e 8 GBytes [Mishra et al., 2010]; 40% das tarefas *grátis* têm o mesmo requisito de CPU e memória, mais de 50% das tarefas possuem menos que 100 segundos de tempo de processamento [Zhang et al., 2014]; 64% das aplicações têm uma simples tarefa [Sheng et al., 2013].

- *resultados sobre eventos de terminação (*kill*,* lost*,* fail *e* evicted*)*: 73% das tarefas são marcadas com término com sucesso, 40% da alocação de memória é destinada para execuções que não completam sua trajetória [Reiss et al., 2012].

3.7 Trabalhos relacionados em escalonamento com SLA

O tema de escalonamento considerando SLA não é exaustivo, haja visto que existem diversos tipos de aplicação, métricas e métodos envolvendo sistemas de computação em larga escala. Busca-se identificar a seguir quais SLOs, métricas e observações foram apresentados na literatura envolvendo escalonamento. Esses dados são úteis para a delimitação da proposta apresentada no próximo capítulo.

Uma heurística de escalonamento para maximização de uso de recursos considerando termos de SLA foi proposta por [Yarmolenko e Sakellariou, 2006]. Nessa heurística, os seguintes SLOs são considerados: menor momento de inicialização T_S; momento mais tardio de encerramento T_F; tempo reservado para execução da aplicação T_D; e número de CPUs requisitadas

N_{CPU}. A aplicação é mensurada em horas de uso de CPU $N_{CPU} \times T_D$. A heurística proposta consiste em encontrar a melhor combinação de valor para a aplicação.

Um escalonamento para maximizar o número de aplicações aceitas considerando SLA em nuvens IaaS foi proposto por [Barquet et al., 2013]. Os autores apresentam que quanto maior é o fator de folga de tempo de processamento, maior é o número de aplicações aceitas. O modelo SLA é definido por $S_i = (f_i, u_i)$, onde f_i é o fator de folga considerando o tempo de processamento p_j, e u_i é o preço de uso de uma unidade de tempo. A competitividade da heurística é definida como $c_v = \frac{\sum_{j=1}^{n} u_i \times p_j}{V(A*)} \leq 1$, onde $V(A*)$ é o melhor caso, que não é conhecido.

Um escalonamento para maximizar o uso de CPUs em nuvens IaaS foi proposto por [Costache et al., 2013]. O tempo limite de processamento de uma aplicação é a propriedade SLO considerada nesta abordagem. As máquinas são classificadas em níveis de unidade de tempo de acordo com o SLO, tais como 100, 150, 200 e outros. A elasticidade toma como base o tempo restante de processamento, o tempo limite acordado e o número restante de interações. Dado um conjunto de máquinas virtuais $b_i(t)$ ofertadas, um intervalo de tempo t e um recurso de processamento com capacidade disponível de C unidades, o escalonador aloca uma quantidade de recursos a_i para cada máquina virtual i. Nessa estratégia as máquinas virtuais são valorizadas com base nas suas ofertas. Assim, $a_i(t) = \frac{b_i(t)}{p(t)}$, $p(t) = \frac{\sum_{i=1}^{n} b_i(t)}{C}$, onde $p(t)$ é o preço de uso de uma unidade de tempo do recurso. Complementarmente, cada usuário possui uma carteira onde os custos são debitados. Quando o preço de uso de um recurso está aumentando, as tarefas de uma aplicação podem ser preemptadas e migradas para recursos mais baratos.

Um escalonamento para identificação de quais aplicações dão maior retorno para o provedor foi feita por [Yeo e Buyya, 2005]. Para uma aplicação i em uma máquina j, o retorno é definido por $return_{ij} = \frac{\frac{utility_{ij}}{runtime_i}}{deadline_i}$, onde $utility_i = budget_i - (delay_i \times penalty\_rate_i)$, $delay_i = (finish\_time_i - submit\_time_i) - deadline_i$. O valor do recurso requisitado é representado por $utility_{ij}$, o tempo de processamento da aplicação por $runtime_i$, o tempo máximo para completar a aplicação por $deadline_i$, o valor máximo que o usuário está pagando para completar a aplicação é representado como $budget_i$, atrasos são representados por $delay_i$, e a flexibilidade dos atrasos por $penalty\_rate$. Um atraso ocorre quando o compartilhamento total dos recursos excede o tempo máximo de processamento que o processador da máquina j pode oferecer. O compartilhamento é definido por $share_{ij} = \frac{remain\_runtime_{ij}}{remain\_deadline_{ij}}$ e o total do compartilhamento por $\sum_{i=1}^{n} share_{ij}$. Conclusivamente, o $return_j = \sum_{i=1}^{n} return_{ij}$ indica se a máquina j está ou não está sobrecarregada com muitas aplicações, o que impacta no cumprimento de SLAs.

Um escalonamento para nuvens SaaS com os objetivos de minimizar custo e minimizar violações de SLA foi proposto por [Wu et al., 2011]. O algoritmo apresentado nesse trabalho identifica onde está o recurso com menor custo. Assim, reduz-se o número de violações SLA pelo uso de uma nova máquina virtual com garantias de tempo de resposta para cada Companhia. O custo é definido por $Cost_{il}^{c} = VMCost_{il}^{c} + PenaltyCost_{il}^{c}$, onde c é o cliente, i é a identificação de um recurso requisitado pelo cliente e l o seu tipo. O $VMCost_{il}^{c}$ é definido por $PriVM_{il} \times (iniTimeSev_{il}^{c} \times conLen^{c})$, onde $PriVM_{il}$ é o preço incluindo equipamento físico, energia, rede de comunicação e o custo dos serviços administrativos. O tempo que leva para iniciar uma máquina virtual é representado como $iniTimeSev$. Por fim, o $conLen$ é o tamanho do contrato, medido em horas, dias ou meses. Existe uma penalidade $PenaltyCost_{il}$ que decorre de um tempo de atraso $delayTime_{il}$. A penalidade é definida por $\alpha + \beta(reqType) \times delayTime_{il}^{c}(reqType)$,

onde α é um número constante, β é a taxa de penalidade e $reqType$ representa um tipo de solicitação.

Coletadas as solicitações do cliente e as características de uma infraestrutura de TI, [Hedwig et al., 2012] apresentam uma estratégia de escalonamento para encontrar o preço mínimo de um serviço. Nessa estratégia o SLO é o máximo tempo de resposta tolerável. Os autores também determinam o risco do serviço considerando o custo das operações e a probabilidade delas concordarem com o SLA. Um acordo SLA não precisa ser cumprido na sua totalidade. Por exemplo, 90% das submissões devem estar de acordo com seus respectivos SLOs. Uma abordagem similar foi proposta por [Breitgand et al., 2007], com foco na busca por um tempo de resposta melhor sob uma dada infraestrutura computacional. Nessa abordagem existe uma carteira de crédito do usuário onde os gastos envolvendo uso de recursos são debitados.

Apresenta-se na Tabela 3.2 um resumo dessas estratégias envolvendo SLA. Destaca-se que não existe taxa de desempenho para essas heurísticas. Observa-se que as principais decisões dessas heurísticas consideram o tempo de resposta de aplicação.

Tabela 3.2: Características das estratégias de escalonamento envolvendo SLA estudadas

Autores	Contexto de aplicação	Objetivo da heurística	SLOs considerados
[Yarmolenko e Sakellariou, 2006]	grids	maximizar o uso de recursos	momento de inicialização, momento mais tardio de encerramento, tempo reservado para execução da aplicação, número de CPUs
[Barquet et al., 2013]	nuvens IaaS	maximizar o número de aplicações aceitas	fator de folga de tempo de processamento, preço de uso de uma unidade de tempo
[Costache et al., 2013]	nuvens IaaS	maximizar o uso de CPUs	máximo tempo de processamento
[Yeo e Buyya, 2005]	aglomerados orientados a serviços	identificar quais aplicações dão maior retorno para o provedor	tempo de processamento, máximo tempo de completude
[Wu et al., 2011]	nuvens SaaS	minimizar custo e minimizar violações	máximo tempo de resposta
[Hedwig et al., 2012]	qualquer infraestrutura de TI	encontrar o preço mínimo para um serviço	máximo tempo de resposta

3.8 Desafios em aberto

Muitos trabalhos sobre escalonamento de aplicações foram publicados desde o escalonamento em lista de [Graham, 1966]. Até o final da década de 90, a grande maioria das novas estratégias de escalonamento foram desenvolvidas para sistemas de computação em larga escala com objetivo de atender cenários HPC, como é a maior parte dos trabalhos destacados na Seção 3.4.

Há uma crescente busca por otimizações em computação em nuvem e muitas propostas de HPC têm sido utilizadas como base para isso, haja visto que a mesma infraestrutura computacional pode ser utilizada por ambas as plataformas, como discutido na Seção 2.2. Como exemplificação, os autores [Pawar e Wagh, 2013] e [Raju et al., 2013] desenvolveram uma estratégia para minimizar o tempo de completude de tarefas em nuvem. Diversos outros trabalhos

foram destacados por [Singh e Singh, 2013] em um levantamento da literatura. Não menos importante, algumas inovações trazidas pelo modelo de negócio em nuvem foram transportadas para HPC, como é o caso do trabalho de [Koller, 2010] sobre a implantação de SLA para garantir requisitos do usuário. A fim de destacar alguns desafios em aberto na temática de escalonamento de aplicações, lista-se a seguir um levantamento feito com base em publicações recentes na área:

- *Escalonamento considerando extensão de recursos locais para dentro da nuvem*: sistemas de computação externos à Central de Dados, como aplicações em aglomerado de máquinas privadas, ou aplicações comerciais com poucas máquinas dedicadas, podem migrar suas aplicações para dentro de uma nuvem como forma de estender seus recursos locais. Devido à cobrança pelo serviço da nuvem contratado, os sistemas externos buscam escalonar o máximo de aplicações nos processadores locais e migrar o mínimo possível para a nuvem. Os trabalhos de [Chauhan e Babar, 2012, Sefraoui et al., 2015, Chopra e Singh, 2013] exploraram este tema, que não é exaustivo, pois existe uma diversidade de métodos que podem ser envolvidos para elaborar novas estratégias de escalonamento.

- *Cooperação e escalonamento em sistemas federados*: a cooperação entre grids como forma de estender recursos locais tem sido um mecanismo bastante explorado em arquiteturas de sistemas de computação em larga escala. Usualmente, a migração de aplicações entre grids é acordada antecipadamente pelo RJMS de ambos os grids envolvidos. Entre plataformas de nuvem, essa cooperação é importante, pois garante mais recursos para aplicações, a fim de evitar violações de SLA. Complementarmente, os aspectos tecnológicos que definem a cooperação e as circunstâncias vantajosas do escalonamento delineiam uma área de pesquisa desafiadora. Alguns trabalhos relacionados com este tema são apresentados por [Sotiriadis et al., 2012, Perez-Mendez et al., 2014, den Bossche et al., 2010, Simarro et al., 2011].

- *Estratégias multidimensionais envolvendo previsão de tempo de resposta e consumo de energia*: estratégias de previsão de tempo de resposta dependem dos requisitos das aplicações e da infraestrutura da Central de Dados [Pore et al., 2015]. Há diversos mecanismos envolvidos entre a chegada da aplicação e a disponibilização do serviço requisitado por ela. Frequentemente o tempo de resposta é negligenciado pelos atrasos entre os diferentes estados da aplicação no sistema. Observa-se que ocorre consumo de energia durante todo esse processo, principalmente para execução da aplicação. Ao mesmo tempo em que objetiva-se prever e reduzir o tempo de resposta, há uma busca crescente por estratégias de escalonamento de aplicações que considerem também aspectos de redução de consumo de energia. Há um vasto campo para pesquisas nesta área, devido a diversas características de aplicação e contextos de execução. Sugere-se o trabalho de [Pore et al., 2015] para entendimento deste tema.

- *Organização de tarefas de aplicações em máquinas virtuais*: uma ou mais aplicações podem ser relacionadas para cada máquina virtual, cujo número de instâncias tem impacto em várias métricas dentro da Central de Dados, como consumo de energia, tempo de resposta e custos para o usuário. O desenvolvimento de uma estratégia de agrupamento de tarefas de aplicações em um conjunto de máquinas virtuais foi discutido por [Nguyen et al., 2013].

- *Problemas envolvendo tempo de resposta de aplicação*: há diversos trabalhos na literatura procurando otimizar métricas que envolvem tempo de res-

posta, como os citados no Capítulo 3.4. Mesmo em HPC, com muitos trabalhos publicados, essa temática ainda é atual. No contexto de computação em nuvem, onde SLA está fortemente relacionado com garantias de tempo de resposta, surgiram vários trabalhos apresentando novas heurísticas de escalonamento. Exemplos de trabalhos nesse contexto abordando nuvem foram apresentados por [Mao e Humphrey, 2011, Maguluri e Srikant, 2013, Moschakis e Karatza, 2012, Patel e Bhoi, 2014, Tammaro et al., 2011, Wang et al., 2014, Xiang et al., 2013].

- *Compartilhamento da arquitetura de memória* cache *por diferentes* threads *de aplicação*: a reserva de recursos computacionais para aplicações tem sido feita usualmente por gerenciadores/aplicadores de Qualidade de Serviço do RJMS. Entretanto, no mais baixo nível da arquitetura computacional, o impacto sobre aplicações em execução devido ao uso compartilhado do processador não é um problema bem definido matematicamente na atualidade. Um fator para isso é que não há mecanismo conhecido até o momento capaz de reservar memória *cache*. Um trabalho relacionado com este tema foi apresentado por [gyu Kim et al., 2012].

3.9 Discussão

Neste capítulo foram apresentadas uma visão detalhada de aplicações e tarefas; partições e processadores; estratégias e representações de escalonamento para aplicações em sistemas de computação em larga escala. Por fim, apresentaram-se trabalhos relacionados e desafios em aberto na área.

Aplicações podem conter mais de uma tarefa, podendo ser escalonadas para execução em mais de processador, de acordo com a flexibilidade do paralelismo e do gerenciamento de memória utilizado. O relacionamento que existe entre as tarefas tem impacto na distribuição delas, pois é necessário considerar as dependências. Destaca-se que algumas aplicações são simples, podem conter apenas uma tarefa, ou quando contêm mais de uma podem ser processadas de maneira sequêncial.

Processadores em sistemas de computação de larga escala são organizados em partições, onde geralmente mantêm-se conjuntos de processadores com características homogêneas, tal como a mesma velocidade de processamento. A maior parte dos trabalhos envolvendo escalonamento de aplicações encontrados na literatura considera processadores homogêneos. A principal razão para isso é que a representação e a validação matemática tornam-se menos complexas. No contexto matemático, busca-se lidar com poucas variáveis, o que diminui a complexidade das análises. Nesse sentido, evita-se a representação de processadores heterogêneos e muitas características de aplicações.

O tempo de resposta de aplicação é amplamente utilizado para determinar métricas em estratégias de escalonamento. Em plataformas de nuvem, as métricas relacionadas com tempo de resposta são preferidas porque envolvem contratos de SLA. Como estudado na Seção de desafios 3.7, trabalhos envolvendo escalonamento e SLA não são exaustivos, pois envolvem muitos elementos. Por essa razão, as análises matemáticas tornam-se demasiadamente complexas, onde o processo de validação não pode mais ser determinado via prova matemática. As estratégias envolvendo nuvem e SLA estudadas visam diminuir o número de recursos por meio de escalonamento do maior número de aplicações possível nesses recursos, ao mesmo tempo em que consideram o número de violações de SLO. Para essas estratégias, também denominadas como heurísticas, não é possível determinar a taxa de desempenho, devido à dificuldade de identificar

o pior e o melhor caso de escalonamento. Frequentemente a validação dessas heurísticas ocorre por meio de simulação.

Identificou-se um tema relevante para pesquisas por meio da contextualização dos conceitos apresentados neste capítulo com os conceitos de centrais de dados apresentados no capítulo anterior. O nível de utilização de recursos como processador e memória principal observados na nuvem Google é alto em relação ao nível de utilização de plataformas HPC [Sheng et al., 2012]. Porém, em nuvem esse nível de utilização não pode ser considerado efetivo, pois existem muitos recursos desperdiçados, como discute-se a seguir. Existe um número significativo de aplicações despejadas em plataformas de nuvem, devido à adoção de prioridades de aplicações e o escalonamento *overbooking*. Aplicações de menor prioridade, após despejadas, ficam consumindo mais recursos computacionais que o previsto. Para ilustrar um caso envolvendo o escalonamento no Google, inicia-se a execução de uma aplicação, consome-se recursos com isso e durante a execução despeja-se a aplicação. Conclui-se que os recursos consumidos até o momento foram desperdiçados, pois a aplicação precisa ser reescalonada e seus dados processados novamente. Preempção de troca de contexto ou seguida de migração sem perda de dados já processados são casos para aplicações de maior prioridade, onde existe alguma relação custo-benefício. Considerando um cenário com significativo número de aplicações de maior prioridade, as aplicações de menor prioridade podem ser reescalonadas mais de uma vez, desperdiçando ainda mais recursos. Nesse contexto com *overbooking*, os SLOs das aplicações são impactados. Segundo [Hemmat e Hafid, 2016], 2,2% do total de aplicações no Google são consideradas como violadas. Esses números são considerados significativos. Dada a variedade de características de aplicações e partições apresentadas neste capítulo, existe um amplo campo para pesquisas objetivando a elaboração de heurísticas de escalonamento que visam diminuir o desperdício de recursos e o número de violações de SLOs.

Alguns mecanismos úteis para elaboração de novas heurísticas foram apresentados neste capítulo. Citam-se a migração de aplicação e o *backfilling*. Também foi observado que plataformas HPC possuem alto nível de ociosidade, podendo ser locais adequados para migração de aplicações vindas de plataformas de nuvem. Um problema desse contexto é o tratamento dado à aplicações de nuvem em HPC e o impacto delas na carga de trabalho da plataforma, haja visto que em HPC não é comum a implantação do mecanismo de SLA e a inserção de mais aplicações pode ampliar o *makespan*. A elaboração de heurísticas nesse tema é tarefa desafiadora e pode trazer contribuições relevantes.

No próximo capítulo detalha-se uma proposta de convergência nuvem-HPC em centrais de dados, onde aplicações podem ser migradas para evitar desperdício de uso de recursos e o escalonamento delas visa reduzir o número de violações de tempo de resposta SLO.

Capítulo 4

Uma Área de Convergência Nuvem-HPC

Em centrais de dados, tal como Google, a sobrecarga de uso de recursos computacionais, tais como processador e memória principal, ocorre devido ao escalonamento *overbooking* de significativo número de aplicações de nuvem [Reiss et al., 2012, Sheng et al., 2013]. O efeito dessa sobrecarga é o aumento do tempo de resposta de aplicações e consequentemente o aumento do número de violações de SLOs.

Processamento HPC também pode ser ofertado como serviço e ter suporte a SLA [Birkenheuer e Brinkmann, 2011]. Isso ocorre frequentemente em nuvem ao ofertar-se HPC. Entretanto, não é comum habilitar SLA quando a plataforma HPC é utilizada no contexto privado de Central de Dados, tal como apresentado na Seção 2.1. Nesse contexto privado, plataformas HPC usualmente aplicam escalonamento em lote e não implementam políticas de negócio. Consequentemente, essas plataformas não monitoram tempo de resposta de aplicações. Um exemplo disso é o Grid'5000 e o seu escalonador OAR, exemplificados na Seção 2.3.

Nos levantamentos de literatura apresentados nas Seções 2.1 e 3.4, observou-se que a ociosidade de uso de recursos em plataformas HPC pode ser aproveitada para diminuir a sobrecarga de escalonamento de aplicações das plataformas de nuvem. Com base nessas considerações, propõe-se neste capítulo uma área de convergência nuvem-HPC, visando a redução do número de violações de tempo de resposta. A caracterização de área de convergência é apresentada na Seção 4.1. Características de aplicações candidatas à migração são discutidas na Seção 4.2. Os fluxos de controle de aplicação dentro da Central de Dados são detalhados na Seção 4.3. As notações dos principais elementos, a representação do sistema base e as funções de previsão de tempo de resposta necessárias para habilitar a área de convergência são descritas respectivamente nas Seções 4.4, 4.5 e 4.6. Por fim, apresentam-se trabalhos relacionados nas Seções 4.7 e 4.8, visando identificar as contribuições desta proposta. Os trabalhos estudados são relacionados em estratégias de previsão de tempo de resposta e estratégias de migração de aplicação. As estratégias dos escalonadores são apresentadas e discutidas no próximo capítulo.

4.1 A área de convergência

Define-se *área de convergência nuvem-HPC* em Central de Dados como um conjunto delimitado m_{conv} de processadores no qual está habilitada a migração de aplicações de plataformas de nuvem para HPC. Uma área de convergência pode ser estática ou dinâmica em termos de número de processadores, podendo envolver processadores de partições de nuvem, partições de HPC e processadores fora dessas partições. No caso de processadores fora de partições, não existe o suporte da plataforma com o sistema base para execução da aplicação, sendo necessário um sistema operacional simplificado que encapsule cada aplicação. O uso de um sistema operacional

simplificado é importante porque evita atrasos de comunicação e inicialização. Um exemplo de aplicação encapsulada por esse tipo de sistema operacional é apresentado por [Kivity et al., 2014]. Segundo esses autores, o tamanho em *bytes* do total do sistema é considerado pequeno e a aplicação possui velocidade de processamento satisfatória, pois executa diretamente no núcleo desse sistema operacional. Para as demais aplicações, a plataforma de destino deve prover o suporte tecnológico.

Os fluxos de controle de aplicação na Central de Dados determinam rotas para aplicações na área de convergência, sendo importantes à tomada de decisão dos escalonadores. Esses fluxos são apresentados na Seção 4.3. Os escalonadores envolvidos possuem funções adicionais que permitem prever o tempo de resposta de aplicações em m_{conv} e tomar decisões sobre migração considerando esses fluxos. Essas funções são apresentadas na Seção 4.6. Mais recursos necessários à tomada de decisão na área de convergência são apresentados nas seções que seguem.

4.2 Aplicações candidatas

Aplicações candidatas são aplicações com potencial para serem migradas na área de convergência e nela obterem menor tempo de resposta. Ao mesmo tempo, deseja-se que essas aplicações contenham características que não impactem demasiadamente no escalonamento da carga de trabalho natural da plataforma receptora. Caracterizam-se aplicações candidatas como aplicações com boa previsão de tempo de processamento e recursos requisitados, com amplo fator de folga de tempo de processamento, com tarefas independentes e não interativas.

A previsão de tempo de processamento é importante porque a maioria das plataformas HPC não aplica o mecanismo de elasticidade, tal como em nuvem, para garantir o tempo de resposta de aplicação pela ampliação da fatia de tempo de processamento definida para cada intervalo no processador. Esse mecanismo foi discutido na Seção 2.2. Frequentemente em plataformas HPC, os recursos alocados não são redimensionados durante a execução da aplicação e a aplicação é encerrada forçadamente ao chegar no término do seu tempo programado. Assim, uma previsão mais curta que o necessário pode implicar na falha da aplicação. Para exemplificar, no Grid'5000 as aplicações são escalonadas considerando o dobro do seu tempo de processamento previsto, o que diminui a possibilidade deste tipo de falha. Por outro lado, uma previsão exagerada pode impactar no escalonamento das demais aplicações e no aumento da ociosidade do sistema.

Em estratégias de escalonamento, como estudado na Seção 3.4, aplicações que requisitam poucos processadores e pouco tempo de processamento são relativamente mais fáceis de serem escalonadas por *backfilling*. Um maior fator de folga de tempo de processamento permite escalonar uma aplicação mais longe do momento da sua chegada na fila de espera [Barquet et al., 2013]. Isso contribui para a redução do número de violações de tempo de resposta SLO, pois a área para escalonamento (processadores $\times$ tempo) sem violação é ampliada. Demais informações sobre SLA e SLO foram apresentadas na Seção 2.4.

Com base nas características de aplicações e estratégias de escalonamento apresentadas na Seção 3.4, deseja-se também que aplicações candidatas possam ser reescalonadas de nuvem para HPC com baixo custo de processamento da estratégia de escalonamento. Nesse contexto, tarefas com dependências podem ser muito custosas. Por outro lado, tarefas independentes exigem menos esforço de escalonamento.

Finalmente, deseja-se evitar a migração de aplicações com tarefas interativas, devido à diminuição da vazão de rede ocasionada pelo significativo número de comunicações que essas aplicações necessitam realizar e devido ao controle adicional que necessita ser considerado quando diversas comunicações são movidas. O principal motivo envolvendo rede de computadores é

que a diminuição da vazão da rede pode impactar na degradação do tempo de resposta de outras aplicações que estão no mesmo domínio de VLAN.

4.3 Fluxos de controle de aplicação

Para contextualizar área de convergência em uma Central de Dados, torna-se relevante modelar um sistema de fluxos de aplicação, onde evidenciam-se o papel dos escalonadores no tratamento dessas aplicações. Elaborou-se uma de Rede de Petri Lugar-Transição como representação desses fluxos, que permite uma melhor compreensão dos fluxos de controle possíveis, além de também permitir identificar eventuais bloqueios na arquitetura proposta. A Rede de Petri está representada na Figura 4.1. Os lugares representam filas, processadores e repositórios. As transições representam os fluxos de controle. Para mais informações a respeito de Redes de Petri sugere-se o trabalho de [Cardoso e Valette, 1997].

No paradigma proposto, o escalonador da Central de Dados pode tomar decisões que levam aplicações para as transições T1, T2 ou T6. A transição T3 representa a migração de aplicação de nuvem para HPC. Após a determinação de um desses destinos e o transporte da aplicação até ele, o tratamento dado à aplicação é determinado pela plataforma receptora.

Pela transição T1 trafegam aplicações de usuários de nuvem e de HPC que são escalonadas em L1, ou seja, fora das plataformas instanciadas. O sistema necessário à essas aplicações foi discutido na Seção 4.1. Aplicações de nuvem também podem ser encaminhadas para T6 para plataformas de nuvem. Aplicações HPC também podem ser enviadas diretamente para T13 para a fila de espera do escalonador L2 da plataforma HPC requisitada, sem passagem pela fila de espera L0 do escalonador da Central de Dados. Essa transição para aplicações HPC é a mesma adotada frequentemente em centrais de dados, como discutido na Seção 2.1.

Visando o uso de processadores L1 e L3 na área de convergência por aplicações puramente HPC, uma otimização no programa do usuário HPC pode habilitar a submissão dessas aplicações para o escalonador L0 da Central de Dados, que por sua vez, toma decisão sobre o uso das transições T1 ou T2, as quais levam à L1 ou L3, respectivamente. Aplicações puramente HPC não são encaminhadas para plataformas de nuvem, apenas são encaminhas para essas plataformas com escalonadores L5 as aplicações HPC encapsuladas para uso de serviço de nuvem.

Em T2 trafegam aplicações de nuvem e de HPC encaminhadas pelo escalonador L0 da Central de Dados para a fila de espera L2 de plataforma HPC. Isso significa que L0 também pode tomar decisões sobre a convergência nuvem-HPC, antes mesmo da aplicações ser escalonada em nuvem.

Em T3 trafegam aplicações de nuvem migradas a partir de escalonadores L5 (plataforma de nuvem) para escalonadores L2 (plataforma HPC). Nas abordagens estudadas na literatura, essas aplicações são resubmetidas para a fila de espera do mesmo escalonador L5 da plataforma de nuvem, diferentemente do que ocorre nesta proposta.

Como apresentado, T2, T3 e T6 representam transições para inserção de aplicações na fila de espera de outra plataforma. Enquanto isso, T1 deriva da fila principal, ou ponto de partida L0 da aplicação dentro da Central de Dados. A fim de evitar problemas de padronização, como resubmissões da mesma aplicação sem condições que possibilitem o seu término, uma aplicação jamais retorna para seu o ponto de partida L0.

Em plataformas de nuvem, a ressubmissão de aplicações para a fila de espera da mesma plataforma é um procedimento comum, derivado da estratégia de *overbooking* com preempção [Reiss et al., 2012]. Nesse contexto existe um controle adicional de escalonamento que fixa o número máximo de ressubmissões. Um outro exemplo que limita a aplicação consiste em determinar o tempo máximo dela dentro do sistema. Na Rede de Petri da Figura 4.1, L8 tem

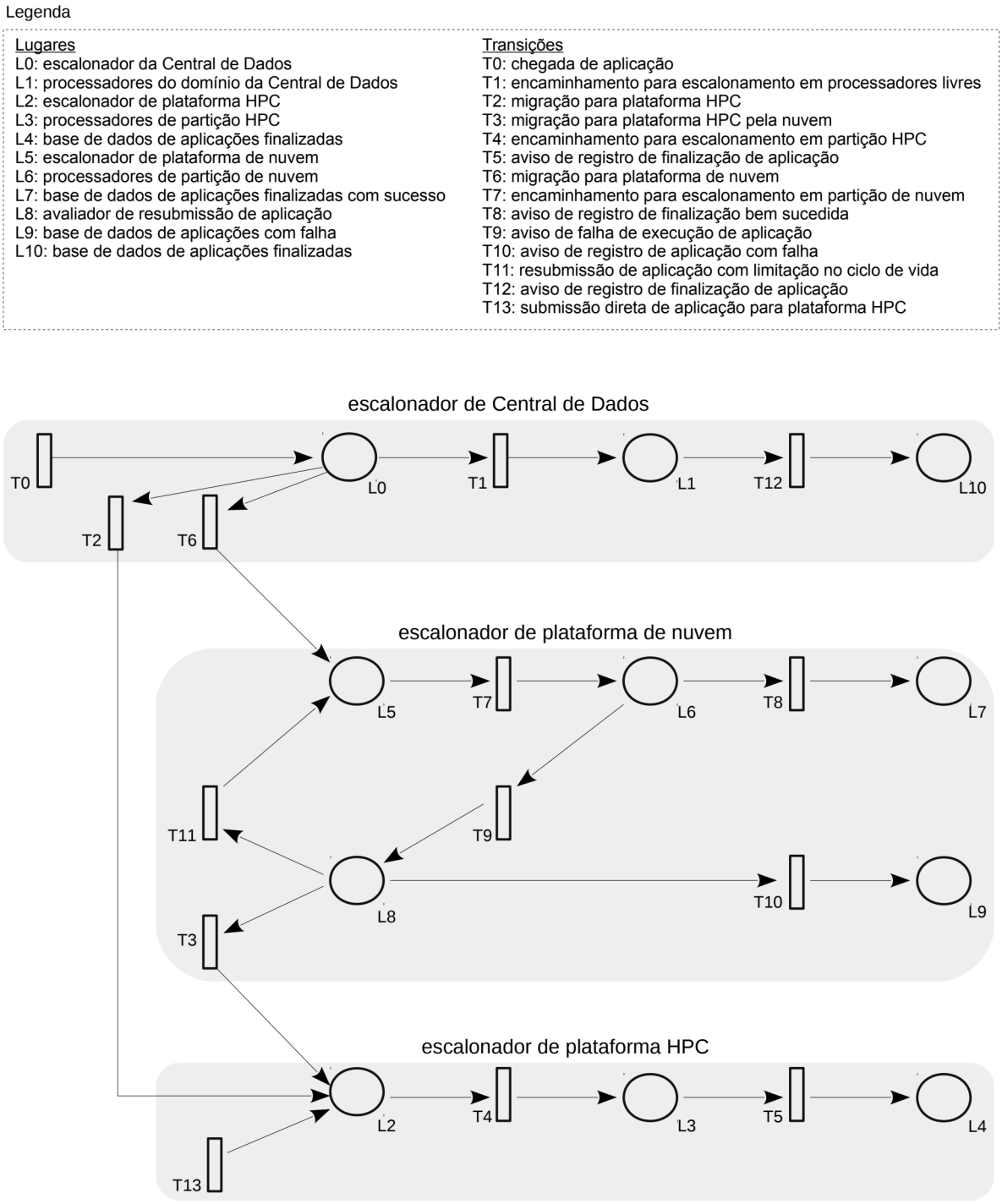


Figura 4.1: Rede de Petri Lugar-Transição para aplicações na Central de Dados.

o papel de avaliar a ressubmissão, podendo encerrar a aplicação com falha enviando-a para T10, ressubmetê-la por meio de migração para T3 ou ressubmetê-la para o escalonador L5 pela transição T11. Por outro lado, em plataformas HPC não é comum existir ressubmissão. Ou seja, em HPC, uma vez escalonada por L2, a aplicação será executada em L3, com falha ou sucesso até seu estado final em L4. Em caso de falha, o emissor da aplicação tem o papel de submeter uma nova aplicação.

Para resolver um potencial problema de enlace envolvendo migração entre plataformas diferentes, considera-se o fluxo de controle tal como apresentado na figura. O fluxo apresentado

não é reinicializável, não há bloqueios e os estados finais são L4, L7, L9 e L10. As invariantes de transição T7, T9 e T11 estão condicionadas à L8. A transição T11 não é utilizada quando L8 identifica que o ciclo da aplicação no sistema está encerrado, como discutido anteriormente, o que elimina a possibilidade de enlace. Por fim, consideram-se os fluxos apresentados como adequados para centrais de dados.

4.4 Notações dos principais elementos

Define-se nesta seção as notações fundamentais para os elementos da área de convergência. Os elementos são: aplicações, processadores e escalonadores. As notações apresentadas possuem como referência o estilo identificado na Seção de revisão 3.4.2. Algumas notações adicionais são apresentadas na Seção 4.6, que descreve funções de previsão de tempo de resposta.

Assume-se letra cursiva para representar conjunto, letra capital para representar elemento do conjunto e letra minúscula para representar propriedade do elemento. Em geral, letra grega é utilizada para representar fator de ajuste, como quando uma aplicação passa de um contexto para outro.

4.4.1 Definição de aplicações

Com base no estudo apresentado no Capítulo 2, identificaram-se quatro categorias de aplicações em centrais de dados: controle interno, plataforma, HPC e nuvem. A categoria controle interno consiste de aplicações de monitoramento de plataformas e de processamento de dados massivos de administração do provedor. As aplicações com dados massivos são similares às que deram origem ao termo HPC, antes mesmo do surgimento de centrais de dados. No Google, por exemplo, as aplicações internas são responsáveis pela indexação de documentos, atualização de banco de dados do sistema e manutenção de dados dos clientes. A categoria plataforma consiste de instâncias de plataformas de nuvem e de plataformas HPC, como apresentado na Seção 2.1. A categoria HPC consiste de aplicações que requisitam processamento em plataformas HPC. Por fim, a categoria nuvem consiste de aplicações que requisitam processamento em plataformas de nuvem.

Para abstrair a complexidade das análises sobre aplicações, sem perda de representatividade, tal como outros autores fizeram nos trabalhos apresentados na Seção 3.4, assume-se que as categorias controle interno e plataforma são também aplicações HPC. Com isso, tem-se apenas duas categorias de aplicação: nuvem e HPC.

Durante o procedimento de negociação de SLA, as centrais de dados consultam valores de tempo de resposta possíveis de serem cumpridos para determinadas aplicações. Existe um conjunto de SLOs de máximo tempo de resposta que é atualizado constantemente, tal como discutido na Seção 2.4. Considera-se $\mathcal{SLO} = \{SLO_1, SLO_2, ..., SLO_i, ..., SLO_z\}$ como representação desse conjunto. Assume-se que propriedades de aplicações HPC estejam compatibilizadas (ou *conciliadas*) com propriedades de aplicações do tipo nuvem. Isso é necessário porque alguns escalonadores HPC não consideram a propriedade máximo tempo de resposta SLO, sendo que esta propriedade é essencial para contratos de SLA em nuvem que devem ser mantidos em qualquer contexto de execução.

Um conjunto de aplicações é definido como $\mathcal{J} = \{J_1, J_2, ..., J_j, ..., J_n\}$. Uma aplicação J_j é definida com as propriedades wt_j, h_j e $SLO_{j,i}$, onde wt_j representa o tempo de processamento previsto, h_j o número de processadores requisitado e $SLO_{j,i}$ o i máximo tempo de resposta acordado para J_j, sendo $SLO_i \in \mathcal{SLO}$. Aplicações HPC são atribuídas com $SLO_{j,i}$ grande o

suficiente para evitar violações. Por outro lado, aplicações de nuvem são atribuídas com $SLO_{j,i}$ conforme seu SLO negociado em nuvem.

Assumindo-se a definição de uma aplicação do tipo nuvem como J^c, a conciliação de propriedades de aplicação de nuvem para o contexto de plataforma HPC é representada por $wt_j = wt^c * \alpha$, $h_j = h^c$, e $SLO_{j,i} = SLO^c_{j,i}$, onde α é o fator de ajuste de tempo de processamento.

Quando uma aplicação chega na fila de espera e é analisada por um escalonador, atribui-se novas propriedades com o objetivo de gerenciar o seu ciclo de vida na plataforma. A Figura 4.2 representa as propriedades de uma aplicação conciliada juntamente com outras propriedades adicionadas pelo escalonador.

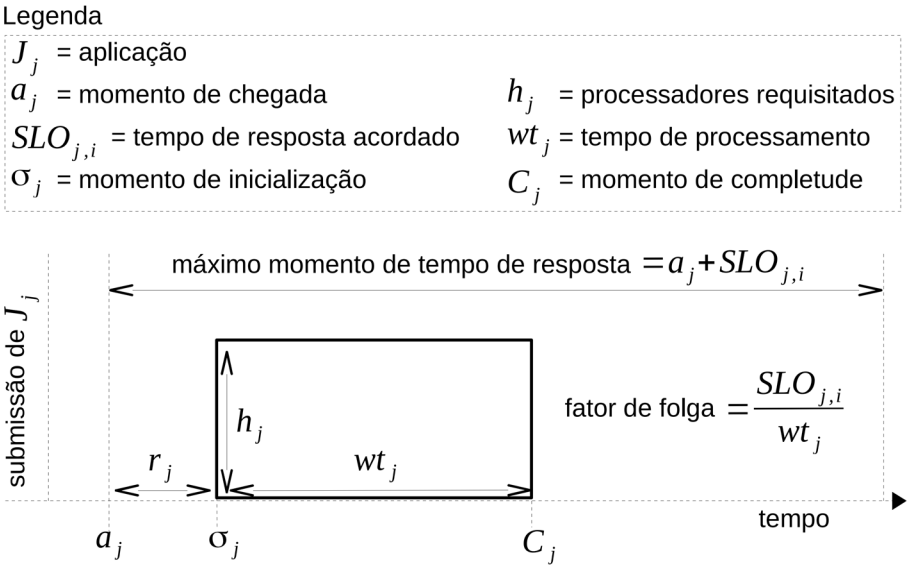


Figura 4.2: Propriedades de uma aplicação na área de convergência.

Na Figura 4.2, a propriedade de momento de chegada a_j é preenchida na chegada da aplicação na fila de espera da Central de Dados. A propriedade r_j representa o tempo gasto desde a_j até a inicialização da aplicação em σ_j. Assim, em r_j estão inclusas todas as latências, como comunicação em rede e tempo de aguardo de escalonamento. O instante do máximo tempo de resposta SLO é computado pelo escalonador por $a_j + SLO_{i,j}$ como ponto de referência para violação. Quando a aplicação é encerrada, o escalonador atribui o momento de completude C_j. Assim, uma aplicação violada possui $C_j >$ máximo tempo de resposta SLO.

O histórico do ciclo de vida de uma aplicação é representado por $J_j = \{wt_j, h_j, SLO_{i,j}, a_j, r_j, \sigma_j, C_j\}$. Observa-se que esta representação também é útil para análises de determinação de SLA de outras aplicações que ainda não chegaram na Central de Dados, pois pode representar a capacidade do escalonador em atender tais requisitos de aplicação.

4.4.2 Definição de processadores

Assume-se que os processadores da partição de um escalonador são homogêneos e representados como $\mathcal{P} = \{P_1, P_2, ..., P_q, ..., P_m\}$. Um processador P_q possui $\{s_q, Q_q\}$, onde s_q é a velocidade de processamento e Q_q uma fila de escalonamento. Determina-se a homogeneidade de processadores considerando $s_q = 1$. Considere m como o total de processadores do domínio e m_{conv} como o número de processadores envolvidos na área de convergência. Assume-se $m = m_{conv}$ quando todos os processadores estão na área de convergência.

4.4.3 Definição de escalonadores

Um conjunto de escalonadores é definido como $\mathcal{E}$. Um escalonador gerencia uma fila de chegada de aplicações no sistema (ou fila de espera) do tipo FCFS e determina o escalonamento de cada aplicação gerenciando a fila de cada processador. Assim, assume-se $\mathcal{L}$ como a fila FCFS, contendo um subconjunto de aplicações de $\mathcal{J}$. Assume-se Q_q como a fila de escalonamento de um processador P_q, considerando que existem $1 \leq q \leq m$ processadores.

4.5 Banco de dados global

Motivado pelo mecanismo *Estado Compartilhado* da Central de Dados do Google, no qual todo escalonador tem uma visão global dos demais escalonadores [Schwarzkopf et al., 2013], propõe-se um banco de dados global para suporte à estratégias de previsão de violação de tempo de resposta e tomada de decisão sobre migração de aplicações na área de convergência. A seguir, apresenta-se o banco de dados global por meio da Figura 4.3.

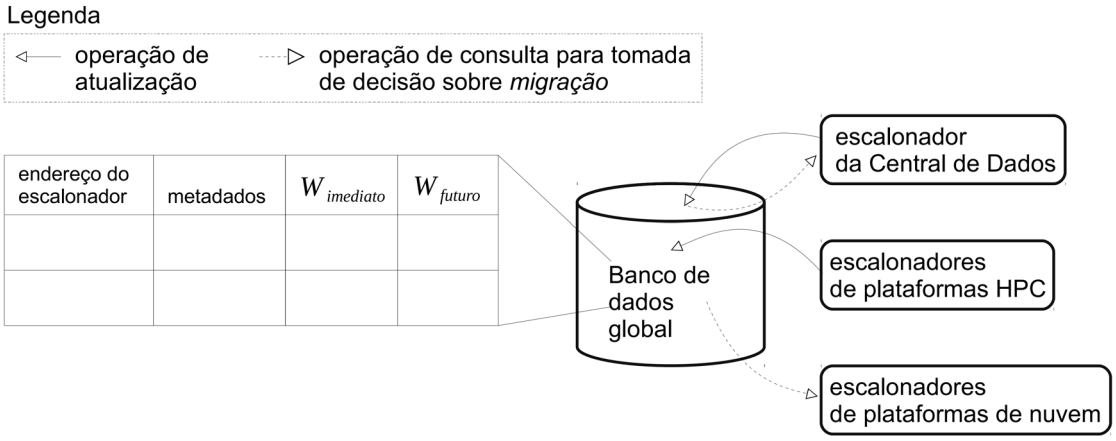


Figura 4.3: Proposta de banco de dados global da área de convergência.

A tupla de um escalonador $E_e \in \mathcal{E}$ no banco de dados global possui o endereço do escalonador na Central de Dados, seu metadados com detalhes sobre recursos para aplicações, um conjunto de *slots* $\mathcal{W}_{imediato}$ com representação do tempo ocioso corrente nos processadores da sua partição e um conjunto de *slots* $\mathcal{W}_{futuro}$ com representação dos espaços ociosos entre aplicações escalonadas e depois delas. Os conjuntos $\mathcal{W}_{imediato}$ e $\mathcal{W}_{futuro}$ formam uma área útil para alocação de aplicações como o exemplo de escalonamento ilustrado na Figura 4.4.

A obtenção de $\mathcal{W}_{imediato}$ é procedimento frequente de um escalonador. Esses *slots* são identificados no início de cada ciclo de escalonamento. Por outro lado, considerando estratégias gulosas, pode ser difícil obter $\mathcal{W}_{futuro}$. Todos os *slots* precisam ser percorridos até que $\mathcal{W}_{futuro}$ seja formado, o que descaracteriza o termo guloso, que consiste em encerrar a navegação assim que encontra um *slot* adequado à estratégia. Portanto, escalonamento com estratégia gulosa não forma o conjunto $\mathcal{W}_{futuro}$. Nesse caso impeditivo, desconsidera-se $\mathcal{W}_{futuro}$ em análise de tempo ocioso.

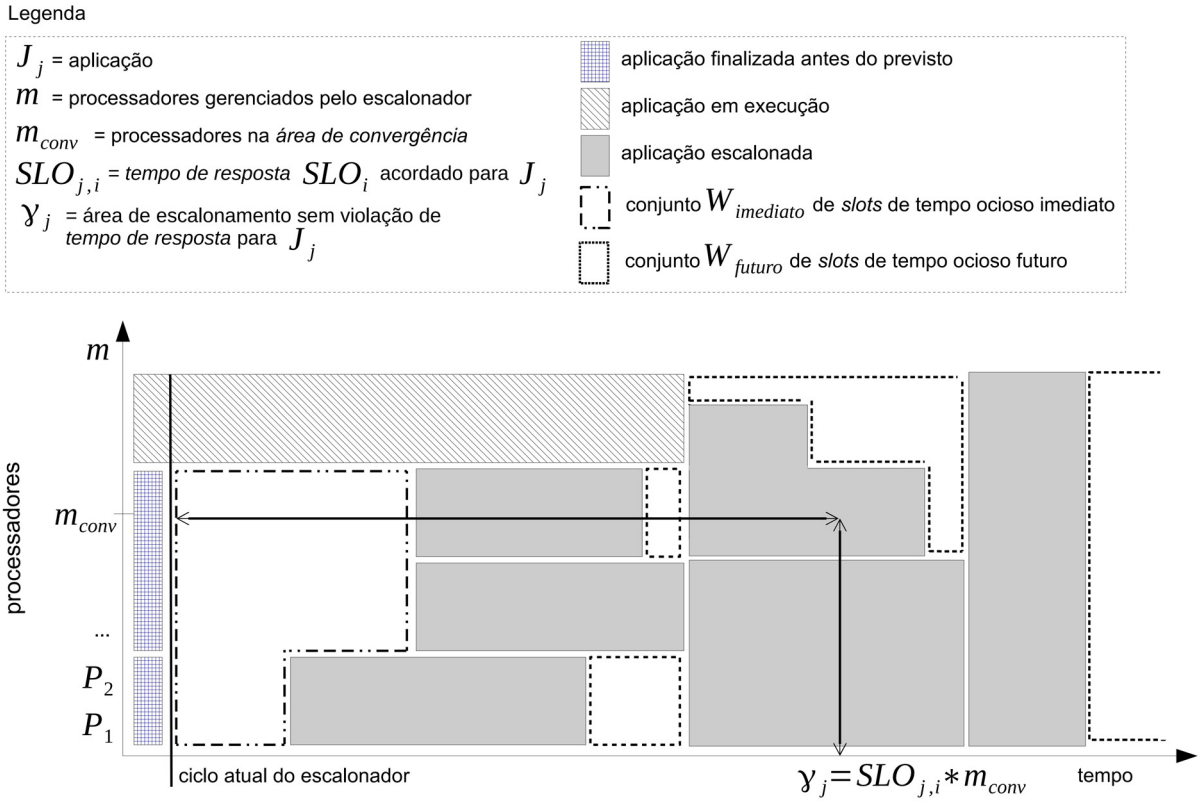


Figura 4.4: Exemplo de escalonamento com *slots* de tempo ocioso.

4.5.1 Discussão sobre escalabilidade

Torna-se relevante discutir características de escalabilidade do banco de dados global. Para tanto, apresentam-se a seguir considerações sobre número de tuplas, operações realizáveis e tamanho em bytes.

Em Central de Dados há um número relativamente pequeno de instâncias de plataformas e, consequentemente, de escalonadores. No banco de dados global, cada escalonador é representado por uma tupla, existindo assim um número pequeno de tuplas.

Quando uma aplicação candidata é identificada na fila de espera, o escalonador que gerencia essa fila faz uso do banco de dados global, realizando operação de consulta, a fim de tomar decisão sobre migração. A frequência dessa operação de consulta é uma por aplicação candidata. A cada ciclo de escalonamento é realizada também uma operação de atualização, que não depende do número de aplicações. O intervalo entre cada ciclo de escalonamento é definido por um temporizador ou tem como base um evento (quando uma aplicação chega ou encerra seu processamento). O escalonador da Central de Dados realiza o maior número operações de consulta, pois opera a fila de espera principal. No Google, por exemplo, a frequência média de chegada de aplicações é de 1 aplicação a cada 6,5 segundos [Sheng et al., 2012], considerando todo tipo de aplicação. Considerando apenas as aplicações candidatas, a frequência de chegada delas é mais espaçada no tempo. Em plataformas de nuvem ocorre uma consulta por aplicação candidata e apenas no processo de ressubmissão. Escalonadores de plataformas HPC não precisam conhecer dados sobre os demais escalonadores, pois não realizam análise de migração, descartando-se assim a realização de operação de consulta. Com isso, escalonadores HPC apenas atualizam seus *slots* de tempo ocioso. Não observa-se um número significativo dessas operações no banco de dados, considerando que o número de operações depende da frequência de chegada

de aplicações candidatas (1 aplicação a cada 6,5 segundos ou mais tempo) e do papel de cada escalonador na área de convergência.

O tamanho em bytes de cada *slot* de tempo ocioso é considerado pequeno, na ordem de Kbytes. Os metadados são formam uma lista simples de recursos que requisita Kbytes de espaço. Assim, cada tupla ocupa espaço na ordem de Kbytes, o que é considerado um tamanho pequeno.

Em termos de número de tuplas, operações e tamanho em bytes, considera-se que esse banco de dados possui escalabilidade. Em termos funcionais, o objetivo não é garantir que os *slots* representem o real estado do escalonamento de aplicações nos processadores. Pois muitas aplicações terminam mais cedo que o previsto, outras podem terminar mais tarde. Um escalonador pode obter dados dos demais escalonadores e no instante seguinte, antes do primeiro escalonador finalizar suas análises, esses dados não representarem mais o real estado do escalonamento. Tornar o banco de dados integro é uma tarefa complexa e os custos de implantação e processamento podem ser significativos, com muito impacto nas estratégias de escalonamento. Por este motivo, desconsidera-se aplicar estratégias que garantam integridade. Visa-se reduzir essa complexidade considerando erros na previsão, para o qual objetiva-se o banco de dados. As funções de previsão são apresentadas na próxima seção e a validação desta proposta considerando acurária e precisão da previsão são apresentadas no Capítulo 6.

4.6 Previsão de tempo de resposta

Na área de convergência, os escalonadores necessitam de funções de previsão de tempo de resposta para tomarem decisões sobre migração, escalonamento e contraoferta de SLO para aplicações. Determinou-se que as funções devem prever o tempo de resposta nos escalonadores, a violação e a oferta de SLO sem violação. Para determinação do menor tempo de resposta são necessárias previsões do momento de inicialização e do momento de completude da aplicação. Essas duas funções essenciais são descritas nas Seções 4.6.1 e 4.6.2, respectivamente. A função para determinação do menor tempo de resposta é descrita na Seção 4.6.3. Por fim, as funções para determinação de violação e oferta de SLO são descritas nas Seções 4.6.4 e 4.6.5, respectivamente. A seguir, apresentam-se as funções propostas. Para o formalismo delas considera-se a notação $J_{j,e}$ para representar a previsão da aplicação J_j no escalonador E_e.

4.6.1 Previsão do momento de inicialização

A previsão do momento de inicialização da aplicação é realizada pela função $F^{\sigma}(J_{j,e})$, a qual identifica o $\sigma_{j,e}$ nos *slots* de tempo ocioso de E_e. Segue abaixo a descrição desta função.

$$F^{\sigma}(J_{j,e}) = \begin{cases} J_{j,e} \text{ com } \sigma_j = t_0^{\mathcal{W}_{imediato,e}} + r_{j,e} : \text{se } F^{\mathcal{W}}(J_{j,e}) \subset \mathcal{W}_{imediato,e} \\ J_{j,e} \text{ com } \sigma_j = t_0^{\mathcal{W}_{futuro,e}} + r_{j,e} + \beta : \text{se } F^{\mathcal{W}}(J_{j,e}) \subset F^{\gamma}(\mathcal{W}_{futuro,e}, J_{j,e}) \\ J_{j,e} \text{ com } \sigma_j = \infty : \text{caso contrário} \end{cases}$$

Em $F^{\sigma}(J_{j,e})$, a chamada da função $F^{\mathcal{W}}(J_{j,e})$ tem o papel de identificar a área de alocação (processadores $\times$ tempo de processamento) desejada pela aplicação $J_{j,e}$; e a chamada da função $F^{\gamma}(\mathcal{W}_{futuro,e}, J_{j,e})$ tem o papel de identificar a área de tempo ociosa antes que a aplicação seja violada, tal como apresentado no exemplo de escalonamento da Figura 4.4. Em $\mathcal{W}_{futuro,e}$, β representa um tempo indeterminado a partir de $t_0^{futuro,e}$ onde o *backfilling* da aplicação pode ser realizado. Porém, nesse contexto, não há estimativa para $\sigma_{j,e}$ devido à imprevissibilidade do

aparecimento de *slots* de tempo ocioso. Assim, como forma de desqualificar $\mathcal{W}_{futuro,e}$, atribui-se $\beta = t_0^{\mathcal{W}_{futuro,e}} + SLO_{i,j,e}$, o que viola o tempo de resposta SLO da aplicação. Para representar o aceite de $\mathcal{W}_{futuro,e}$, atribui-se $0 \leq \beta \leq (SLO_{i,j,e} - wt_{j,e})$. Por fim, retorna-se ∞ em caso de não existir estimativa para $\sigma_{j,e}$ em $\mathcal{W}_{imediato,e}$ e $\mathcal{W}_{futuro,e}$.

4.6.2 Previsão do momento de completude

A previsão do momento de completude da aplicação é realizada por meio da função $F^C(J_{j,e})$, a qual identifica o $C_{j,e}$ nos *slots* de tempo ocioso de E_e. Uma previsão tem equações diferentes para escalonador com *overbooking* e sem *overbooking*. A diferenciação dessas equações está ilustrada na Figura 4.5. Como observado nessa figura, nas equações envolvendo *overbooking* utiliza-se o fator λ de risco de escalonamento. Um exemplo desse fator é descrito por [Birkenheuer, 2012].

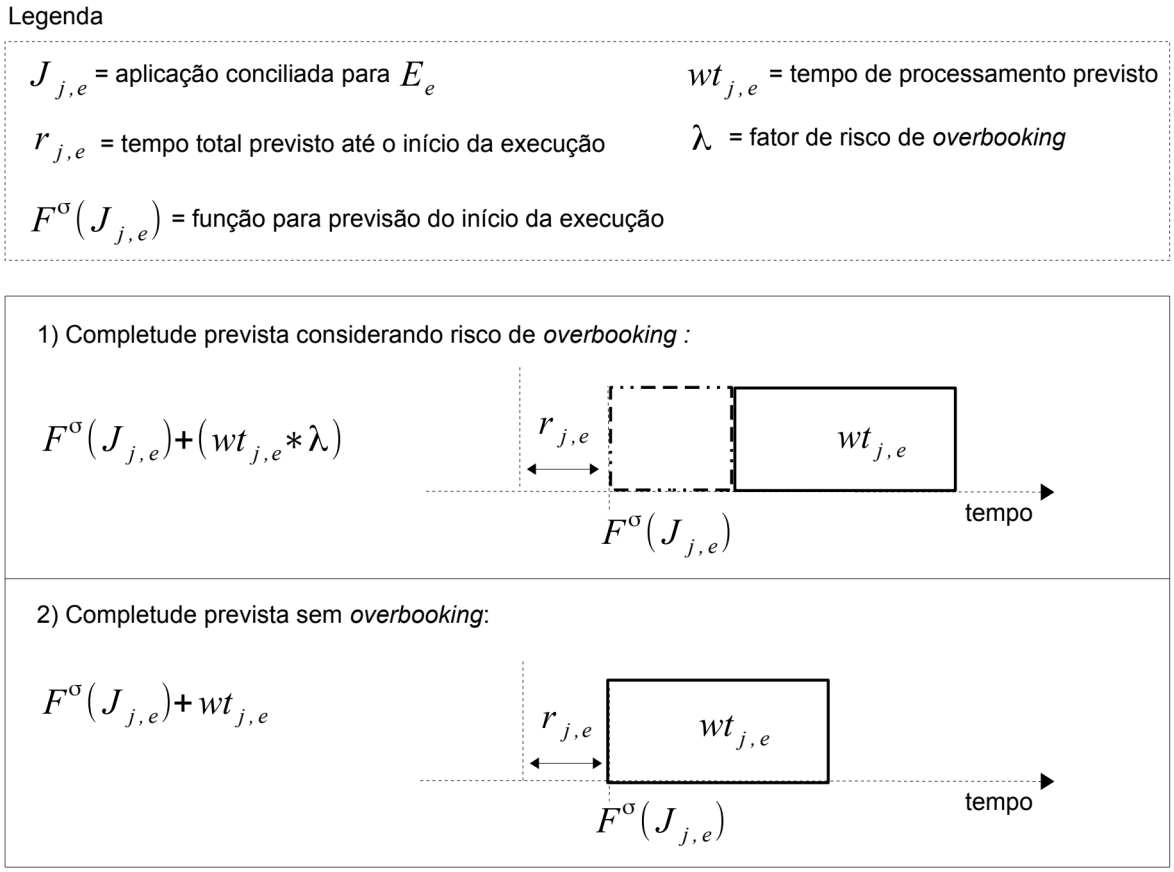


Figura 4.5: Equações de previsão para identificar o momento de completude da aplicação em plataformas com ou sem *overbooking*.

Na função $F^C(J_{j,e})$, como apresentado a seguir, a identificação de $C_{j,e}$ é ∞ quando $F^{\sigma}(J_{j,e})$ também é ∞, representando a imprevisibilidade $J_{j,e}$ em E_e. Nos demais casos, tem-se o $C_{j,e}$ conforme a classificação do tipo de escalonador E_e: com ou sem *overbooking*.

$$F^C(J_{j,e}) = \begin{cases} J_{j,e} \text{ com } C_{j,e} = \infty : \text{se } F^{\sigma}(J_{j,e}) = \infty \\ J_{j,e} \text{ com } C_{j,e} = F^{\sigma}(J_{j,e}) + (wt_{j,e} * \lambda) : \text{se } E_e \text{ é com } \textit{overbooking} \\ J_{j,e} \text{ com } C_{j,e} = F^{\sigma}(J_{j,e}) + wt_{j,e} : \text{se } E_e \text{ é sem } \textit{overbooking} \end{cases}$$

4.6.3 Previsão do menor tempo de resposta

A função $F^{J'}(J_j)$ para previsão do menor tempo de resposta de J_j considera um conjunto de escalonadores $\mathcal{E}_{seleto}$. O $\mathcal{E}_{seleto}$ representa escalonadores no banco de dados global que podem atender os requisitos técnicos da aplicação (tipo de *hardware* e *softwares*). Esses requisitos técnicos estão relacionados nos metadados do escalonador. Apresentam-se a seguir os passos desta função.

$$F^{J'}(J_j) = \begin{cases} J'_{j,e}: & \forall E_e \in \mathcal{E}_{seleto} \\ & \quad C \leftarrow F^C(F^\sigma(J_{j,e})) \\ & \text{ordenação de } C \text{ por } J_{j,e} \text{ com menor } C_{j,e} \end{cases}$$

Todos os escalonadores habilitados para executar a aplicação são selecionados considerando seu metadados na base de dados global, sem organização. Entre eles também está incluso o escalonador que está analisando a aplicação. Há uma exceção quando o escalonador que utiliza esta função não é o escalonador da Central de Dados. Neste caso, remove-se o escalonador da Central de Dados para evitar enlaces de aplicações no sistema, como apresentado na análise da Rede de Petri da Figura 4.1. Como resultado dessa seleção, tem-se $\mathcal{E}_{seleto} = \{E_1, ..., E_e, ..., E_f\}$, sendo $E_e = \{A_e, \mathcal{W}_{imediato,e}, \mathcal{W}_{futuro,e}\}$. O endereço do escalonador está representado por A_e, seus *slots* de tempo ocioso por $\mathcal{W}_{imediato,e}$ e $\mathcal{W}_{futuro,e}$.

A tupla de $J_{j,e}$ para previsão contém as propriedades $\{wt_{j,e}, h_{j,e}, SLO_{i,j,e}, a_{j,e}, r_{j,e}, \sigma_{j,e}, C_{j,e}\}$. As propriedades tempo de processamento previsto $wt_{j,e}$ e número de processadores requisitado $h_{j,e}$ são calculadas pelo processo de conciliação discutido na Seção 4.4. A propriedade tempo de resposta $SLO_{j,i,e}$ é mantida como acordada para o SLO em nuvem ou definida com um valor grande suficiente para não violar aplicações HPC. O momento de chegada $a_{j,e}$ é valorado pelo escalonador da Central de Dados quando a aplicação chega no sistema (ponto inicial *L0* da Rede de Petri). As propriedades $SLO_{j,i,e}$ e $a_{j,e}$ são mantidas como antes da conciliação porque elas são utilizadas como referência do SLA. A latência até a inicialização $r_{j,e}$ é calculada por uma função $F^r(E_e)$ de subssistema da Central de Dados, que considera o tamanho em bytes de $J_{j,e}$, dados sobre monitoramento de escalonadores e dados sobre os fluxos de rede T1, T2 e T3 representados na Rede de Petri. Atribui-se $r_{j,e} = 0$ quando o destino é o próprio escalonador que está analisando a aplicação. O momento de inicialização $\sigma_{j,e}$ e o momento de completude $C_{j,e}$ são valorados respectivamente pelas funções $F^\sigma(J_{j,e})$ e $F^C(J_{j,e})$. Assim, as previsões de momento de completude representadas por $C \leftarrow C_{j,e}$ ocorrem por meio de $F^C(J_{j,e}), \forall E_e \in \mathcal{E}_{seleto}$, considerando $1 \leq e \leq f$, sendo f o tamanho de $\mathcal{E}_{seleto}$.

A aplicação com menor tempo de resposta previsto é definida como $J'_{j,e}$. O processo de descoberta de $J'_{j,e}$ consiste em organizar C pelo menor momento de completude $C_{j,e}$ primeiro. O menor $C_{j,e}$ é o menor tempo de resposta, pois refere-se à finalização da aplicação na linha de tempo do escalonamento.

4.6.4 Previsão da violação de tempo de resposta

Uma violação de tempo de resposta ocorre quando o momento de completude C_j está acima do máximo tempo de resposta SLO_i acordado para J_j. Considerando o momento de chegada de aplicação a_j na Central de Dados e o acordo $SLO_{j,i}$, uma previsão de violação em $\mathcal{E}_{seleto}$ é obtida como descreve a função $F^V(J_j, J'_{j,e})$ a seguir. Nesta função, $C'_{j,e}$ representa o tempo de resposta da aplicação $J'_{j,e}$ em $\mathcal{E}_{seleto}$.

$$F^{V}(J_j, J'_{j,e}) = \left\{ C'_{j,e} - (a_j + SLO_{j,i}) \right.$$

Um resultado de $F^{V}(J_j, J'_{j,e})$ com valor negativo representa que a aplicação pode encerrar antes do acordo $SLO_{i,j}$. Ou ainda, esse resultado preve folga em unidades de tempo. Por outro lado, um resultado com valor positivo significa que o acordo $SLO_{i,j}$ pode não ser cumprido e ocasionar uma violação.

4.6.5 Oferta de tempo de resposta SLO

Em geral, como apresentado por [Costache et al., 2013] na revisão da Seção 3.8, uma Central de Dados gerencia um conjunto tal como $\mathcal{SLO}$ de aceite de valores de tempo de reposta. Um acordo SLO de tempo de resposta para J_j é representado por $SLO_{j,i}$, onde $SLO_i \in \mathcal{SLO}$. O SLO_i é um valor tabelado, em constante atualização pela Central de Dados, usualmente maior que o valor inicialmente requisitado por J_j.

Nesta proposta, assume-se o valor de $SLO_{j,i}$ como o tempo de resposta $C'_{j,e}$ previsto em $J'_{j,e}$, com a possibilidade de ser estendido por um fator de ajuste. Assume-se, a priori, $\tau = 1$ como o fator de ajuste. Segue a descrição da função.

$$F^{SLO}(J'_{j,e}) = \left\{ C'_{j,e} * \tau \right.$$

Não arredonda-se o resultado da função para encaixe em valores tabelados de $\mathcal{SLO}$ por entender que esse escopo é de um plano de negócios, fora do contexto desta proposta.

4.7 Trabalhos relacionados em previsão de tempo de resposta

Nesta seção são apresentadas estratégias, métricas, recursos computacionais e resultados considerados por trabalhos na literatura relativos à previsão de tempo de resposta. Esses trabalhos são importantes para compreender a relevância desta proposta. Algumas informações relevantes também foram apresentadas na discussão sobre RJMSs (Seção 2.3), onde foram identificadas as estratégias utilizadas por Centrais de Dados bem conhecidas, como seguem. No Google analisa-se o histórico de aplicações similares, a fim de prever o tempo de resposta das novas aplicações [Schwarzkopf et al., 2013]. No Yahoo e no eBay a identificação ocorre durante a execução, portanto, as aplicações são escalonadas sem a previsão [Vavilapalli et al., 2013]. No Twitter analisa-se um conjunto de variáveis da aplicação e do histórico de execuções por meio de aprendizado de máquina [Hindman et al., 2011], a fim de prever a tendência do tempo de resposta. No Facebook não foi possível determinar a estratégia de previsão, observou-se apenas que a elasticidade é aplicada para acelerar execuções [Facebook-team, 2012], reduzindo assim o tempo de resposta quando necessário. No Alibaba, o tempo de resposta é informado pelo programa que interage com o usuário, que cria e envia aplicações à Central de Dados [Zhang et al., 2014]. Por fim, no Grid'5000, o usuário interno de plataformas HPC é quem prevê o tempo de resposta [Margery et al., 2014] e o RJMS atribui o dobro desse tempo informado para garantir sua execução. Observa-se com base nesses trabalhos que o tempo de resposta pode ser determinado por vários agentes, tais como usuários, programas dos usuários e RJMSs. Embora algumas dessas estratégias de previsão tenham como fundamento o uso de modelos de aprendizado de máquina e histórico de execução de aplicações, as soluções são otimizadas para cada contexto e variam bastante. Apresentam-se a seguir as considerações e estratégias propostas por outros autores.

Uma estratégia de previsão baseada em modelo de aprendizado de máquina e análise de histórico de execução de aplicações foi proposta no trabalho [Leitner et al., 2010]. Nesse trabalho, a previsão pode ocorrer em vários momentos, que são: periódico; baseado em instância (quando um número de instâncias é recebido); sob demanda do usuário; devido algum erro (quando excede-se um erro na média da previsão); customizada (com base em condições aplicadas, por exemplo, quando um número conhecido de contratos do usuário são recebidos). No ambiente de execução da aplicação estão incluídos pontos de checagem, onde verifica-se o tempo de resposta por uma estratégia chamada *prevenção online*. Monitora-se e adapta-se o ambiente de execução conforme os dados vão sendo obtidos. A previsão não é aplicada para todo tipo de aplicação, sendo condicionada a requisitos do usuário e estados do sistema. Nessa estratégia aplica-se a métrica *média de erro da previsão*, que compara o resultado previsto com o observado. Considera-se o desvio padrão para determinar a variabilidade dos resultados. O melhor resultado dos experimentos mostrou que foram previstas 78% das violações. Em contrapartida, o pior resultado foi 60%. O dado utilizado para previsão é o tempo de resposta do serviço, com base no estado do ambiente. Os autores não apresentaram os recursos computacionais analisados para obtenção desse dado.

O método *Fuzzy* foi utilizado na estratégia de previsão proposta por [Wu e Guo, 2015]. Nesse trabalho foi realizado um levantamento sobre os recursos de usuário, aplicação e sistema que podem ser monitorados para determinação de previsões. Monitoram-se o caminho das aplicações do usuário no provedor, desempenho do banco de dados do provedor, eventos das tarefas de aplicação, *pool* de conexões criadas, fila de execução, rede de comunicação, memória principal, disco rígido, processadores, entre outros. Em seguida, os resultados do monitoramento foram classificados para compor as métricas: utilização dos recursos; desempenho do serviço; e disponibilidade. Analisaram-se também o histórico de execuções. Por fim, a previsão de desempenho foi determinada por um método *Fuzzy*.

Uma proposta de previsão em dois níveis foi apresentada por [Wang e Pazat, 2012]. No primeiro nível estima-se o tempo de resposta por meio de análise de histórico de execução. Na ausência de histórico, aplica-se uma prevenção *online*, onde avalia-se a degradação dos recursos utilizados pela aplicação, oportunizando adaptações do ambiente por parte do RJMS. No segundo nível determina-se o menor tempo de resposta por meio de diversas funções propostas pelos autores. Destacam-se o uso da acurácia e da precisão para a avaliação da proposta. A melhor acurácia obtida foi 0,97 e precisão 0,96. A pior acurácia obtida foi 0,89 e precisão 0,80. Os autores determinaram que a estratégia de adaptação do ambiente por prevenção *online* é significativa no melhoramento das métricas, podendo melhorar a acurária em mais 10% e a precisão em mais 15%. Porém, a estratégia de prevenção *online* não necessita ser sempre utilizada, já que a acurácia sem ela também apresenta resultados satisfatórios. A não aplicação dessa estratégia reduz as adaptações do ambiente durante a execução das aplicações, que pode ser mais relevante para alguns casos.

Uma proposta de previsão envolvendo uma combinação de estatísticas de interações *web* foi apresentada por [Jayathilaka et al., 2015]. As interações *web* envolvem comunicação entre programas de usuários externos e aplicações hospedadas em nuvem PaaS. Visou-se nesse trabalho prever o tempo de resposta dessas interações. Como resultado de diversos experimentos, os autores concluiram que 95% das previsões se realizam em menor tempo que o previsto.

Duas estratégias de previsão com base em aprendizado de máquina foram comparadas por [Hemmat e Hafid, 2016], que são: *Naive Bayes* e *Random Forest*. Utilizou-se dados de execução de aplicações do Google para essas análises. Foram utilizados os parâmetros de aplicação: memória requisitada, processadores requisitados, prioridade da aplicação, espaço ocupado em disco rígido e classe de escalonamento. Com base nos eventos de escalonamento, os

autores observaram o uso dos recursos por tarefas e o estado final delas. Em seguida, aplicaram-se as estratégias que identificaram as violações. Destacam-se nesse trabalho a determinação da acurácia e da precisão como forma de comparar as estratégias. Como conclusão, foi identificado que 2,2% das tarefas despejadas não foram reescalonadas, sendo caracterizadas como violadas. Observa-se que esse percentual refere-se a tarefas, não aplicações, as quais compreende-se ter um percentual menor que 2,2%. Embora este percentual ignore casos de falha do programa do usuário e não relaciona tarefas com aplicações, ele foi usado como referência para análises. Segundo esses autores, 0,2% de violações é um número frequentemente observado em sistemas de computação. No caso das estratégias avaliadas, *Random Forest* obteve o melhor desempenho com acurácia 0,99 e precisão 0,99. Por outro lado, *Naive Bayes* obteve acurácia 0,91 e precisão 0,91.

Um *fator de trabalho* para previsão de tempo de resposta foi proposto por [Mehrotra et al., 2015]. Calcula-se esse fator por meio do tempo de resposta sob um número de interações. Um segundo nível coleta esse fator e considera o tamanho da interação para prever o tempo de resposta. Mostrou-se por meio de uma estratégia de escalonamento que é possível manter o tempo de resposta previsto e reduzir o consumo de energia com desempenho satisfatório para aplicações científicas HPC hospedadas em centrais de dados.

Observou-se que trabalhos recentes focam em objetivos multidimensionais, procurando prever além do tempo de resposta, também o consumo de energia. Como exemplificação, no trabalho de [Jhawar e Piuri, 2015] apresenta-se que a redundância de instâncias de aplicações contribui para redução do tempo de resposta. Entretanto, essas réplicas consomem mais energia no sistema. Nesse contexto, [Jhawar e Piuri, 2015] apresentam a importância de aplicar outras estratégias que visam reduzir o consumo de energia, tal como a *consolidação de servidores*. Adicionalmente, os autores [Pore et al., 2015] apresentaram o DVFS (*Dynamic Voltage and Frequency Scaling* - Voltagem Dinâmica e Frequência Escalar), que é um recurso promissor no tema de estratégias multidimensionais. O DVFS habilita ajustes de frequência do processador e de energia consumida, fazendo com que RJMSs de centrais de dados possam gerenciar melhor seus gastos.

Na Tabela 4.1 são apresentadas as principais características das estratégias de previsão de tempo de resposta estudadas nesta seção. Observou-se que as métricas acurácia e precisão são utilizadas para determinar a eficácia de algumas dessas estratégias. Essas métricas são interessantes porque consideram erros negativos e positivos da previsão, onde os negativos são caracterizados por violações que ocorreram e que não foram previstas. Complementarmente, os erros positivos referem-se a previsões onde as violações não ocorreram. Observou-se também que a maioria das estratégias estudadas utilizam histórico de execução de aplicações e aplicam métodos complexos de previsão. Comparando-se essas informações com a estratégia de previsão proposta neste trabalho, conclui-se que a proposta é menos complexa e com potencial promissor de adaptação para diversos contextos. Não foi encontrado na literatura trabalho similar à estratégia proposta.

Tabela 4.1: Comparação de características de estratégias de previsão de tempo de resposta

Autor(es)	Contexto	Estratégia(s)	Recurso(s) considerado(s)	Métrica(s) de validação	Resultado(s)
[Schwarzkopf et al., 2013]	Google	análise de histórico	histórico de execuções	-	-
[Vavilapalli et al., 2013]	Yahoo e eBay	identificação durante execução	estado do ambiente	-	-
[Hindman et al., 2011]	Twitter	aprendizado de máquina com análise de histórico	histórico de execuções	-	-
[Facebook-team, 2012]	Facebook	identificação durante execução	estado do ambiente	-	-
[Zhang et al., 2014]	Alibaba	determinação por programa que interage com o usuário	-	-	-
[Margery et al., 2014]	Grid'5000	determinação por usuário, aplica-se o dobro do informado	determinação por usuário	-	-
[Leitner et al., 2010]	simulação	aprendizado de máquina com análise de histórico, identificação durante execução	estado do ambiente	média de erro	melhor resultado: 78% de erros previstos pior resultado: 60% de erros previstos
[Wu e Guo, 2015]	nuvem privada	*Fuzzi* com histórico de execuções e estado do ambiente	histórico e estado do ambiente considerando: caminho das aplicações do usuário no provedor, desempenho do banco de dados do provedor, eventos das tarefas de aplicação, *pool* de conexões criadas, fila de execução, rede de comunicação, memória principal, disco rígido, processadores, entre outros	-	-
[Wang e Pazat, 2012]	serviços *web* reais e simulação	dois níveis com análise de histórico, seguida de identificação durante execução	histórico de execuções e estado do ambiente	acurácia e precisão	melhor resultado: acurácia 0,97 e precisão 0,96 pior resultado: acurácia 0,89 e precisão 0,80
[Jayathilaka et al., 2015]	serviços *web* reais	combinação de estatísticas	interações *web* entre programa do usuário e serviço hospedado em nuvem	média aritmética	95% das previsões ocorrem em menor tempo que o previsto
[Hemmat e Hafid, 2016]	simulação	variações de *Naive Bayes* e *Random Forest*	memória requisitada, processadores requisitados, prioridade da aplicação, espaço ocupado em disco rígido e classe de escalonamento	acurácia e precisão	*Random Forest*: acurácia 0,99 e precisão 0,99 *Naive Bayes*: acurácia 0,91 e precisão 0,91
[Mehrotra et al., 2015]	simulação	dois niveis	fator de trabalho, tamanho das interações e funções probabilísticas	-	-

4.8 Trabalhos relacionados em migração de aplicação

A proposta de convergência nuvem-HPC é caracterizada pela delimitação de um número de processadores na Central de Dados, fluxos de controle de aplicação, funções de previsão de tempo de resposta, estratégias de migração e estratégia de escalonamento de aplicações. Trabalhos relativos a centrais de dados foram discutidos no Capítulo 2, estratégias de escalonamento e SLA no Capítulo 3, previsão de tempo de resposta na Seção 4.7. Nesta seção destacam-se trabalhos relativos à migração de aplicações. No próximo capítulo serão apresentadas as estratégias dos escalonadores e o escalonamento nuvem-HPC.

Uma migração pode ser realizada por muitas razões, tais como balanceamento de carga, redução de energia ou aumento da disponibilidade [Sharma e Chawla, 2013]. Nesta proposta, uma migração é realizada com o objetivo de enviar uma aplicação até uma plataforma onde é possível reduzir o seu tempo de resposta. Esse processo é precedido pela previsão de violação e seguido pelo escalonamento da aplicação. Objetiva-se com isso a redução de número de violações de tempo de resposta SLO na Central de Dados.

Há um significativo número de trabalhos na literatura envolvendo migração a partir aplicações de programas de usuários finais externos para a nuvem, aplicações de nuvem para nuvem e aplicações HPC para nuvem. Por outro lado, não foram encontrados na literatura trabalhos relativos à migração de aplicações de nuvem para HPC ou trabalhos envolvendo convergência no paradigma desta proposta. Assim, nesta seção são apresentados trabalhos relativos à migração nesses outros contextos.

A partir do ponto de vista de usuários externos, uma nuvem é vista como extensão para *hardware* virtualizado e *software*. Dessa forma, aplicações são migradas de contextos externos para dentro da nuvem a fim de expandir seus recursos disponíveis. Destaca-se que não é realizável a migração de aplicações de usuários externos para alguma plataforma HPC de Central de Dados no sentido de usá-la como um serviço. O que existe em algumas centrais de dados comerciais é a oferta de processamento HPC como um serviço dentro de plataformas de nuvem [Gupta et al., 2013]. Um *middleware* que captura especificações de aplicações de usuários e produz configurações otimizadas para seu uso em nuvem foi proposto por [Sefraoui et al., 2015].

Uma proposta de migração entre nuvem pública e privada foi apresentada por [Suen et al., 2011]. Propõe-se a duplicação de imagens de máquinas virtuais nessas nuvens a fim de reduzir o tempo de transferência em rede e tempo de inicialização quando uma migração é requisitada. Entretanto, esses autores comentam que muitas migrações mitigam qualquer relação custo-benefício.

Uma proposta de migração de aplicação mantendo sua execução (ou migração *ao vivo*) foi apresentada por [Amani e Zamanifar, 2014]. Nessa proposta utilizam-se dois parâmetros para tomada de decisão: tempo total de duração da migração e *downtime* (ou tempo de inatividade). A técnica apresentada reduz em média 51% do tempo total de migração. Isso ocorre devido à utilização do mecanismo *precópia*, o qual reserva recursos antecipadamente no computador de destino da aplicação e faz cópia de páginas da aplicação que são menos utilizadas. Com isso, ao decidir sobre migração, boa parte da aplicação já está apta para ser executada na nova localização. O mecanismo de precópia tem sido amplamante utilizado por diversos autores em propostas de migração ao vivo. A seguir são apresentados alguns desses trabalhos.

Uma comparação de benefícios entre fazer ou não fazer migração ao vivo em plataformas HPC foi realizada por [Atif e Strazdins, 2009]. Esses autores concluiram que a migração contribui para diminuição do tempo de processamento das aplicações. Comparando-se com o mecanismo de precópia no *hypervisor* Xen, esses autores também propõem uma técnica que melhora a transferência de páginas das aplicações, reduzindo assim o tempo de inatividade. Outros autores

também têm realizado esforços no sentido de melhorar o mecanismo precópia. Por exemplo, [Chanchio e Thaenkaew, 2014] conseguiram reduzir o tempo de inatividade habilitando mais *threads* para transferência de páginas da aplicação.

Um levantamento bibliográfico com detalhes sobre migração ao vivo foi elaborado por [Kapil et al., 2013]. Os autores [Sharma e Chawla, 2013] apresentam outro levantamento destacando técnicas de precópia e compressão de dados para transferência de aplicações. O autor [Strunk, 2012] faz um levantamento sobre os parâmetros considerados por diferentes estratégias de migração. Sobre este último trabalho, relatam-se os parâmetros: utilização de CPU, utilização da rede de comunicação, tamanho de memória e taxa de dados não modificados residindo em *cache* (ou *dirty page*). Recomenda-se a leitura das tabelas desses trabalhos a título de comparação entre diferentes características envolvendo migração de aplicações.

No contexto da Central de Dados do Google, quando uma aplicação de baixa prioridade é despejada para dar lugar a outra de maior importância, a primeira aplicação é encerrada prematuramente, perde seus dados e retorna para a fila de espera para ser reescalonada [Reiss et al., 2011] [Liu e Cho, 2012a]. Isso ocorre porque aplicações de baixa prioridade são em geral aplicações curtas, onde a relação custo-benefício de migração ao vivo não é satisfatória. Migrações ao vivo são priorizadas para aplicações longas e de maior prioridade, onde frequentemente existem clientes pagando por esses serviços.

Em comparação com plataformas de nuvem, plataformas HPC apresentam pouco nível de utilização de recursos ao longo do tempo [Sheng et al., 2012]. Embora não seja usual a migração de aplicações de HPC à nuvem, como forma de estender recursos de processamento locais, pois muitos desses ainda estão disponíveis em HPC, algumas aplicações científicas utilizam bibliotecas especializadas para processar tarefas na nuvem. Assim, a nuvem é utilizada como extensão de *software*.

Um caso específico de aplicações de nuvem em plataformas de nuvem com *overbooking* foi estudado por [Sheng et al., 2013]. Nesse estudo foi apresentado que aplicações de baixa prioridade podem ser encerradas prematuramente e em seguida podem ser ressubmetidas para a fila de espera da mesma plataforma. Nesse contexto, ocorre, portanto, migração para a mesma plataforma onde a aplicação foi interrompida.

Plataformas de nuvem e de HPC são instanciadas como aplicações de longa duração de tempo de processamento. Essas plataformas recebem aplicações de usuários e gerenciam arcabouços que podem executá-las. Do ponto de vista dessas plataformas, a migração de aplicação para outra plataforma é somente necessária se não existe mais recurso para escalonar a aplicação localmente. Prioriza-se a migração de aplicação para outra plataforma dentro da mesma Central de Dados. Quando o destino da aplicação é uma plataforma fora do domínio da Central de Dados, envolvem-se os chamados *Sistemas Federados*. Sistemas Federados não fazem parte do escopo deste trabalho. Para entendimento desses sistemas sugere-se a leitura do trabalho de [Hafshejani et al., 2013].

Com base nesses estudos, observou-se que as aplicações migradas não são plataformas instanciadas. Pois frequentemente essas plataformas são grandes em volume e interativas, o que impactaria na interrupção de muitas aplicações que estão se comunicando com o arcabouço. Por outro lado, a migração de aplicações candidatas caracterizadas na Seção 4.2 é um procedimento factível de ser realizado.

Conclui-se que a migração proposta na área de convergência é um procedimento simples em comparação com outros trabalhos envolvendo migração. Na proposta, considera-se aplicações candidatas e a migração considerando reinicialização da aplicação na plataforma receptora. Não aplica-se migração ao vivo na proposta porque as aplicações candidatas são curtas em tempo de processamento e de baixa prioridade, características que mitigam a relação custo-benefício.

4.9 Discussão

Este capítulo apresentou uma proposta de área de convergência nuvem-HPC. Definiu-se área de convergência, determinou-se quais características de aplicações são desejáveis à migração, definiu-se uma Rede de Petri que representa o fluxo de controle de aplicações na Central de Dados e, por fim, descreveram-se funções de previsão de tempo de resposta. As funções são utilizadas por escalonadores que serão apresentados no próximo capítulo.

Neste trabalho visam-se desenvolver estratégias que causem pouco impacto nos sistemas atuais de centrais de dados, aproveitando-se de mecanismos já existentes em escalonadores. Exemplos disto são a conciliação de propriedades de aplicação; o uso de *slots* de tempo ocioso para previsão de violação; e a migração considerando reinicialização de aplicação. A conciliação evita que as aplicações de nuvem sejam tratadas de forma diferenciada pelos escalonadores HPC, não exigindo destes modificação impactante. Os *slots* de tempo ocioso são obtidos frequentemente para o processo de escalonamento e, agora, no contexto de área de convergência, eles são aproveitados para a previsão de tempo de resposta. A migração considerando reinicialização da aplicação é um procedimento transparente para a plataforma receptora.

A maioria das estratégias de previsão estudadas na Seção 4.7 visa identificar o estado do sistema por meio de métodos matemáticos complexos, considerando vários elementos do sistema. Em geral, essas estratégias consideram o histórico de execução das aplicações. Quando essas estratégias são comparadas com a proposta neste trabalho, observa-se que a proposta contém contribuições significativas, como discute-se a seguir. Os *slots* de tempo ocioso utilizados na previsão podem ser vistos como o contexto histórico de recursos disponíveis mais recente na vida da Central de Dados. Não foi encontrado trabalho na literatura que utiliza esse recurso. Embora não sejam considerados dados da história de execução das aplicações, a proposta pode ser futuramente adaptada para isso, haja visto que todas as propriedades relativas à aplicação discutidas formam a história da aplicação. Ainda na proposta, assume-se uma variável r como forma simplificada de representação dos atrasos envolvidos na fila de espera, do transporte e da demora do escalonamento da aplicação. Assim como os *slots* de tempo ocioso, o valor de r também é obtido frequentemente por RJMSs existentes, que torna a proposta mais factível de ser implantada.

Após a previsão de tempo de resposta, pode ocorrer a migração da aplicação. A migração considerada nesta proposta consiste em inserir a aplicação na fila de espera de outra plataforma, seguida pela reinicialização da aplicação na mesma. A plataforma receptora não necessita realizar esforços adicionais, pois considera uma aplicação candidata migrada como qualquer outra aplicação na sua fila de espera.

Há uma variedade de soluções que podem tornar a área de convergência altamente priorizada para garantir o tempo de resposta de aplicações migradas. O que têm-se proposto neste trabalho são os mecanismos fundamentais à convergência nuvem-HPC e a redução do número de violações. Ideias complementares podem envolver atualização de sistemas externos e outras atualizações mais impactantes à Central de Dados. Apresentam-se de maneira simplificada algumas dessas ideias:

- Um conjunto de dados sobre histórico de execução de aplicações pode ser incluído no banco de dados global para melhorar a negociação de valores de SLOs.
- Um programa de usuário que submete aplicações puramente HPC pode implantar adaptações que permitem o uso de serviços de nuvem.
- O escalonador que analisa a migração de uma aplicação pode selecionar mais de um destino, replicar a aplicação e assim obter o resultado da execução mais antecipada,

que também pode contribuir para a redução do número de violações de SLOs. Nesse contexto, a eficiência energética também pode ser considerada.

A fim de completar o sistema proposto, no próximo capítulo são apresentadas as estratégias dos escalonadores envolvidos.

Capítulo 5

Escalonadores na Área de Convergência

Os escalonadores na área de convergência consistem em dar suporte à aplicação desde o seu aceite de SLO, passando pela migração se necessário, o escalonamento da aplicação, até o seu encerramento com verificação de violação de tempo de resposta. Os fluxos de controle de aplicação e as funções de previsão de tempo de resposta foram propostos e discutidos no Capítulo 4.

Neste capítulo são apresentadas as estratégias dos escalonadores, que podem utilizar as funções de previsão e tomar decisão sobre migração de aplicação. A estratégia do escalonador de Central de Dados é apresentada na Seção 5.1, do escalonador de plataforma de nuvem na Seção 5.2, do escalonador de plataforma HPC na Seção 5.3. Na seção do escalonador HPC é apresentada a estratégia de escalonamento da convergência. Com isso, tem-se uma representação completa de convergência nuvem-HPC em Central de Dados. Os trabalhos relacionados são apresentados na Seção 5.4 e as discussões na Seção 5.5.

5.1 Escalonador de Central de Dados

Para o escalonador de Central de Dados, considera-se que uma aplicação J_j possui um acordo $SLO_{j,i}$ de tempo de resposta ofertado/sugerido pelo programa do usuário, tal como ocorre na negociação de SLA apresentada na Seção 2.4. Com a aplicação desta estratégia, o escalonador pode sugerir um novo tempo de resposta SLO sem violação, a aplicação J_j pode ser escalonada nos processadores livres do domínio do escalonador ou a aplicação pode ser migrada para nuvem ou para HPC. O Algoritmo 1 a seguir representa esta estratégia.

Algoritmo 1 escalonador de Central de Dados

1: $J'_{j,e} \leftarrow F^{J'}(J_j)$ // previsão do menor tempo de resposta
2: $v \leftarrow F^{V}(J_j, J'_{j,e})$ // previsão da violação, resultado negativo significa término antes da violação
3: **if** $v < 0$ **then**
4: **if** e é local **then**
5: escalona-se J_j
6: **else**
7: envia-se J_j para e
8: **end if**
9: **else**
10: oferta-se SLO com base em $F^{SLO}(J'_{j,e})$
11: **end if**

O escalonador de Central de Dados tem o papel de prever se o acordo $SLO_{j,i}$ de uma aplicação J_j pode ser cumprido. Para tanto, utiliza-se como base os *slots* de tempo ocioso $\mathcal{W}_{imediato,e}$ e $\mathcal{W}_{futuro,e}$ de escalonadores $E_e \in \mathcal{E}_{seleto}$. A previsão de tempo de resposta com base nesses *slots* e a previsão de violação SLO são realizadas respectivamente pelas funções $F^{J'}(J_j)$ e $F^V(J_j, J'_{j,e})$ (linhas 1 e 2). A variável v é atribuída com valor negativo por $F^V(J_j, J'_{j,e})$ quando existe previsão com folga para o escalonamento. Caso o acordo $SLO_{j,i}$ possa ser cumprido (i.e. $v < 0$) (linha 3), J_j é enviado para o escalonador e determinado pelas funções de previsão, cujo e pode ser o próprio escalonador que analisa a aplicação (linha 4) ou outro escalonador na área de convergência (linha 7). Caso negativo, oferta-se um novo valor para $SLO_{j,i}$ obtido por meio da função $F^{SLO}(J'_{j,e})$ (linha 10). Ressalta-se que em situação de não necessidade de verificar o cumprimento do SLO, devido alguma decisão estratégica da Central de Dados por entender que o acordo $SLO_{j,i}$ já foi finalizado, remove-se a condição $v < 0$ e a oferta de novo SLO, passando diretamente para as etapas de migração e escalonamento.

Observa-se que há transparência no tratamento das aplicações HPC e de nuvem, por considerar-se o processo prévio de conciliação de propriedades realizado na função $F^{J'}(J_j)$. Também há transparência por parte deste escalonador sobre qual caminho de migração a aplicação deve seguir, já que todos os escalonadores em $\mathcal{E}_{seleto}$ são capazes de atender os requisitos técnicos e apenas o mais elegível em termos de tempo de resposta é considerado o escalonador de destino, ou e de $J'_{j,e}$. A estratégia de escalonamento de aplicação deste escalonador é a mesma aplicada em plataformas HPC. Descreve-se essa estratégia na Seção 5.3.

5.2 Escalonadores de plataformas de nuvem

Escalonadores de plataformas de nuvem na proposta de área de convergência tem o papel adicional de migrar aplicações para plataformas HPC por meio de análise de ressubmissão. Entende-se que essas aplicações já foram despejadas na preempção e, portanto, possuem seu acordo de máximo tempo de resposta mais próximo do limite. Frequentemente, plataformas HPC são lugares mais ociosos que plataformas de nuvem e podem escalonar e executar a aplicação candidata sem novo risco de preempção, tal como apresentado no Capítulo 4. Portanto, a previsão de tempo de resposta pode indicar para o escalonador da plataforma de nuvem se é melhor considerar o reescalonamento da aplicação na plataforma local ou migrá-la para outra plataforma. Essa estratégia de migração é apresentada no Algoritmo 2.

Algoritmo 2 escalonador de plataforma de nuvem

1: ajuste de J_j para limitação do seu ciclo no sistema
2: $J'_{j,e} \leftarrow F^{J'}(J_j)$ // previsão do menor tempo de resposta
3: envia-se J_j para e

Nesse algoritmo atribui-se um limite de ciclo para a aplicação a fim de evitar enlaces infinitos (linha 1), devido a novas preempções. A previsão de tempo de resposta de cada escalonador é obtida pela função $F^{J'}(J_j)$ (linha 2). O destino da aplicação é o escalonador e de $J'_{j,e}$ (linha 3), que pode ser o próprio escalonador.

5.3 Escalonadores de plataformas HPC

Os escalonadores de plataformas HPC têm somente o papel de escalonar aplicações. Assim, eles não fazem previsão de violação e também não tomam decisão sobre migração. A estratégia proposta é representada por um escalonamento em lote, que pode ser igualmente utilizada por escalonadores de centrais de dados. A fim de evitar controles específicos para cada tipo de aplicação e evitar distinção sobre o local do seu escalonamento, considera-se além da conciliação de propriedades de aplicação, também a definição que todos os processadores do domínio do escalonador estão inseridos na área de convergência. De outra maneira, haveria necessidade de delimitar essa área para controle distinto dos processadores, habilitando apenas alguns deles para aplicações migradas. Deseja-se não criar muitos controles na proposta, facilitando dessa maneira a sua incorporação em sistemas atuais de centrais de dados.

A estratégia de escalonamento está dividida em estágios. Uma aplicação pode ser escalonada antes de alcançar o último estágio e todas as aplicações são escalonadas em algum momento. Segue resumo desses estágios:

- **estágio preliminar** - coleta de aplicações na fila de espera;
- **1º estágio** - aplicação de *backfilling* devido a terminação de aplicações mais cedo que o previsto: são identificados *slots* de tempo ocioso no momento corrente do ciclo de escalonamento. Aplicações escalonadas nesses *slots* são inicializadas rapidamente;
- **2º estágio** - aplicação de *backfilling* entre aplicações escalonadas: são identificados *slots* de tempo ocioso entre aplicações escalonadas até que a nova aplicação possa ser escalonada ou não existam mais *slots* de tempo ocioso;
- **3º estágio** - escalonamento de aplicações além da área de *backfilling*: neste estágio todas as aplicações são escalonadas;
- **estágio de contribuição para previsões** - identificação dos *slots* de tempo ocioso $W_{imediato}$ e W_{futuro} e envio para o banco de dados global.

O fracionamento da estratégia de escalonamento permite aplicar mecanismos distintos em cada estágio. No 1º e no 2º estágios aplica-se *backfilling*. Motivações pelo uso de *backfilling* vêm dos trabalhos sobre EASY *backfilling* [Lifka, 1995] e *Conservative Backfilling* [Feitelson et al., 1997]. Em distinção ao EASY *backfilling*, o escalonamento proposto não considera aspectos de previsão de tempo de terminação para aplicações em execução. O *backfilling* do escalonamento proposto nesta seção assemelha-se ao *Conservative Backfilling*.

No *Conservative Backfilling* proposto por [Feitelson et al., 1997], considera-se uma estrutura de dados para armazenar o número de processadores disponíveis em cada tempo futuro. A principal diferença entre esse algoritmo e o proposto nesta seção está na consideração sobre processadores e *slots* de tempo ocioso. Nesta proposta obtêm-se dados de tempo ocioso de maneira gulosa analisando-se apenas o estado corrente de escalonamento das aplicações.

É comum na literatura abordar estratégias de escalonamento em uma seção denominada *definição do problema*, seguida pela descrição da estratégia e, por fim, pela identificação do seu fator de aproximação ou de competitividade. Entretanto, esta estratégia proposta é uma heurística com *backfilling*, o que não habilita a determinação de qualquer um desses fatores. Isto ocorre devido ao *backfilling* tornar imprevissível a identificação de melhor e pior caso possíveis. Apresentam-se a seguir a definição do problema e a descrição da estratégia de escalonamento.

5.3.1 Definição do problema

Considera-se um conjunto $\mathcal{J}$ com aplicações de nuvem e de HPC. Essas aplicações são paralelas, rígidas e independentes. Aplicações de nuvem do tipo não-plataforma não são interativas e o paralelismo delas é completo, ou seja, não há comunicação entre suas tarefas internas. O escalonamento na área de convergência pode alocar essas aplicações tal como segue:

- **A instância**: um conjunto de aplicações $\mathcal{J}$ com tuplas $J_j = (wt_j, h_j, SLO_{j,i})$, um conjunto de processadores homogêneos $\mathcal{P}$ de tamanho m, funções de verificação de adequação de estágio k como $V^k(J_j)$ e funções de escalonamento $P^k(J_j)$ para J_j em $\mathcal{P}$, considerando $m \geq 1$ e $m = m_{conv}$, onde m_{conv} é o tamanho da área de convergência.
- **O problema**: determinar um escalonamento factível de redução do número de violações de tempo de resposta SLO para aplicações de nuvem, escalonando-as juntamente com aplicações HPC. Considera-se o mecanismo *backfilling*. Todas as aplicações são escalonadas. Violações são aceitas. Desconsideram-se elasticidade, preempção de troca de contexto e migração de aplicações após o seu escalonamento.

5.3.2 Descrição da estratégia

Apresenta-se a seguir um resumo dos estágios de escalonamento e na sequência o detalhamento de cada um deles em descrição matemática. O estágio preliminar consiste em coletar um subconjunto $\mathcal{L} \subseteq \mathcal{J}$ da fila de espera FCFS. Considera-se $\mathcal{L}$ com tamanho u. Nos estágios seguintes, para cada aplicação J_j em $\mathcal{L}$, executam-se funções de verificação de adequação $V^k(J_j)$ e de escalonamento $P^k(J_j)$. Uma adequação é encontrada quando existe momento de inicialização σ_j factível de ser executado em um processador $P_q \in \mathcal{P}$, considerando o tempo de processamento wt_j. Esse processador é denominado *raiz*. Determina-se um escalonamento assim que um estágio k encontra em $\mathcal{P}$ o número de processadores h_j restantes com tempo ocioso disponível para wt_j. Quando acionado, o último estágio sempre escalona J_j. Por fim, os *slots* de tempo ocioso identificados durante os estágios são atualizados no banco de dados global como forma de contribuição para previsão de tempo de resposta de outras aplicações. Estes passos estão representados no Algoritmo 3.

Algoritmo 3 ações de um ciclo de escalonamento

for $J_j \in \mathcal{L}$ **do**
  determinam-se os *slots* de tempo ocioso do momento imediato
  if $V^1(J_j)$ determina σ_j e $P^1(J_j)$ encontra h_j com disponibilidade para wt_j **then**
    escalona-se J_j
  else
    determinam-se os *slots* de tempo ocioso de maneira gulosa
    if $V^2(J_j)$ determina σ_j e $P^2(J_j)$ encontra h_j com disponibilidade para wt_j **then**
      escalona-se de J_j
    else
      organizam-se as aplicações em Q pelo menor tempo de terminação de aplicação
      determinam-se os *slots* de tempo ocioso de maneira gulosa
      determina-se σ_j por $V^3(J_j)$
      determina-se h_j com disponibilidade para wt_j por $P^3(J_j)$
      escalona-se J_j
    end if
  end if
  determinam-se os *slots* de tempo ocioso $W_{imediato}$ e W_{futuro}
  enviam-se $W_{imediato}$ e W_{futuro} para o banco de dados global
end for

- **estágio preliminar - coleta de aplicações na fila de espera**:
  - Passo 1) coletar aplicações na fila de espera FCFS do escalonador: aplicações na FCFS são inseridas em uma nova fila para análises do ciclo atual de escalonamento. Defini-se a nova fila como $\mathcal{L}$. Este procedimento torna possível futuras implementações relativas a organização de aplicações em $\mathcal{L}$, por exemplo, devido a diferentes prioridades.
- **1° estágio** - aplicação de *backfilling* devido à terminação de aplicações mais cedo que o previsto:
  - Passo 1) identificar *slots* de tempo ocioso $\forall P \in \mathcal{P}$, considerando $1 \leq q \leq m$, sendo m o tamanho de $\mathcal{P}$, a fim de criar um mapeamento $\mathcal{S}$ para *backfilling*. Considere $\mathcal{S} = \{S_1, S_2, ..., S_z, ..., S_y\}$ denotar um conjunto de *slots* de tempo ocioso. Um *slot* S contém (v_z, q_z), onde v_z é o valor de tempo ocioso no processador q_z. Considere um tempo ocioso v_z ser determinado por $(\sigma_{q,x} - \zeta - 1)$, onde x é a próxima aplicação escalonada na fila Q_q de P_q. O processador P_q possui $\{s_q, Q_q\}$, onde s_q representa a velocidade de processamento. Determina-se $s_q = 1$ para todos os processadores, tornando-os homogêneos. Os *slots* de tempo ocioso descobertos são ordenados em $\mathcal{S}$ como $v_1 \geq v_z \geq v_y$. Considere, portanto, y de $\mathcal{S}$ como o número de processadores em estado de ociosidade no instante ζ. A área total desses *slots* é denotada por $\sum_{z=1}^{y} v_z$ e representa o tempo total disponível para escalonamento de J_j neste estágio.
  - Passo 2) verificar a adequação inicial de $\mathcal{S}$ para escalonamento de $J_j \subseteq \mathcal{L}$.

$$V^1(J_j) = \begin{cases} \sigma_j \leftarrow \zeta : & \text{se } h_j \leq y \text{ e } (h_j * wt_j) \leq \sum_{z=1}^{y} v_z \\ \sigma_j \leftarrow \emptyset : & \text{caso contrário} \end{cases}$$

    Embora $\sigma_j \leftarrow \zeta$ indique adequação, ainda é necessário verificar se wt_j é factível de ser realizado em todos os processadores em ociosidade. Esse é o papel da função $P^1(J_j)$, que verifica se $(wt_j \leq v_z)$, considerando $1 \leq z \leq y$ em S, até que o número de processadores h_j seja obtido ou $z = y$. Considera-se os processadores q necessários à J_j representados por um subconjunto H_j. Em caso de aceite dessas condições, um novo escalonamento é determinado por $J_j = (\sigma_j, wt_j, H_j, SLO_{j,i})$. A propriedade $\sigma_j = \zeta$ determina que todos os processadores em H_j devem iniciar a execução da aplicação imediatamente. Com base nas verificações de $V^1(J_j)$ e $P^1(J_j)$, todos esses processadores são factíves de executar o tempo de processamento wt_j.

- **2° estágio - *backfilling* entre aplicações escalonadas**:
  - Passo 1) verificar a adequação de $J_j \subseteq \mathcal{L}$ em $\mathcal{P}$: considere $V^2(J_j)$ identificar se existe algum processador P_q com *slot* de tempo ocioso adequado para J_j. Considere $\sigma_a - (\sigma_a + wt_a)$ denotar o tamanho do *slot* de tempo ocioso entre J_{a,Q_q} e J_{a+1,Q_q}, ou seja, entre aplicações escalonadas na fila Q_q. Objetiva-se determinar σ_j considerando $1 \leq a < f$ e $1 \leq q \leq m$.

$$V^2(J_j) = \begin{cases} \sigma_j \leftarrow (\sigma_a + wt_a + 1): & \text{se } \sigma_{a+1} - (\sigma_a + wt_a) > wt_j \\ \sigma_j \leftarrow \emptyset: & \text{caso contrário} \end{cases}$$

    Considere $(\sigma_a + wt_a + 1)$ ser representado também por σ_{raiz}. Considere $P^2(J_j)$ determinar o escalonamento para J_j com processadores entre $q, ..., m$, quando $V^2(J_j)$ encontra σ_{raiz}. Considere P_q ser o processador raiz incluído em H_j. Assim, por enquanto, o tamanho de H_j é 1. A fim de aumentar o tamanho de H_j até h_j requisitado, inclui-se P_g em H_j se $(\sigma_{raiz} > (\sigma_a + wt_a))$ e $((\sigma_{raiz} + wt_j) < \sigma_{a+1})$, considerando $q + 1 \leq g \leq m$ processadores e $1 \leq a < f$ aplicações escalonadas em Q_g. A verificação em $P^2(J_j)$ é satisfatória quando $H_j = h_j$. Define-se $\sigma_j = \sigma_{raiz}$ e $J_j = (\sigma_j, wt_j, H_j, SLO_{j,i})$ para determinar o escalonamento de J_j.

- **3° estágio - escalonamento de aplicações dentro de uma ampla área de recursos e tempo disponíveis após *backfilling***:
  - Passo 1) organizar o conjunto $\mathcal{P}$ pelo tempo de terminação mais cedo de aplicação em Q_q: define-se $\mathcal{D}$ como o conjunto das últimas aplicações escalonadas em Q_q, considerando $1 \leq q \leq m$. A organização é determinada pela mais curta terminação de aplicação primeiro considerando $\sigma_f + wt_f$, onde f é a última aplicação escalonada em Q_q.
  - Passo 2) verificar a adequação de $J_j \subseteq \mathcal{L}$ ser escalonada depois de $\mathcal{D}$: Considere $\mathcal{D} = \{D_1, D_2, ..., D_b, ..., D_d\}$ como representação de $\mathcal{P}$ organizado pelo tempo de terminação de processamento mais curto primeiro. Uma tupla em $\mathcal{D}$ contém (f_b, q_b, J_o), onde f_b é o índice da última aplicação escalonada dentro da fila identificada como q_b e J_o uma aplicação fictícia. A aplicação J_o representa uma operação que desabilita o processador do cenário de escalonamento entre os momentos σ_o e wt_o para, por exemplo, realização de manutenção programada. Aplicações não são enviadas para o escalonador que aplica esta estratégia quando J_o se aproxima na linha de tempo. A função $V^3(J_j)$ de verificação de adequação determina o tamanho do *slot* de tempo ocioso entre J_{f,q_b} e J_{o,q_b} por meio de $\sigma_o - (\sigma_f + wt_f)$, considerando $1 \leq b \leq d$.

$$V^3(J_j) = \begin{cases} \sigma_j \leftarrow (\sigma_f + wt_f + 1): & \text{se } \sigma_o - (\sigma_f + wt_f) \geq wt_j \\ \sigma_j \leftarrow (\sigma_o + wt_o + 1): & \text{caso contrário} \end{cases}$$

    Considere $(\sigma_f + wt_f + 1)$ ser representado por σ_{raiz}. Considere $P^3(J_j)$ determinar o escalonamento para J_j quando $V^3(J_j)$ encontra um σ_{raiz}. Considere P_b ser o processador raiz incluído em H_j. Assim, por enquanto, o tamanho de H_j é 1. A fim de aumentar o tamanho de H_j até h_j requisitado, incluí-se P_c em H_j se $(\sigma_{raiz} < (\sigma_f + wt_f))$ e $((\sigma_{raiz} + wt_j) < \sigma_o)$, considerando $b + 1 \leq c \leq d$ processadores. A verificação em $P^3(J_j)$ é interrompida quando o tamanho de

$H_j = h_j$. Na mesma direção que o estágio anterior, considere $\sigma_j = \sigma_{raiz}$ e o escalonamento como $J_j = (\sigma_j, wt_j, H_j, SLO_{j,i})$.

- **estágio de contribuição para previsões - identificação dos *slots* de tempo ocioso $W_{imediato}$ e W_{futuro} e envio para o banco de dados global**:
  - Passo 1) definir $W_{imediato}$: considere $(\sum_{z=1}^{y} v_z) - (h_j * wt_j)$ obtido no 1º estágio como representação de $W_{imediato}$ no banco de dados global.
  - Passo 2) definir W_{futuro}: caso J_j tenha sido escalonado no 1º estágio, considere W_{futuro} como o conjunto dos *slots* de tempo ocioso determinados nesse estágio até o escalonamento de J_j. Caso J_j tenha sido escalonado no 3º estágio, adiciona-se $\mathcal{D}$ na representação de W_{futuro}. A simplificação desta representação para um formato de área, tal como $W_{imediato}$, pode ser realizada pelo escalonador ou por procedimento no banco de dados global.

5.4 Trabalhos relacionados

Uma revisão de literatura relativa a RJMSs em centrais de dados foi apresentada na Seção 2.3, estratégias de escalonamento envolvendo problemas clássicos foram discutidas na Seção 3.4, escalonamento envolvendo SLA na Seção 3.8, estratégias de previsão de tempo de resposta na Seção 4.1 e estratégias de migração de aplicação na Seção 4.8. Não foram encontrados trabalhos na literatura relativos ao contexto de área de convergência e escalonamento envolvendo nuvem-HPC. Esta seção visa mostrar que, embora algumas estratégias de grids, nuvem e HPC estudadas utilizem *backfilling*, elas não consideram a migração de aplicação vinda de outro tipo de plataforma. Essas estratégias também não consideram a cooperação entre estratégias de *overbooking* e escalonamento em lote, como forma de melhor escalonar aplicações despejadas em nuvem.

Uma revisão sobre meta-escalonadores para HPC, grid, e nuvem foi apresentada por [Sotiriadis et al., 2012]. Na avaliação desses autores, a interoperabilidade pode ser realizada entre *Organizações Virtuais* para aplicações em execução. Entretanto, a *inter-colaboração* entre plataformas distintas é negligenciada, devido à complexidade de configuração e de adição de requisitos. Na contramão dessas afirmações, apresentou-se nesta proposta de convergência a migração apenas de aplicações candidatas dentro da mesma Central de Dados, que é factível de ser implantada devido à simplicidade das aplicações de nuvem consideradas e da migração seguida de reinicialização da aplicação.

Um escalonamento em Central de Dados para minimizar a média de tempo de resposta do sistema, ou seja, o *stretch* da inicialização de aplicação, foi desenvolvido por [Tran et al., 2015]. Esses autores destacam que a literatura sobre escalonadores para aglomerados de computação distribuída tem focado seus esforços distintamente em localidade e em balanceamento de aplicações.

Um melhoramento da taxa de aproximação para minimizar o *makespan* de aplicações rígidas e não-preemptivas em MCSP (*Multiple Cluster Scheduling Problem* - Problemas de Escalonamento em Multiplos Clusters) foi apresentado por [Bougeret et al., 2015]. Esses autores provêm uma taxa $\frac{7}{3}$-aproximação, onde a melhor taxa identificada na literatura é 2-aproximação, cuja solução foi proposta por [Ye et al., 2009]. As aplicações são submetidas em lotes e escalonam-se esses lotes de tal maneira que as aplicações do próximo lote possam iniciar o mais cedo possível. Em cada lote organizam-se as aplicações pelo maior tempo de processamento primeiro. Destaca-se que *backfilling* não é aplicado nessa abordagem.

Um escalonamento *online non-clairvoyant* e não preemptivo foi apresentado por [Tchernykh et al., 2010] como $GP_m|r_j, size_j|C_{max}$, onde GP_m representa um modelo de grid de tamanho m máquinas, r_j o tempo do lançamento da aplicação e $size_j$ o seu grau de paralelismo. O fator de competitividade para cenários de grids com processadores idênticos é de $2e + 1$. Esses autores utilizam o escalonamento em dois níveis, apresentam um cenário com alocação priorizando as máquinas com menor tempo de completude e outro cenário considerando o balanceamento de carga e o grau de paralelismo. O segundo cenário obtém menor *makespan*, como pode ser observado na Figura 5.1.

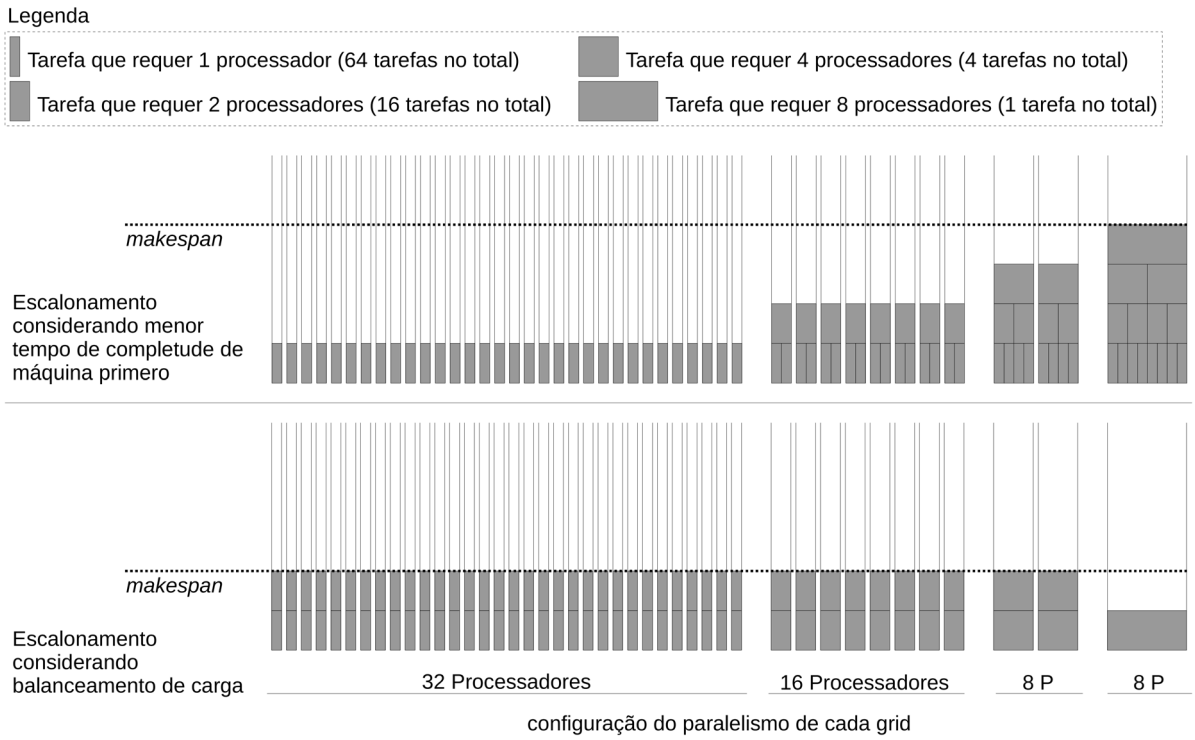


Figura 5.1: Escalonamento $GP_m|r_j, size_j|C_{max}$ [Tchernykh et al., 2010].

Um escalonamento *offline* com hierarquia em dois níveis para aplicações paralelas rígidas em grids foi apresentado por [Quezada-Pina et al., 2012], sendo caracterizado como $GP_m|r_j, size_j|C_{max}$, onde r_j é o tempo do lançamento da aplicação e $size_j$ o seu grau de paralelismo. Nesse trabalho são apresentados dois algoritmos de PS (*Parallel Scheduling* - Escalonamento Paralelo) com 3-aproximação e 5-competitividade. Esses algoritmos são nomeados como $MLB_a + PS$ (*Min-Lower-Bound* com Admissibilidade) e $MCT_a + PS$ (*Min-Completion-Time* com Admissibilidade) e escalonam a aplicação na máquina de menor capacidade primeiro. Os autores também apresentam adaptações onde obtêm-se 9-aproximação e 11-competitividade.

Os autores [Zhuk et al., 2004] comparam três heurísticas de escalonamento para minimizar o *makespan* em sistemas de grids descentralizados com máquinas paralelas homogêneas. A hierarquia de dois níveis é utilizada nesse contexto, sendo que no segundo nível são aplicados algoritmos de *strip-packing*. No *strip-packing*, as aplicações são definidas como retângulos com número de processadores p_j na largura e tempo de execução t_j na altura. O empacotamento consiste em alocar aplicações em escaninhos (ou *bins*) de largura limitada, mas com altura ilimitada. Os empacotamentos são realizados em uma das seguintes formas:

- BL (*Bottom-Left*): coloca-se os retângulos movendo-os para a esquerda. Não existe um empacotamento ótimo para BL.

- DBL (*Decreasing Bottom-Left*): ordena-se as aplicações por tamanho decrescente de número de processadores e com isso obtem-se taxa 3-aproximação.

As heurísticas abordadas por [Zhuk et al., 2004] são definidas como MCT+BL (*Minimum Completion Time* com *Bottom-Left*), MCT+SORT+BL (MCT+BL com organização descrecente de número de processadores) e MCT+DBL (MCT com *Decreasing Bottom-Left*). Como resultado desse estudo, os autores afirmam que BL e DBL podem melhorar a qualidade da taxa de aproximação.

Um escalonamento *online* não preemptivo e *non-clairvoyant* para aplicações paralelas em grids hierárquicos foi apresentado por [Ramirez-Alcaraz et al., 2011]. No primeiro momento na estratégia ocorre uma distribuição aleatória de aplicações para escalonadores independentes em outros locais, que por sua vez escalonam aplicações nos processadores com menor carga. Esse problema de escalonamento é caracterizado como $GP_m|r_j, size_j|\{t_w, SD_b, SWCT_w\}$, onde $t_w = \frac{1}{n} \times \sum_{j=1} n(c_j - p_j - r_j)$ é o tempo de espera da tarefa, SD_b é o *slowdown* limitado $\frac{1}{n} \times \sum_{j=1} n\frac{c_j - r_j}{max\{10, p_j\}}$, e $SWCT_w$ é o total do tempo de completude ponderado (ou *Weighted Completion Time*), obtido por meio de $\sum_{j=1} nc_j \times w_j$.

Um escalonamento *online*, *non-clairvoyant* e não preemptivo para aplicações rígidas independentes em grids foi apresentado por [Schwiegelshohn et al., 2008]. As aplicações são organizadas em modo decrescente pelo seu grau de paralelismo. Segundo esses autores, essa organização garante um bom local para a aplicação, independente da configuração do grid. O fator de *competitividade* alcançado é 5.

Um escalonamento *online clairvoyant* em grids com máquinas paralelas homogêneas para aplicações DAG (*Directed Acyclic Graph* - Grafo Acíclico Direcionado) foi apresentado por [Hirales-Carbajal et al., 2010]. Nessa estratégia as máquinas são organizadas em ordem crescente de seus tamanhos e o grau de paralelismo do Workflow é identificado com intuito de melhor escalonar as tarefas dependentes. Complementarmente, [Hirales-Carbajal et al., 2012] tem estendido seu trabalho aplicando EASY *backfilling*, considerado o caminho crítico para tempo de espera em fila e o *slowdown* para tomada de decisões de escalonamento.

Apresenta-se na Tabela 5.1 um resumo das estratégias de escalonamento estudadas nesta seção. Observou-se com base nos estudos apresentados que os objetivos das estratégias envolvendo grids, nuvem e HPC não focaram na convergência de aplicações. Um detalhe importante observado é que esses trabalhos focaram em minimizar o C_{max}, que é uma forma de minimizar o tempo de resposta do conjunto de aplicações. Isso alerta sobre o impacto da proposta de convergência nas plataformas HPC. Não é desejável o aumento do *makespan* HPC devido à convergência, pois isso descaracteriza o objetivo da plataforma e vai na contramão das estratégias HPC estudadas. Com isso, na validação apresentada no próximo capítulo, deseja-se verificar também qual é impacto causado no *makespan* da carga de trabalho HPC, devido à inserção de aplicações nuvem classificadas como candidatas.

Tabela 5.1: Características das estratégias de escalonamento em grids estudadas

Autores	Estratégia	Tipo de máquina	Tipo de aplicação	Objetivo	Aprox./Comp.
[Bougeret et al., 2015]	*offline*, não preemptivo	aglomerados idênticos	aplicações paralelas rígidas	minimizar o C_{max}	5
[Tran et al., 2015]	*offline*	centrais de dados com servidores heterogêneos	aplicações paralelas com execução de tarefas em processadores de mesma configuração	melhorar a média do tempo de resposta	-
[Zhuk et al., 2004]	*online*	grids descentralizados com máquinas paralelas homogêneas	aplicações paralelas independentes	minimizar o C_{max}	3
[Tchernykh et al., 2010]	*online*, *non-clairvoyant*, não preemptivo	grids com máquinas idênticas e paralelas	aplicações paralelas rígidas independentes	minimizar o C_{max}	$2e + 1$
[Quezada-Pina et al., 2012]	*offline*	grids com máquinas paralelas	aplicações paralelas rígidas independentes	minimizar o C_{max}	$MLB_a + PS$: 3
[Quezada-Pina et al., 2012]	*online*	grids com máquinas paralelas	aplicações paralelas rígidas independentes	minimizar o C_{max}	$MCT_a + PS$: 5
[Hirales-Carbajal et al., 2010]	*online*, *clairvoyant*	grids de máquinas paralelas homogêneas	DAG	distribuição de tarefas em grids com base no menor *slowdown*	-
[Ramirez-Alcaraz et al., 2011]	*online*, *non-clairvoyant*, não preemptivo	grids hierárquicos	aplicações paralelas	reduzir o *slowdown*	-
[Schwiegelshohn et al., 2008]	*online*, *non-clairvoyant*, não preemptivo	grids com máquinas homogêneas	aplicações paralelas rígidas independentes	minimizar o C_{max}	5

5.5 Discussão

Neste capítulo foram apresentadas propostas para escalonadores na área de convergência e uma estratégia de escalonamento nuvem-HPC com *backfilling* e gerenciamento de tempo de resposta. Os escalonadores apresentados utilizam as funções de previsão de tempo de resposta propostas na Seção 4.6 para tomarem decisão sobre migração. Com isso, completou-se a descrição do sistema proposto.

Identificou-se abertura para ideias complementares, pois o tema não é exaustivo. Exemplificando uma ideia complementar, observa-se que há possibilidade da aplicação migrada não ser escalonada no estágio previsto em escalonadores que priorizam aplicações na fila de espera. Com isso, a aplicação migrada, de baixa prioridade, pode aguardar mais tempo que o previsto, o que potencialmente impacta na violação SLO. Uma solução nessa fila de espera consiste em alterar a prioridade da aplicação migrada, sem alterar a prioridade nativa dela acordada em nuvem, fazendo com que a aplicação seja atendida mais rapidamente quando existem muitas aplicações na fila de espera. Esse processo de alteração de prioridade pode ser feito na conciliação de propriedades de aplicação.

Existe um significativo número de estratégias de escalonamento na literatura que consideram tempo de resposta, tais como os trabalhos estudados na Seção 5.4 sobre minimização do C_{max}. Entretanto, não foi encontrado trabalho sobre violação de tempo de resposta SLO em HPC. Tal como também não foi encontrado trabalho nesse sentido nas revisões dos capítulos anteriores. Também não foi encontrado trabalho envolvendo convergência nuvem-HPC. Com base nos trabalhos estudados, considera-se esta proposta como relevante para centrais de dados. No capítulo seguinte apresenta-se a validação da proposta.

Capítulo 6

Validação da Proposta

Com base nos trabalhos relacionados discutidos no decorrer desta tese, compreende-se que todas as estratégias propostas e discutidas nos Capítulos 4 e 5 são relevantes para o tratamento de aplicações em centrais de dados. Nesse contexto, a estratégia de escalonamento nuvem-HPC da Seção 5.3 é considerada a mais importante, pois por meio dela é que as aplicações são executadas e encerram seu processamento. Na literatura científica analisada no Capítulo 3, identificou-se que a validação de estratégias de escalonamento ocorre frequentemente por meio de análises de taxa de desempenho de aproximação ou de competitividade. Entretanto, identificou-se também que em heurísticas com *backfilling* não existe determinação dos casos pior e melhor, necessários à determinação dessas taxas. Por esta razão, valida-se a convergência nuvem-HPC por meio de simulação.

Neste capítulo são apresentadas a metodologia de validação na Seção 6.1; características do simulador na Seção 6.2; a carga de trabalho HPC e aplicações de nuvem candidatas respectivamente nas Seções 6.3 e 6.4; determinação das métricas na Seção 6.5; determinação do cenário de Central de Dados na Seção 6.6; determinação dos experimentos na Seção 6.7; resultados e discussão na Seção 6.8; e considerações finais na Seção 6.9.

6.1 Metodologia

A proposta é validada por meio de simulação *in silico* em um conjunto de arcabouços integrados, apresentados na seção seguinte. A simulação *in silico* é caracterizada por utilizar dados reais. Com isso, objetiva-se avaliar se a convergência nuvem-HPC é factível de redução de tempo de resposta em contexto de produção de centrais de dados. Esses experimentos são conduzidos e avaliados pelo seguinte método científico:

1. Determinação da carga de trabalho HPC;
2. Determinação das aplicações de nuvem candidatas que serão inseridas na carga de trabalho HPC;
3. Determinação das métricas;
4. Determinação do cenário de Central de Dados;
5. Determinação dos parâmetros para variação dos experimentos;
6. Execução dos experimentos;

7. Coleta e classificação dos resultados;
8. Avaliação dos resultados por meio das métricas;
9. Análise das características das aplicações e do ambiente que são mais impactantes em cada métrica;
10. Comparação dos resultados com outros resultados encontrados na literatura.

Nesse método, os experimentos não são executados em separado para distinção das métricas, pois as métricas estão fortemente relacionadas no contexto da convergência, onde avaliam-se a previsão de violação, a ocorrência ou não da violação e o impacto da execução do conjunto de aplicações de nuvem no HPC. Embora os mesmos experimentos sejam aproveitados à avaliação de todas as métricas, cada métrica é discutida em separado, visando destacar as características das aplicações e do ambiente que impactam no resultado delas. Por fim, comparam-se os resultados de cada métrica com outros encontrados na literatura.

6.2 Características do simulador

Para a simulação da estratégia de previsão, migração e escalonamento proposta nesta tese, foi construído um simulador denominado *CArS – Convergence Area Scheduler*. O CArS[1] foi construído na linguagem Python, usando serviços de outros dois arcabouços de simulação já existentes, o simulador de RJMS BatSim[2] (*Batch Scheduler Simulator*) [Dutot et al., 2016] e o simulador de grids Simgrid [3] (*Simulator of grids*) [Casanova et al., 2014]. O simulador CArS é uma ramificação do escalonador de aplicações em lote do RJMS OAR, descrito na Seção 2.3.

A integração desses arcabouços habilita a simulação de características de centrais de dados, como as apresentadas no Capítulo 2.1. Enquanto o Simgrid simula os recursos computacionais, tais como processador, memória principal, rede de comunicação e outros, o BatSim atua como um *middleware* do Simgrid e como um RJMS, habilitando integração de um novo escalonador. O CArS incorpora-se ao BatSim e recebe dados de eventos do ambiente. Apresenta-se na Figura 6.1 a forma de integração entre esses programas. Em seguida são discutidas as funcionalidades do CArS.

Os experimentos executados nessa integração de arcabouços são reprodutíveis. Nesse contexto, as aplicações são sempre recebidas no seu tempo programado e utilizam-se constantes de tempo que impactam no tempo de processamento delas. Isso significa que os resultados são repetidos em novas simulações que utilizam a mesma carga de trabalho e a mesma configuração de ambiente. Complementarmente, esse mecanismo faz que não sejam necessárias análises matemáticas relativas a confiança e a variabilidade dos resultados considerando uma série de experimentos.

A respeito do funcionamento do CArS, um ciclo de escalonamento inicia-se quando uma aplicação chega na fila FCFS ou quando uma aplicação termina. Esses são os únicos eventos no sistema de escalonamento. Como discutido na Seção 4.4.1, tem-se a conciliação de propriedades de aplicação como o primeiro passo para convergência nuvem-HPC. No ambiente simulador, a conciliação é realizada *offline*, durante a inserção de aplicações na carga de trabalho HPC. A carga de trabalho produzida é representada por um arquivo de formato JSON, contendo a configuração dos recursos requisitados, o máximo tempo de resposta SLO e o momento de

[1] O código-fonte deste simulador está disponível em https://gforge.inria.fr/projects/cars.
[2] Mais informações em https://gforge.inria.fr/projects/batsim.
[3] Mais informações em http://simgrid.gforge.inria.fr.

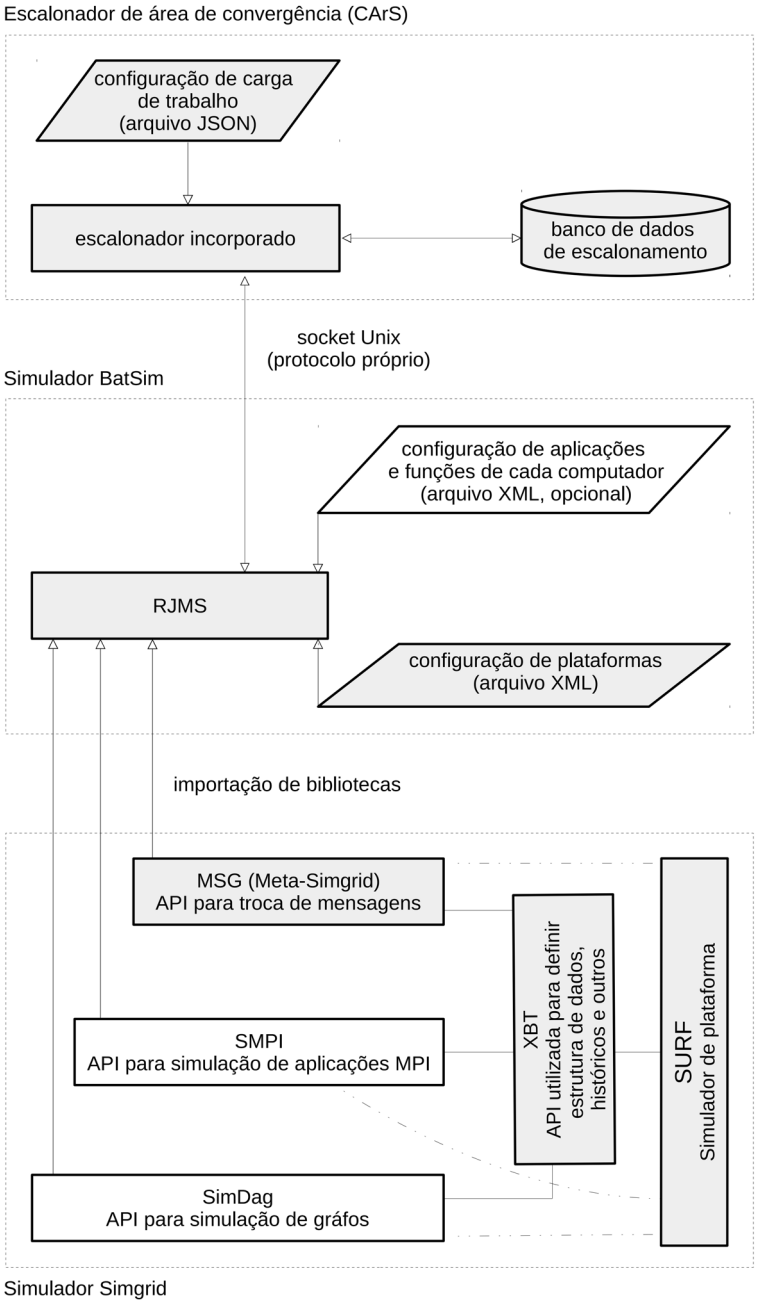


Figura 6.1: Integração dos programas utilizados para simulação de centrais de dados. Os recursos utilizados estão sombreados.

submissão de cada aplicação. Com a inicialização do programa simulador, lê-se o arquivo XML de configuração de plataformas e a carga de trabalho JSON. Em seguida, uma fila FCFS começa a receber as aplicações no seu tempo programado. Uma vez escalonada e iniciada a execução de uma aplicação, o término dela é antecipado ao seu wt_j (ou tempo de processamento), a fim de representar um cenário onde aplicações terminam mais cedo que o previsto. Essa técnica é usual em simulações de escalonamento realizadas para publicações científicas do grupo MOAIS. Ao término da aplicação, verifica-se a violação.

Todas as informações das aplicações escalonadas são armazenadas em um banco de dados e o núcleo do sistema CArS extrai um negativo das alocações, ou seja, espaços na linha de tempo onde não existe aplicação escalonada. Esses espaços são os *slots* de tempo

ocioso. A operacionalização do banco de dados é totalmente transparente e a determinação de escalonamento é realizada com base nos *slots*, em alto nível via linguagem Python. Não foi identificado problema de escalabilidade devido a operações no banco de dados. Com a inserção de aplicações de nuvem, é natural que os ciclos do escalonador ocorram mais vezes, pois há maior número de eventos de aplicação sendo tratados. Por meio dos trabalhos de [Dutot et al., 2016] e [Casanova et al., 2014], também não foi encontrado problema de escalabilidade devido ao aumento do número de ciclos.

A estratégia implantada no CArS considera as características apresentadas nos Capítulos 4 e 5. Há, porém, distanciamento entre alguns detalhes da descrição matemática e o que pôde ser implementado. Essas limitações são discutidas na seção seguinte. Demais detalhes técnicos do ambiente simulador, como instalação, produção de arquivos de carga de trabalho, execução de simulação e obtenção de resultados são apresentados no Apêndice B.

6.2.1 Limitações do simulador

Devido a divergências técnicas do simulador em relação a descrição matemática proposta, considera-se uma atualização na representação dos *slots* de tempo ocioso. Outras limitações são discutidas na sequência. Considera-se $\mathcal{S}$ como o conjunto organizado de todos os *slots* de tempo ocioso. Considera-se $S_z = (inicio_z, termino_z, H_z)$ como descrição de um desses *slots*, onde $H_z \subseteq \mathcal{P}$ representa um subconjunto de processadores ociosos no momento entre $inicio_z$ e $termino_z$ contidos em $\mathcal{P}$.

Ainda a respeito dos *slots* de tempo ocioso, adotou-se a seguinte estratégia para uso deles nos estágios do escalonamento: organiza-se os *slots* pelo menor momento de início primeiro e com isso obtêm-se *slots* de tempo ocioso do tempo corrente (adequado para o 1º estágio da proposta) e *slots* de tempo ocioso depois do tempo corrente até um momento futuro determinado por um número *mágico* grande suficiente para escalonamento de todas as aplicações. Esta segunda categoria de *slots* representa o 2º e o 3º estágio: (1) para *backfilling*; (2) para escalonamento após as últimas aplicações já escalonadas.

Em cada sessão de escalonamento, todas as aplicações escalonadas anteriormente são organizadas novamente, a fim de antecipar suas execuções. Esse comportamento não foi representado na estratégia de escalonamento proposta na Seção 5.3. No simulador isso é necessário porque ele é utilizado como representação do RJMS OAR, onde o tempo de processamento previsto é estimado com o dobro do requisitado. Nesse contexto, as aplicações terminam frequentemente mais cedo que o previsto, deixando longos *slots* de tempo ocioso. A fim de evitar esse desperdício de tempo deixado pelas aplicações que encerram antes do previsto, a estratégia de escalonamento do OAR reorganiza as aplicações trazendo-as para mais perto da linha do tempo corrente. Após esse processo de reorganização, escalonam-se as novas aplicações. Esta é uma característica de escalonamento com adoção de *Conservative Backfilling*, discutido na Seção 3.4.4. Na implementação do CArS, preservou-se esse mecanismo e ordenou-se os *slots* de tempo ocioso como nos estágios propostos. Compreende-se que essa estratégia é compatível e sem perda de representatividade com o que tem-se proposto no escalonamento da Seção 5.3.

Há também distinção entre a estratégia gulosa proposta e o que foi possível implementar no simulador. Na proposta, devido à separação em estágios, analisa-se o tempo corrente de cada processador (1º estágio) e em seguida analisa-se o escalonamento de aplicações, a fim de encontrar *slots* de tempo ocioso (demais estágios). No simulador, dispõe-se de um negativo das alocações, como apresentado anteriormente. Isto significa que todos os *slots* são descobertos pelo núcleo do simulador de maneira não gulosa. A maneira gulosa ocorre somente após a organização desses *slots*, no escalonamento de uma aplicação. Devido a essa característica do

simulador, implementou-se a mesma estratégia de escalonamento para ambos os estágios. Isso faz com que não sejam necessárias as etapas da descrição matemática onde verificam-se o estado de cada processador e as aplicações escalonadas, pois o negativo das alocações já representa esses dados. Preserva-se a estratégia principal que consiste em aplicar *backfilling* para escalonar uma aplicação no primeiro *slot* de tempo ocioso disponível, mantendo-se assim a representatividade da proposta.

O Algoritmo 4 a seguir representa o escalonamento de uma aplicação $J_j = (a_j, wt_j, h_j, SLO_{j,i})$ pelo escalonador, onde a_j representa o momento de chegada da aplicação na Central de Dados, wt_j é o tempo de processamento requisitado para cada processador, h_j o número de processadores requisitado e $SLO_{j,i}$ o i máximo tempo de resposta acordado, sendo $SLO_i \in \mathcal{SLO}$. Para a determinação de um escalonamento, adiciona-se a propriedade σ_j para indicar o momento de início da aplicação e substitui-se a propriedade h_j por H_j para representar o subconjunto de processadores P que estão aptos para iniciar a aplicação em σ_j. O início simultâneo em σ_j representa a rídigez da aplicação. As propriedades a_j e $SLO_{j,i}$ não são consideradas na determinação do escalonamento, pois todas as aplicações são escalonadas. No evento de término da aplicação computa-se $(a_j + SLO_{j,i}) \leq C_j$ para determinar se houve violação, onde C_j é o momento de completude da aplicação.

Algoritmo 4 escalonamento de uma aplicação

1: coleta-se $J_j = (wt_j, h_j, SLO_{j,i})$
2: obtem-se o conjunto $\mathcal{S}$ de *slots* de tempo ocioso
3: ordena-se $\mathcal{S}$ pelo menor momento de início primeiro
4: **for** $S_z \in \mathcal{S}$ **do**
5: **if** $termino_z - inicio_z + 1 \geq wt_j$ **then**
6: **if** tamanho de $H_z \geq h_j$ **then**
7: $H_j \leftarrow P \subseteq H_z$
8: $\sigma_j \leftarrow inicio_z$
9: encerra-se o enlace
10: **end if**
11: **end if**
12: **end for**
13: $J_j \leftarrow (\sigma_j, wt_j, H_j, SLO_{j,i})$

Várias estratégias estão envolvidas na proposta de convergência nuvem-HPC. Por outro lado, o CArS representa apenas uma instância de escalonador e não uma Central de Dados com o sistema completo. Ainda no campo das limitações, o BatSim habilita somente um escalonador integrado. Apresentam-se na Tabela 6.1 as limitações do CArS quando consideradas todas as estratégias propostas nos Capítulos 4 e 5.

6.3 Determinação das aplicações HPC

Coletou-se uma carga do trabalho HPC do repositório de cargas de trabalho paralelas mantido por Dror Feitelson [4]. Esse repositório contém cargas de trabalho de diversas plataformas de computação em larga escala, utilizadas em experimentação científica por diversos trabalhos na literatura, sendo considerado um repositório relevante para os objetivos desta proposta.

Utilizou-se uma carga de trabalho de produção HPC com 11K aplicações extraídas do PIK (*Potsdam Institute for Climate Impact Research*, na Alemanha) [Feitelson, 2016]. O

[4]Disponível em http://www.cs.huji.ac.il/labs/parallel/workload.

Tabela 6.1: Itens da proposta implementados no programa CArS

Itens da proposta	**Implementação no simulador**
Gerenciamento de tempo de resposta SLO	Implementado no programa.
Domínio da área de convergência	Implementado no programa. Considera-se uma área de convergência estática, com todos os processados inseridos no domínio.
Fluxos de aplicação	Não houve necessidade de implementação. Consideram-se habilitados os fluxos: aplicações de nuvem e de HPC para o escalonador da Central de Dados; aplicações de nuvem e de HPC para o escalonador HPC; e migração de aplicações de nuvem para HPC.
Banco de dados global	Não implementado. Entretanto, considera-se a tupla do escalonador corrente, contendo $W_{imediato}$ e W_{futuro}.
Função $F^{\sigma}(J_{j,e})$ para previsão de momento de inicialização de aplicação	Implementado no programa.
Função $F^{C}(J_{j,e})$ para previsão de momento de completude de aplicação	Implementado no programa.
Função $F^{J'}(J_j)$ para previsão de menor tempo de resposta	Não implementado porque depende de várias instâncias de escalonadores e do banco de dados global. Considera-se o escalonador atual como o melhor destino.
Função $F^{V}(J_j, J'_{j,e})$ para previsão de violação de tempo de resposta	Implementado no programa.
Função $F^{SLO}(J'_{j,e})$ para oferta de tempo de resposta SLO	Implementado no programa, mas não utilizado porque não existe simulação de programas de usuários que podem fazer uso desta previsão.
Estratégia de escalonador de Central de Dados	Implementada a estratégia de escalonamento. Os demais elementos, como decisão sobre migração e reoferta de SLO, não foram implantados porque considera-se apenas uma instância de escalonador e não existe aplicativo de usuário no cenário para negociação do SLO.
Estratégia de escalonadores de plataformas de nuvem	Não implementada. Considera-se que aplicações de nuvem foram migradas para a instância do escalonador.
Estratégia de escalonamento de escalonadores de Central de Dados e de HPC	Implementada no programa. Considera-se apenas uma instância com recepção e alocação de aplicações de nuvem e de HPC.

aglomerado onde essas aplicações foram executadas possui 2560 processadores e o nível de utilização médio de seus recursos é de 38%. Ressalta-se que essa carga de trabalha foi selecionada porque contém histórico de execução do mesmo mês da carga de trabalho do Google, maio de 2011. Realizou-se uma separação das aplicações referentes somente a esse mês, extraindo as 11K aplicações de um conjunto de 749K registrado de abril de 2009 até julho de 2012.

Essas aplicações HPC possuem diferentes tipos de requisitos de processador e tempo de processamento. Frequentemente as aplicações tem tempo de processamento de horas ou dias, e podem requisitar entre 1 e 1024 processadores, sendo que a maior parte delas requisita poucos processadores (até 16). A frequência de submissão das aplicações é esparsa. Na experimentação

científica preservou-se o momento de submissão de cada aplicação registrada no histórico de uso do aglomerado de processadores.

6.4 Determinação das aplicações de nuvem candidatas

Como discutido na Seção 4.2, aplicações candidatas têm potencial para serem migradas na área de convergência e nela obterem menor tempo de resposta. Ao mesmo tempo, deseja-se que essas aplicações contenham características que não impactem demasiadamente no escalonamento da carga de trabalho natural da plataforma receptora. Nesta seção, analisa-se a carga de trabalho da nuvem Google com o objetivo de identificar aplicações candidatas de um cenário de produção. Essa carga de trabalho foi disponibilizada pelo Google como documento público [Reiss et al., 2011] e refere-se ao mês de maio de 2011.

Tendo como base o estudo realizado na Seção de revisão de estratégias 3.6, onde analisou-se a nuvem Google, foi identificado que há um número significativo de aplicações que são despejadas pelo *overbooking*. Segundo [Reiss et al., 2012], 50% das tarefas (não aplicações) tem baixa prioridade e 30% de todo o conjunto de tarefas é encerrado anormalmente pelo menos uma vez. Nos trabalhos estudados não há como identificar o número de aplicações de baixa prioridade que são despejadas. Por outro lado, devido à menor prioridade atribuída, um significativo número dessas aplicações está sujeito à preempção seguida de migração e novo escalonamento, sendo, portanto, aplicações propensas à classificação como candidatas. Segue a caracterização dessas aplicações:

- **aplicações curtas**: requisitam tempo de processamento próximo de 3 minutos [Reiss et al., 2012];
- **aplicações que requisitam poucos recursos**: entre 1 e 4 processadores [Mishra et al., 2010];
- **aplicações com baixa prioridade**: número estimado de 270K aplicações na carga de trabalho analisada [Abdul-Rahman e Aida, 2014];
- **aplicações muito preemptadas**: a baixa prioridade delas faz com que sejam despejadas com frequência [Sheng et al., 2013].

As aplicações identificadas consumem mais recursos que o necessário, pois ficam mais tempo no sistema, sendo propensas à violação. Algumas aplicações classificadas como *gratis* no Google (vide Seção 3.4, como as utilizadas para envio/recepção de mensagens de e-mail e busca *web*, possuem SLO sem um limite de tempo rígido [Mishra et al., 2010]. Não foi encontrado na literatura o detalhamento a respeito da configuração de tempo de resposta SLO dessas aplicações.

Outra característica relevante para experimentação científica é a determinação da frequência de submissão de aplicações. Não foi encontrada na literatura informação específica sobre a frequência de aplicações de baixa prioridade. Por outro lado, há estudos que mostram dados de frequência considerando todas as prioridades de aplicação. Os autores [Sheng et al., 2012] analisaram a submissão de aplicações no Google por meio do índice de justiça (ou *fairness*). Identificou-se nesse estudo que a frequência de submissão é regular, conforme apresentado na Tabela 6.2.

Tabela 6.2: Síntese do trabalho de [Sheng et al., 2012] sobre frequência de submissão de aplicações no Google

Tipo de dado	Valor identificado em aplicações por hora
Número máximo de submissões	1421
Número médio de submissões	552 (uma submissão a cada 6,5 segs)
Número mínimo de submissões	36
Índice de justiça	0,94

6.5 Determinação das métricas

As métricas consideradas visam verificar dados relativos a violações de tempo de resposta e o *makespan* da carga de trabalho das aplicações. A métrica *percentual de violações* foi identificada com base nos trabalhos relativos a escalonamento com SLA na Seção 3.7. A eficácia da previsão de violação é determinada por meio da *acurácia* e da *precisão*, identificadas na Seção de trabalhos relativos a estratégias de previsão 4.7. O *makespan* foi identificado nos trabalhos relativos a escalonamento nas Seções 3.5 e 5.4. As discussões a respeito dos métodos de determinação dessas métricas aplicados a esta proposta são apresentados a seguir.

O percentual de violações é definido como $(\frac{violadas}{submetidas} \times 100)$, onde $violadas$ refere-se ao número de aplicações de nuvem com tempo de resposta violado e $submetidas$ ao número total de aplicações de nuvem submetidas à fila de entrada do escalonador. Não considera-se variabilidade desse percentual porque analisa-se cada caso de violação em separado, onde os resultados exatos são reprodutíveis, tal como discutido nas características do simulador na Seção 6.2.

Determina-se a acurácia da previsão por meio da média aritmética dos erros positivos e negativos ocorridos em todos os experimentos. Analisa-se, portanto, um cenário global, onde há variações entre os cenários. Cada resultado de experimento é calculado como: $(\frac{erros\_positivos+erros\_negativos}{total\_previsto})$, onde $erros\_positivos$ refere-se ao número de violações previstas não ocorridas, $erros\_negativos$ ao número de violações não previstas ocorridas e $total\_previsto$ o número total de violações previstas. Quando a média dos erros é menor que 1, tem-se uma acurácia considerada satisfatória. Em contrapartida, resultado maior que 1 indica que muitos erros não foram previstos. A variabilidade do erro da acurácia é determinada pelo desvio padrão. Nesta métrica considera-se a variabilidade porque estão envolvidas diversas configurações de experimento, que irão compor apenas um resultado: a acurácia do conjunto de experimentos. Esse resultado é reprodutível se todos os experimentos diversos forem executados novamente.

Embora seja um método amplamente utilizado para análises de erro de previsão [Hemmat e Hafid, 2016], somente a acurácia não identifica a eficácia da previsão, pois ela considera todos os erros, positivos e negativos. Nesse contexto é importante destacar a previsão somente sob os erros ocorridos. Para tal, considera-se a obtenção da precisão da previsão. O cálculo da precisão é determinado por $(\frac{previstas\_ocorridas}{total\_ocorrido})$, onde $previstas\_ocorridas$ refere-se ao número de violações previstas ocorridas e $total\_ocorrido$ ao número total de violações ocorridas. A precisão é melhor quando o resultado é mais próximo de 1. Entre outras palavras, isso significa que a maioria dos erros foram previstos com sucesso. A variabilidade, tal como na acurácia, é determinada pelo desvio padrão.

Por fim, o *makespan* da carga de trabalho, ou C_{max} do conjunto de aplicações, é determinado pelo momento de completude da última aplicação no sistema. A variabilidade do *makespan* não é necessária porque analisa-se cada caso em separado.

6.6 Determinação do cenário de Central de Dados

A representação de uma Central de Dados completa com integração de diversos escalonadores é um processo demasiadamente complexo, seja em simulação *in vitro* (simulação matemática), *in silico* (simulação usando dados reais) ou em *vivo* (simulação em ambiente real). No campo de limitações do simulador, o BatSim permite apenas uma instância de escalonador integrado ao seu RJMS, tal como discutido na Seção 6.2. Com o intuito de resolver essa complexidade e ainda continuar avaliando de maneira satisfatória, consideram-se apenas as principais características de área de convergência da proposta, representadas pela instância de um único escalonador, o CArS, onde ocorre o escalonamento final de aplicações de nuvem em HPC. O CArS contempla a mesma estratégia de escalonamento do escalonador da Central de Dados e do escalonador de plataforma HPC (vide Rede de Petri da Central de Dados apresentada na Seção 4.3). Existem três casos discutidos na proposta envolvendo a convergência que estão relacionados a esses escalonadores, que são considerados na determinação do cenário. Esses casos são apresentados como seguem:

1. Aplicações chegando no escalonador da Central de Dados, para serem escalonadas no seu domínio de processadores;

2. Aplicações migradas do escalonador da Central de Dados para o escalonador HPC, para serem escalonadas na sua partição;

3. Aplicações migradas da nuvem para o escalonador HPC, para serem escalonadas na sua partição.

Uma instância dos arcabouços integrados habilita, portanto, a simulação das principais características da proposta. Na seção seguinte apresentam-se os parâmetros de variação dos experimentos executados nesta representação de Central de Dados.

6.7 Determinação dos experimentos

Os experimentos consistem na variação das configurações de aplicação de nuvem inseridas no HPC e do tamanho da área de convergência. Inseriu-se as aplicações de nuvem com frequência regular na carga de trabalho de produção HPC com frequência esparsa. A seguir apresenta-se a Tabela 6.3 com os parâmetros de variação dos experimentos.

Discutem-se a seguir os motivos para escolha dos parâmetros apresentados nessa tabela. Manteve-se a configuração das aplicações HPC como no histórico de produção do PIK, tal como apresentado na Seção 6.3. O menor número de processadores (10) da área de convergência foi determinado para aumentar o número de violações, pois testes preliminares mostraram que não há violação considerando o tamanho total do aglomerado de processadores do PIK (2560). Quando ocorre a variação do tamanho da área de convergência, modifica-se também o número de processadores das aplicações HPC que requisitam mais processadores que o disponibilizado. As demais propriedades de aplicação são mantidas como na carga de trabalho original. Assim, exemplificando, uma área de convergência com 10 processadores executa aplicações com no

Tabela 6.3: Parâmetros de variação dos experimentos, que determinam a existência de 432 experimentos distintos

Parâmetro	**Valor**
número de aplicações HPC	11K
características das aplicações HPC	diversas, como no histórico de produção
número de aplicações de nuvem	267K
número de processadores requisitados por aplicações de nuvem	1, 2, 4
tempo de processamento requisitado por aplicações de nuvem	180 segundos
fator de folga de tempo de processamento requisitado por aplicações de nuvem	1,25; 1,5; 2; 3; 4 até 15
número de processadores da área de convergência	10, 20, 40, 80, 160, 320, 640, 1280, 2560

máximo esse mesmo tamanho. Isso ocorre porque definiu-se que a área de convergência ocupa toda partição HPC. Ao diminuir o tamanho dessa área para avaliação de diferentes situações de violação, é necessária também a redução do número de processadores requisitados por aplicações. Não desconsideram-se aplicações HPC.

As características das aplicações de nuvem consideradas nos experimentos foram determinadas com base nas discussões relativas a aplicações candidatas (vide Seção 6.4). A respeito do número dessas aplicações, o número 267K foi determinado considerando frequência regular de submissão de 1 aplicação a cada 10 segundos. Essa frequência é diferente da identificada considerando todas as aplicações do Google, que é de 1 aplicação a cada 6,5 segundos. Entende-se que a intenção da simulação não é representar a migração de toda a carga de trabalho da nuvem. Por meio da frequência considerada, chega-se ao número de 267K aplicações, que é um número próximo do observado por [Abdul-Rahman e Aida, 2014] ao determinar 270K aplicações *gratis* no Google, que são as aplicações candidatas à migração. Com isso, considera-se a frequência adotada como sendo satisfatória.

A determinação do fator de folga 1,25 deve-se a representação de aplicações muito urgentes, com maior potencial de violação. Por outro lado, o maior fator de folga (valor 15) foi determinado por meio de testes preliminares, onde objetivou-se identificar zero número de violações de tempo de resposta. No contexto explorado, as violações começam a ocorrer ao descrescimo do fator de folga 15, o que oportuniza várias discussões por meio das métricas adotadas. Na próxima seção são discutidos os resultados dos experimentos.

6.8 Resultados dos experimentos e discussão

Nesta seção são apresentados os resultados dos experimentos e as discussões considerando as métricas. Na Seção 6.8.1 discutem-se as violações de tempo de resposta. Na Seção 6.8.2 discute-se a eficácia da estratégia de previsão de violação. Por fim, na Seção 6.8.3 discute-se o impacto na carga de trabalho HPC devido à inserção das aplicações de nuvem.

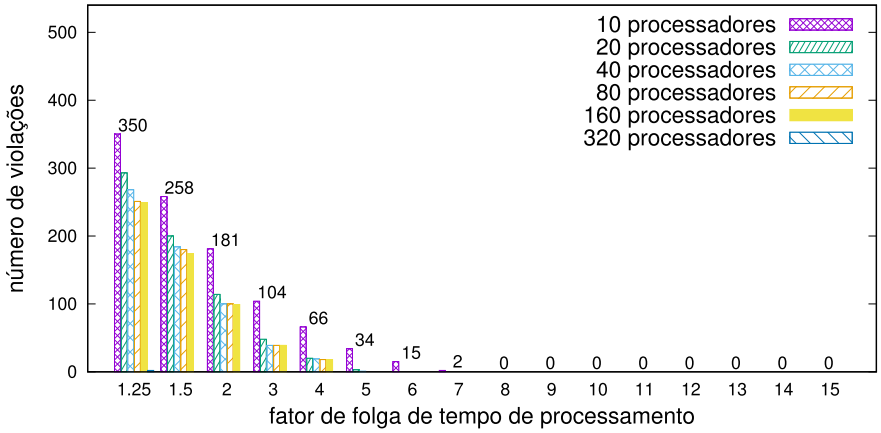
6.8.1 Violações de tempo de resposta

Os resultados sobre violações de tempo de resposta são apresentados na Figura 6.2. Nessa figura classifica-se o número de violações por requisitos de processador e fator de folga de tempo de processamento das aplicações. Analisando-se a demanda de processadores, as aplicações que requisitam maior número são mais violadas devido à maior necessidade de tempo ocioso para *backfilling*. Não é sempre que existe *slot* de tempo ocioso no plano do escalonamento antes do máximo tempo de resposta negociado para uma aplicação de nuvem. Isso faz com que essas aplicações sejam escalonadas mais adiante na linha de tempo, que provoca maior número de violações. Uma abstração desse caso de violação é apresentada na Figura 6.3.

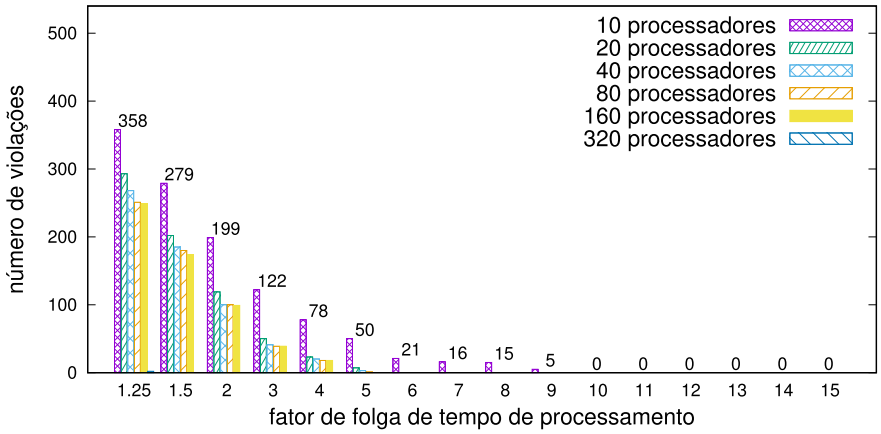
Considerando-se os resultados sobre fatores de folga {1,25; 1,5; 2} na área de convergência (AC) de tamanho 10 e 4 processadores requisitados por aplicações nuvem, os percentuais de violação obtidos foram respectivamente 0,0018%, 0,0014% e 0,0009%. Esses foram os maiores percentuais relativos a violação observados nos experimentos. Esses valores são menores que o relatado por [Hemmat e Hafid, 2016] ao considerar 0,2% de violações em centrais de dados. Destaca-se que as aplicações de nuvem candidatas consideradas nesses experimentos possuem curto tempo de processamento e requisitam poucos processadores, enquanto o percentual de violações relatado por [Hemmat e Hafid, 2016] considera todo tipo de aplicação. Segundo o trabalho [Abdul-Rahman e Aida, 2014], as aplicações *gratis* no Google são 41% do total de aplicações. Considerando que essas são as aplicações candidatas dos experimentos, uma fração considerável de violações foi reduzida dentro do percentual médio 0,2% de violações identificado por [Hemmat e Hafid, 2016]. Consideram-se os percentuais obtidos nos experimentos deste trabalho como sendo satisfatórios, pois são muito pequenos. Há, além disso, discussão relativa ao fator de folga, como segue.

Com base na Figura 6.2, obteve-se zero número de violações quando o fator de folga de tempo de processamento foi definido como 15. Esse valor permite que as aplicações possam ser escalonadas mais longe do momento de chegada delas na fila de espera. Assim, existem mais *slots* de tempo ocioso disponíveis para *backfilling* e a ocorrência de violações é reduzida. O mesmo comportamento de evolução de violações seguiu nos experimentos com diferentes tamanhos de AC e diferentes requisitos de processadores por aplicações, que mostra que o principal atributo para evitar violações de tempo de resposta é o fator de folga. O trabalho de [Barquet et al., 2013] também apresentou uma conclusão neste sentido.

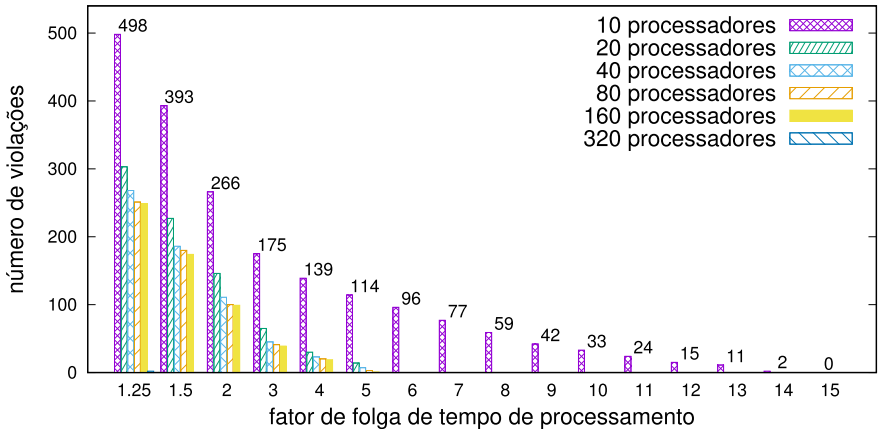
Com base no trabalho de [Reiss et al., 2012] foi possível estimar um valor mínimo de fator de folga das aplicações de baixa prioridade no Google. Segundo esse trabalho, aplicações de baixa prioridade aguardam em média até 241,9 segundos para serem escalonadas. Considerando que a maioria dessas aplicações requisita até 180 segundos de tempo de processamento, concluí-se que o menor fator de folga delas no Google é $((241,9 + 180)/180) = 2,34$. Com isso, tem-se um valor de referência para discussões. Observa-se que a estratégia de escalonamento proposta obteve resultados satisfatórios com fatores de folga menores que o estimado no Google. Os resultados dos experimentados indicam que a convergência Nuvem-HPC pode envolver migração



(a) 1 processador requisitado



(b) 2 processadores requisitados



(c) 4 processadores requisitados

Figura 6.2: Resultados relativos a violações de tempo de resposta.

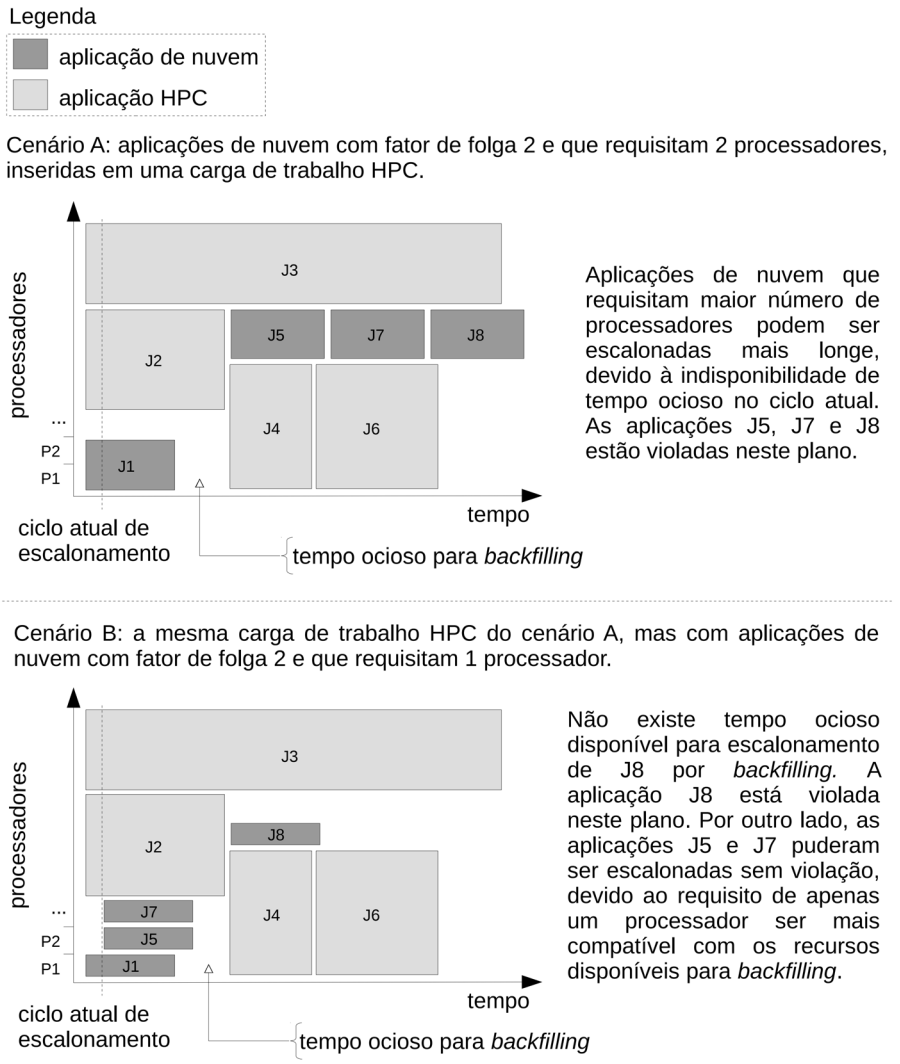
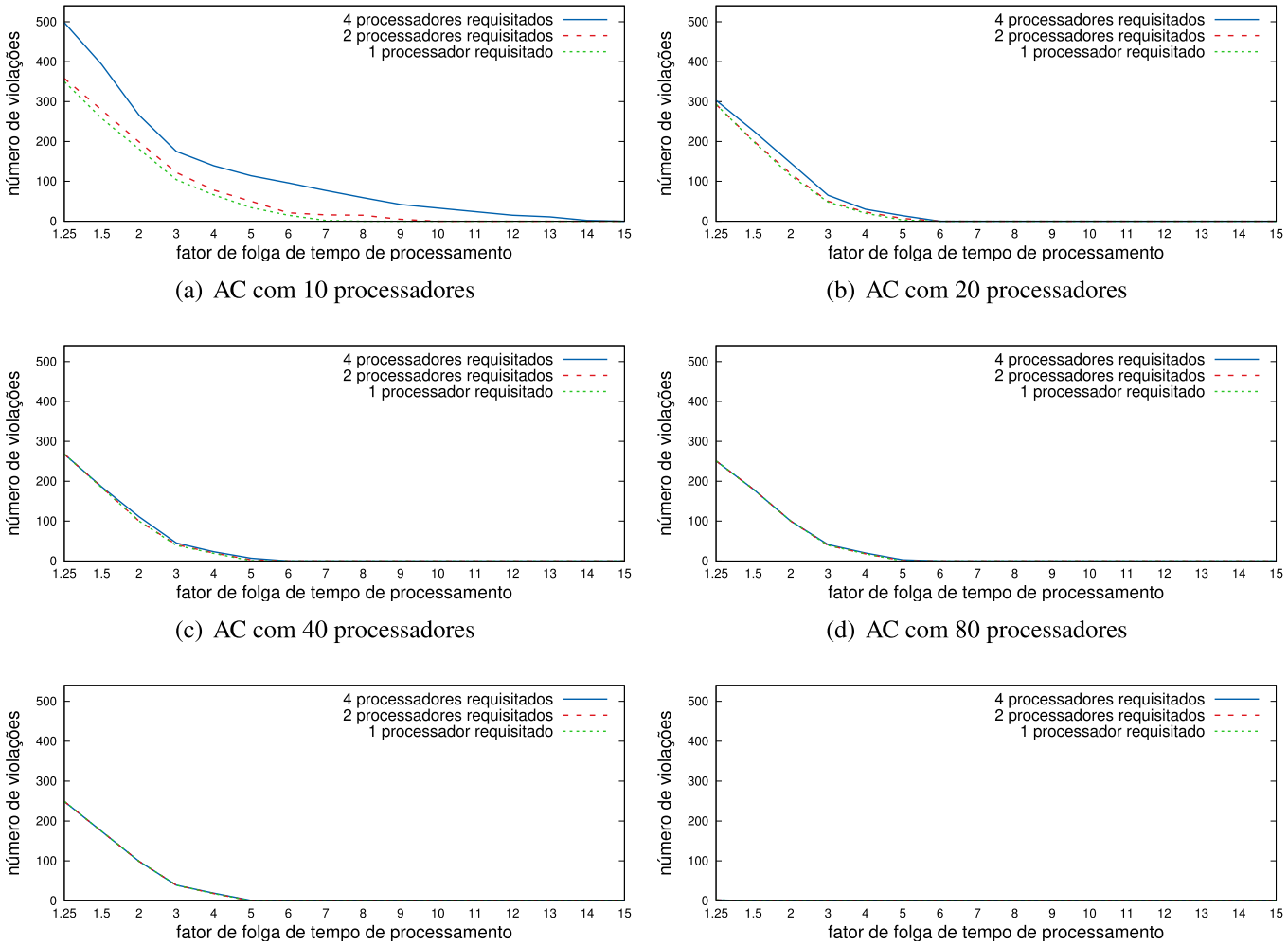


Figura 6.3: Exemplo de aumento do número de violações devido ao maior número de processadores requisitado por aplicações de nuvem.

e execução de aplicações de nuvem de maior prioridade, além das denominadas *gratis*, pois obteve-se resultados satisfatórios utilizando fatores de folga 1,25 e 1,5.

Embora as características discutidas tenham impacto sobre o número de violações, existem outros detalhes importantes observados nos resultados, que são discutidos a seguir. Para análises sob outra perspectiva, apresenta-se uma nova organização dos resultados na Figura 6.4. Nessa nova organização, observa-se que há um número próximo de violações nos casos *b*, *c*, *d* e *e*, não importando o número de processadores requisitado por aplicações e o tamanho da área de convergência (AC). Isso ocorre porque as situações de escalonamento com violação são as mesmas para qualquer aplicação de nuvem quando existem aplicações HPC que requisitam a maioria dos processadores na AC. A ampliação do tamanho da AC não impacta demasiadamente nas violações observadas nesses casos porque as aplicações HPC continuam requisitando a maioria do número de processadores disponíveis. Apresenta-se uma abstração desse escalonamento na Figura 6.5.



(a) AC com 10 processadores

(b) AC com 20 processadores

(c) AC com 40 processadores

(d) AC com 80 processadores

(e) AC com 160 processadores

(f) AC com 320 processadores

Figura 6.4: Resultados relativos a violações de tempo de resposta sob a perspectiva de diferentes tamanhos de área de convergência (ou AC).

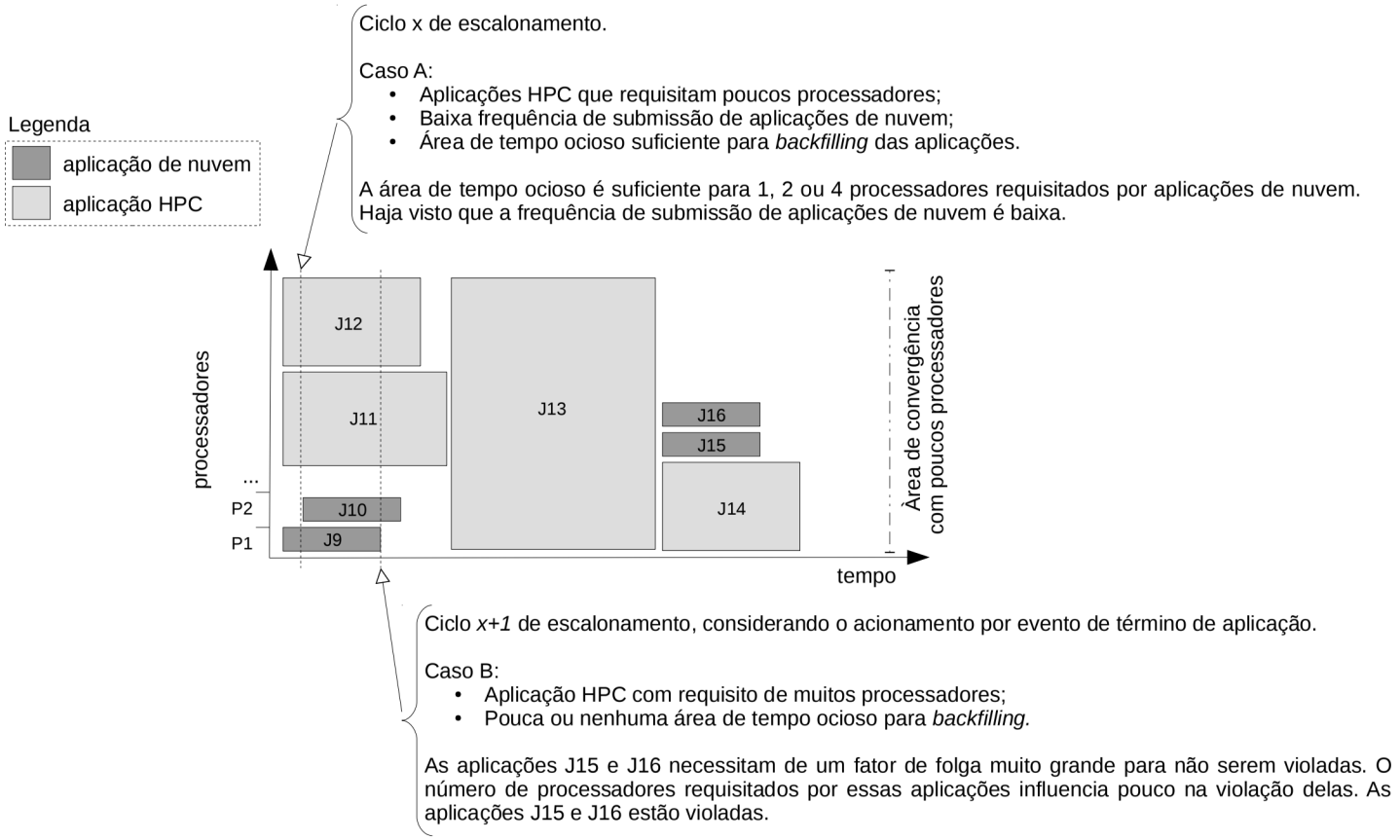


Figura 6.5: Exemplo de situação de escalonamento onde o número de processadores requisitado por aplicações de nuvem não tem impacto significativo no número de violações.

Ainda em relação ao tamanho da AC, há exceções nos casos *a* e *f* da Figura 6.4. Analisando-se o caso *a*, quando a AC é muita pequena, os *slots* de tempo ocioso podem ser muito pequenos também. Para aplicações que requisitam 4 processadores, os *slots* adequados ao escalonamento aparecem com menor frequência nesse contexto. Por isso, há diferença maior de violação entre os diferentes números de processadores requisitados. Por outro lado, analisando-se o caso *f*, quando a AC é grande o suficiente, as aplicações HPC não requisitam a maior parte dos processadores disponíveis. Assim, surgem muitos *slots* de tempo ocioso e todas as aplicações de nuvem podem ser escalonadas sem violação. Resultados sem violação foram observados em todas as configurações de cenário com tamanho de AC acima de 320 processadores. Por este motivo, esses resultados foram removidos dos gráficos, que contêm somente os dados mais relevantes relativos à violação. De maneira simplista, apresenta-se um caso similar na Figura 6.6, onde o número de violações foi reduzido devido a ampliação do tamanho da AC.

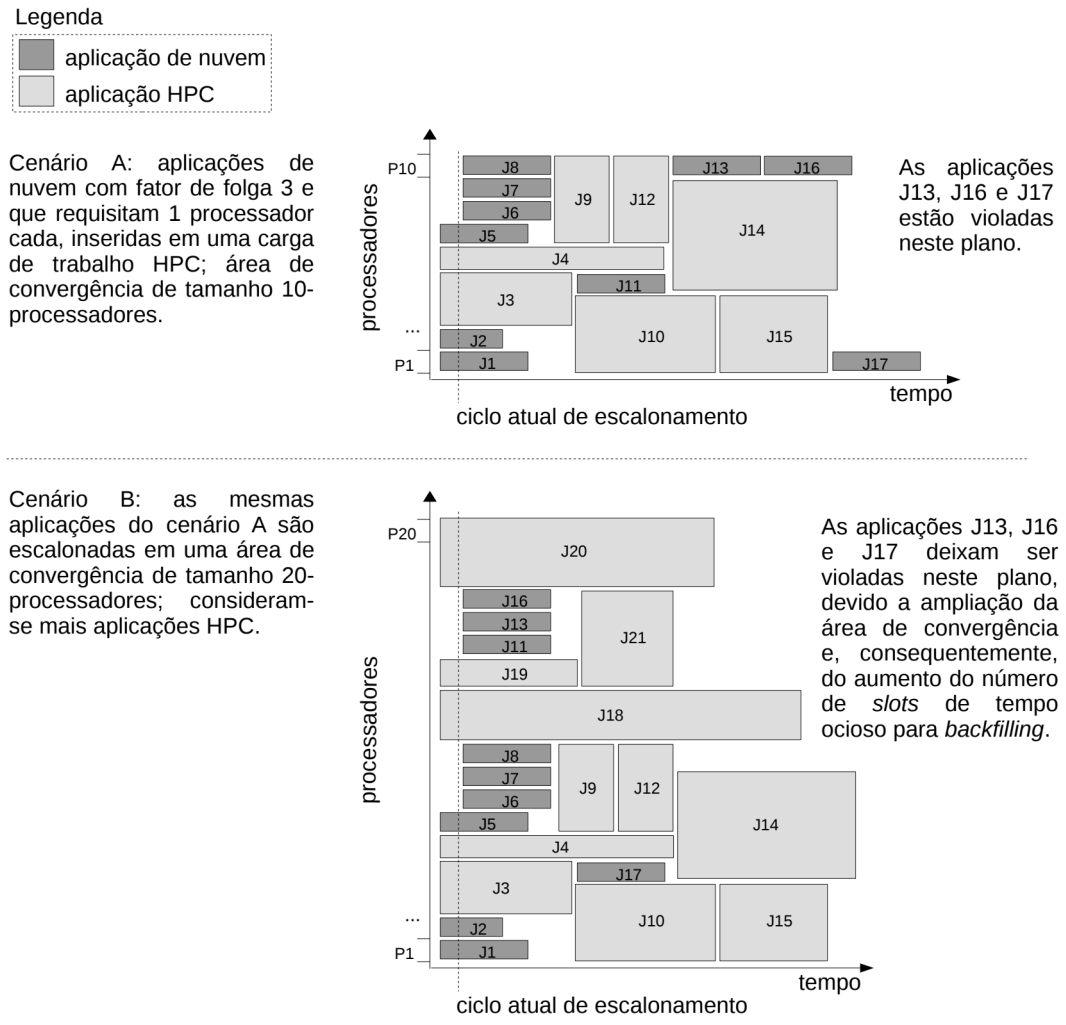
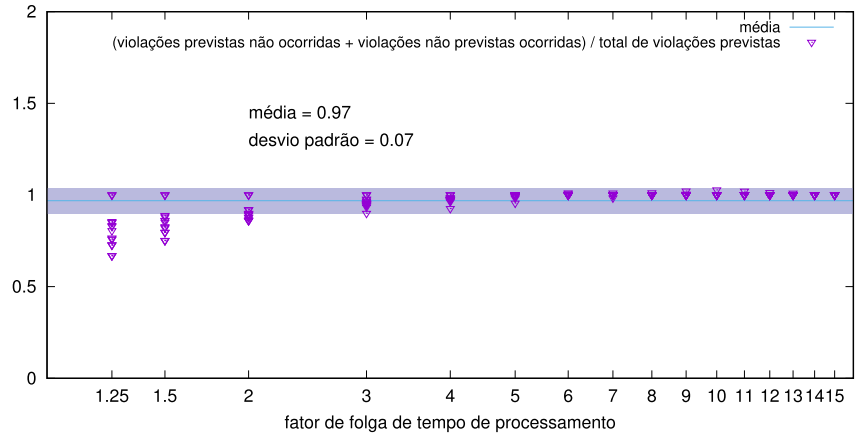


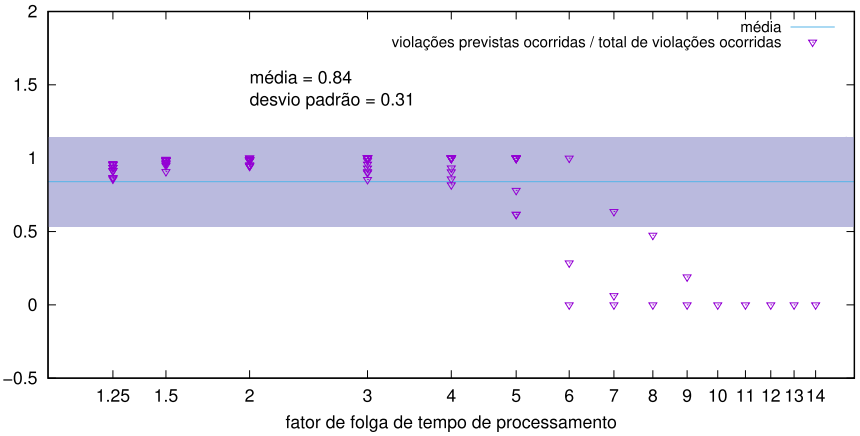
Figura 6.6: Exemplo de redução do número de violações devido à ampliação do tamanho da AC.

6.8.2 Eficácia da estratégia de previsão

Apresentam-se na Figura 6.7 os resultados relativos a eficácia da previsão. No cálculo das métricas de eficácia e na representação gráfica não foram considerados os resultados envolvendo mais de 320 processadores na área de convergência. Esse decisão foi tomada porque



(a) Identificação da acurácia



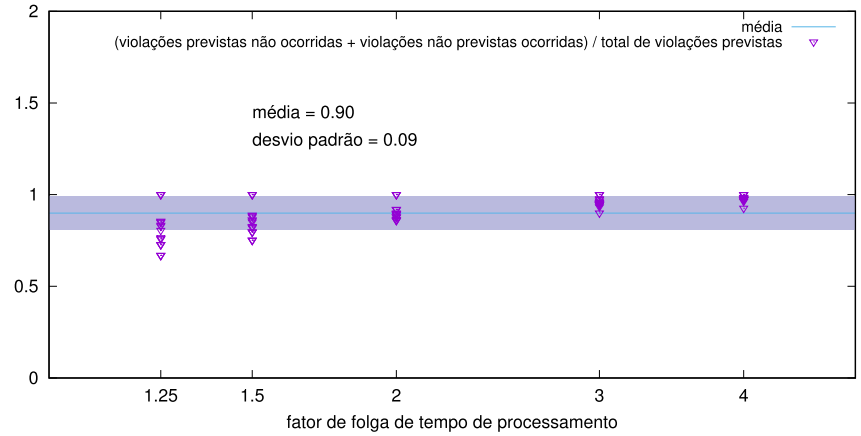
(b) Identificação da precisão

Figura 6.7: Resultados relativos a estratégia de previsão de tempo de resposta.

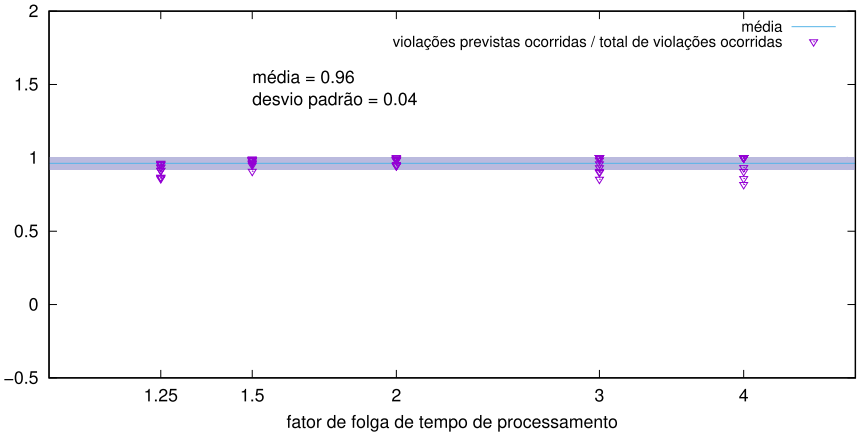
não ocorreram violações nesse contexto e a estratégia de previsão também não previu violação. Analisando-se essa figura, observa-se acurácia 0,97 e precisão 0,84. A acurácia 0,97 significa que a estratégia abrange muitos erros que não ocorrem, o que não é necessariamente uma conclusão negativa. A conclusão torna-se efetivamente negativa quando considerada a variabilidade de 0,07, chegando a acurácia 1,04. Esse resultado representa a ocorrência de mais violações não previstas, em relação as previstas corretamente.

Uma razão para essa conclusão negativa deve-se ao fato de que após o fator de folga 4 ocorre um número menos expressivo de violações, onde o erro de previsão de apenas uma aplicação tem muito impacto no resultado global. Para exemplificar, na AC com 10 processadores ocorreram 34 violações de aplicação que requisita 1 processador e fator de folga 5. A precisão desse experimento foi de 0,61, com 21 violações ocorridas como previsto (ou $\frac{21}{34} = 0,61$). Ao considerar-se 0,61 e outros resultados similares de outros experimentos, referentes ao mesmo contexto negativo discutido, a média aritmética que determina a precisão global é impactada negativamente. Analisando-se a Figura 6.7, nota-se que após o fator de folga 4 a estratégia de previsão torna-se irrelevante no cenário explorado. Portanto, descartam-se valores de precisão obtidos após o uso desse fator e discute-se a nova classificação a seguir.

Os resultados da nova classificação apresentados na Figura 6.8 mostram que a previsão é somente relevante para aplicações com maior urgência de execução (fator de folga menor que 5). Nesses casos foi determinada a acurácia 0,90 e a precisão 0,96. Uma razão para o número de



(a) Determinação da acurácia



(b) Determinação da precisão

Figura 6.8: Resultados dos experimentos para determinação da eficácia da estratégia de previsão desconsiderando valores de pouca relevância.

violações ser menor que o previsto (ou acurácia menor que 1) é devido ao *backfilling*. Quando uma aplicação termina, aciona-se um novo ciclo de escalonamento onde todos os escalonamentos anteriores são refeitos, tal como discutido nas características do simulador na Seção 6.2. Como resultado desse processo, algumas aplicações são reescalonadas para mais perto da linha do tempo corrente, o que diminui o valor previsto de momento de inicialização de cada aplicação e, consequentemente, diminui-se o número de violações previstas. A estratégia de previsão não antecipa-se a essa situação de encurtamento do escalonamento e consequentemente ocorrem erros de previsão. Por outro lado, considerando-se a variabilidade dos resultados, em alguns casos a estratégia proposta pode-se chegar à resultados próximos ao melhor método *Random Forest* analisado por [Hemmat e Hafid, 2016], onde determiram-se acurácia 0,99 e precisão 0,99, sendo estes os melhores resultados encontrados na literatura.

No trabalho de [Hemmat e Hafid, 2016] foram avaliadas as violações da carga de trabalho do Google por meio de vários métodos matemáticos de aprendizado de máquina, envolvendo *Naive Bayes* e *Random Forest*. Estimou-se que o número de violações é cerca de 2,2% considerando o não reescalonamento de aplicações que foram despejadas pelo escalonador. Com base nesse conjunto de dados, aplicaram-se os vários métodos para prevenção de futuras violações. Como resultado desse estudo, o método *Random Forest* obteve o melhor desempenho com acurácia 0,99 e precisão 0,99; e o melhor método *Naive Bayes* obteve acurácia 0,91 e precisão 0,91.

6.8.3 Impacto no HPC devido a aplicações de nuvem

Embora os resultados apresentados nas seções anteriores indiquem que as aplicações candidatas consideradas e o tamanho da área de convergência possam ser gerenciados de maneira a reduzir o número de violações, no destino das aplicações encontra-se a plataforma HPC, onde diversos trabalhos na literatura objetivam reduzir o *makespan* da sua carga de trabalho. Com base nessa discussão, não é desejável que a convergência impacte no *makespan* HPC.

Analisando-se os resultados de todos os experimentos, não foi identificada alteração significativa de *makespan*, devido a inserção de aplicações de nuvem, que impossibilitou a produção de gráficos. Devido a isso, nesta seção apresentam-se abstrações de escalonamento por meio de figuras que contribuem para o entendimento do por que não houve alteração no *makespan*. A primeira abstração discutida está representada na Figura 6.9 e não corresponde ao observado nos experimentos. Entretanto, essa figura é útil quando comparada com outro contexto: o da Figura 6.10, onde apresenta-se uma abstração tal como ocorreu nos experimentos.

Observa-se por meio da abstração da Figura 6.9 que o *makespan* da carga de trabalho HPC aumenta ao inserir-se um número elevado de aplicações de nuvem. Isso ocorre porque os *slots* de tempo ocioso não são suficientes para alocação de todas as aplicações de nuvem, fazendo com que elas sejam escalonadas após as aplicações HPC, pelo 3º estágio da estratégia de escalonamento, onde não aplica-se *backfilling*. O aumento do *makespan* é potencializado com a chegada de novas aplicações HPC e o seu escalonamento mais à frente na linha de tempo, devido a ocupação mais cedo de uma ampla área de tempo pelas aplicações de nuvem inseridas. Esse contexto é difícil de ser observado, pois em geral as cargas de trabalho HPC possuem muito tempo ocioso, com ampla capacidade de inserção de aplicações de nuvem como as classificadas como candidatas, as quais requisitam poucos recursos.

A abstração da Figura 6.10 ilustra o escalonamento ocorrido nos experimentos. Nesse escalonamento estão aplicações HPC submetidas de maneira esparsa, proporcionando muitos *slots* de tempo ocioso para aplicações de nuvem. Mesmo nos experimentos com área de convergência

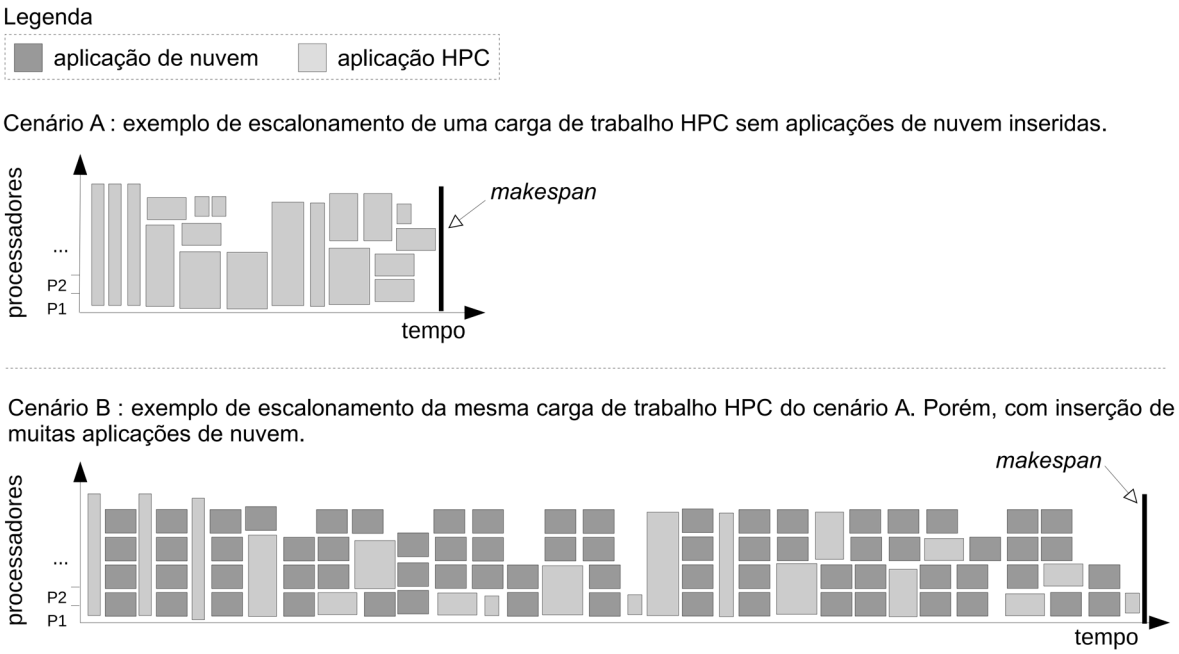


Figura 6.9: Exemplos de *makespan* considerando dois cenários de escalonamento para a mesma carga de trabalho HPC: um sem aplicações de nuvem e outro com impacto devido a muitas aplicações de nuvem inseridas.

de pequeno tamanho, a frequência de submissão de aplicações de nuvem não foi suficiente para ampliar o *makespan* do último subconjunto de aplicações HPC.

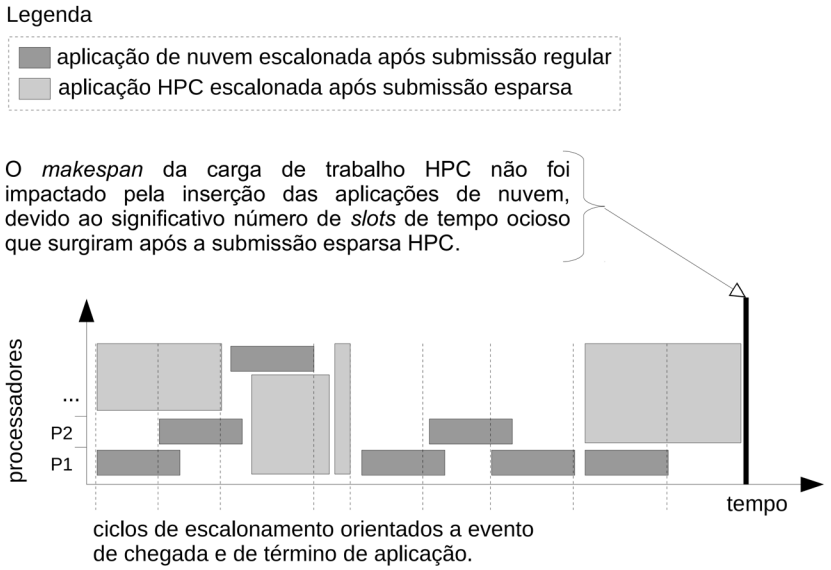


Figura 6.10: Exemplo de *makespan* não impactado pela inserção de aplicações de nuvem.

A taxa de ocupação dos recursos da carga de trabalho exclusivamente HPC dos experimentos é de 38%. Porém, segundo [Feitelson, 2016], as taxas de cargas de trabalho HPC em diversos grids podem variar entre 10% e 88%. Devido a essa ampla variação, não há, portanto, forma de relacionar um padrão de carga de trabalho HPC com uma frequência de submissão de aplicações de nuvem mais adequada para redução do número de violações. Assim, a melhor relação entre aplicações HPC e de nuvem depende de cada caso. No caso do cenário explorado,

considera-se que a configuração adotada foi satisfatória, pois não houve aumento do *makespan* e a plataforma HPC conseguiu absorver as aplicações de nuvem com baixo percentual de violações de tempo de resposta.

6.9 Considerações finais

Este capítulo apresentou a validação da proposta por meio de simulação *in silico*, utilizando dados de aplicações reais. Apresentaram-se a metodologia dos experimentos, características do simulador, detalhes dos experimentos, resultados e discussão. Identificaram-se as seguintes características que têm impacto no número de violações: carga de trabalho HPC, tamanho da área de convergência em número de processadores, frequência de submissão de aplicações de nuvem, características das aplicações de nuvem e principalmente o fator de folga de tempo de processamento dessas aplicações. O maior percentual de violações determinado foi de 0,0018, que é considerado baixo se comparado com o percentual de 0,2 determinado por [Hemmat e Hafid, 2016] sobre sistemas de computação. Ressalta-se que as aplicações de nuvem são aplicações candidatas, ou seja, selecionadas de maneira a não impactar na carga de trabalho HPC. Por isso, o percentual de violações obtido nos experimentos é menor que o encontrado na literatura, que considera todo tipo de aplicação.

A estratégia de previsão de violação foi considerada satisfatória com precisão 0,96 para aplicações com pequeno fator de folga. Para aplicações com fator de folga maior que 4, o percentual de violações é inexpressivo, não havendo necessidade de aplicação da estratégia de previsão. Nesse contexto, o trabalho de [Costache et al., 2013] destaca a importância de aceitar aplicações com tempo de término previsto conforme as capacidades do escalonador, criando assim uma tabela de valores aceitáveis que é utilizada na negociação do SLA. Com base nos resultados apresentados na Figura 6.2, foi identificado no cenário o valor 15 de fator de folga prevendo a ocorrência de zero número de violações na carga de trabalho HPC. Esse fator de folga pode ser utilizado como referência na negociação entre a Central de Dados considerada e seus usuários, tal como propõe [Costache et al., 2013].

Por fim, propõe-se que em cenários reais de centrais de dados a frequência de submissão seja monitorada (ou regulada ao longo do tempo por escalonadores que realizam migração), para equilibrar o escalonamento evitando aumento do número de violações. Por todas essas considerações, a área de convergência nuvem-HPC é considerada factível de redução do número de violações de tempo de resposta SLO, podendo causar pouco impacto sobre demais aplicações em plataformas HPC.

Capítulo 7

Conclusão

Neste trabalho foram apresentadas uma proposta de área de convergência nuvem-HPC e estratégias para escalonadores em centrais de dados, onde o objetivo é reduzir o número de violações de tempo de resposta SLO. No Capítulo 2 foram apresentadas uma arquitetura de Central de Dados com plataformas de nuvem e de HPC, exemplos de Centrais Dados bem conhecidas e o conceito de SLA. Foram apresentados brevemente os RJMSs das Centrais de Dados do Grid'5000, Facebook, Twitter, eBay, Yahoo e Alibaba. Destacou-se o Google no decorrer das revisões, onde foi identificado um problema relacionado ao despejo de aplicações e violações.

No Capítulo 3 foram apresentados diversos conceitos e mecanismos para solução de problemas relativos a escalonamento de aplicações. Nesse capítulo também foram identificados diversos desafios envolvendo escalonamento e trabalhos relativos a SLA em nuvem. Em centrais de dados onde existe um modelo de negócio envolvendo cliente e provedor, um SLA é estabelecido para garantir a disponibilidade do serviço. Algumas das cláusulas do SLA são chamadas de SLO. Como exemplo, cita-se o percentual tolerável de violações de tempo de resposta de aplicações submetidas pelo programa do usuário.

Com base nos levantamentos realizados nos Capítulos 2 e 3, observou-se que plataformas HPC são em geral mais ociosas que plataformas de nuvem. Segundo [Feitelson, 2016], a taxa de ocupação de plataformas HPC pode variar entre 10% e 88%. Esses percentuais indicam que algumas plataformas são muito ociosas. Elas poderiam estar sendo aproveitadas para execução de mais aplicações. Em contrapartida, em nuvem há sobrecarga de número de aplicações, onde aplicações de menor prioridade são despejadas para darem lugar ao processamento de outras de maior prioridade. Essa sobrecarga e despejo faz com que muitas aplicações sejam violadas, podendo ocorrer descumprimento do SLO. Segundo [Hemmat e Hafid, 2016], o percentual de violações nesses sistemas é cerca de 0,2%. Assim, considerou-se a hipótese que para determinados tipos de aplicação de nuvem, denominadas neste trabalho como aplicações candidatas, as plataformas HPC podem ser úteis à redução do número de violações. As aplicações candidatas são curtas em tempo de processamento e requisitam poucos processadores. Determinaram-se essas características como forma dessas aplicações não impactarem demasiadamente no escalonamento das demais aplicações na área de convergências. Devido à caracterização dessas aplicações, aproveitam-se melhor os *slots* de tempo ocioso presentes em plataformas HPC, por meio do mecanismo *backfilling*. Levou-se também em consideração que não é comum a habilitação de controle de tempo de resposta para verificação de violações em plataformas HPC, sendo, portanto, também necessária uma proposta de escalonamento HPC para habilitação da convergência.

No Capítulo 4 foi apresentada uma proposta de área de convergência nuvem-HPC, onde constam os fundamentos, um modelo de fluxos de aplicação dentro da Central de Dados, os elementos essenciais da arquitetura e as funções de previsão de tempo de resposta. O uso da área de convergência com o objetivo de reduzir o número de violações de tempo de resposta é assunto do Capítulo 5, onde apresentaram-se estratégias para o escalonador principal da Central de Dados, escalonadores de nuvem e escalonadores HPC.

No Capítulo 6 de validação foi apresentado o CArS (*Convergence Area Scheduler* - Escalonador de Área de Convergência), que foi desenvolvido como uma ramificação do RJMS OAR, contendo características de centrais de dados. Os experimentos no CArS envolveram carga de trabalho de produção HPC com 11K aplicações e um conjunto de aplicações migradas de nuvem. As aplicações de nuvem foram criadas com base em levantamento de aplicações executadas no Google. Apenas aplicações candidatas foram migradas, totalizando 267K aplicações de nuvem. Destaca-se que ambas as aplicações de nuvem e de HPC referem-se ao mesmo mês de processamento real, que ocorreu em maio de 2011. Portanto, a validação trata de simulação com características de aplicações reais, *in silico*. Determinaram-se ainda no Capítulo 6 as métricas para o percentual de violações de tempo de resposta, a eficácia da previsão de violação e o impacto na carga de trabalho HPC devido à inserção das aplicações de nuvem.

Os resultados das simulações mostraram que devido ao amplo conjunto de *slots* de tempo ocioso na carga de trabalho HPC, as aplicações de nuvem ditas candidatas são executadas com baixo número de violações. Identificou-se que a principal característica de aplicação que impacta no número de violações é o fator de folga de tempo de processamento, pois esse atributo determina o quão tarde a aplicação pode ser executada após a sua chegada na fila de espera. Discutiram-se também situações de escalonamento que levam uma aplicação à violação. Considerando-se as análises dos resultados, o maior percentual de violação (0,0018%) ocorreu na área de convergência de menor tamanho (10 processadores), para aplicações com baixo fator de folga (1,25) e maior número de processadores requisitados (4 processadores). Essas aplicações são as que mais requisitaram recursos e o tamanho da área de convergência foi o menor considerado nos experimentos. Esse percentual é considerado baixo se comparado com o 0,2% destacado por [Hemmat e Hafid, 2016] como valor médio de violação em sistemas de computação em larga escala. O fator de folga 1,25 dessas aplicações mais violadas também é um dado importante, pois indica que aplicações mais urgentes que as candidatas no Google podem ser executadas em HPC com baixo percentual de violação. No Google estimou-se um fator de folga mínimo de 2,34.

Determinou-se também que a estratégia de previsão de violação de tempo de resposta possui acurácia 0,90 e precisão 0,96. Pelo resultado da acurácia, entende-se que muitas previsões de violação acabaram não ocorrendo. Por outro lado, o resultado da precisão mostra que a maior parte das violações que ocorreram foram previstas corretamente, as vezes todas para determinado fator de folga se considerada a variabilidade de 0,4. O estudo realizado por [Hemmat e Hafid, 2016] determinou a melhor e a pior precisão ao analisar diversos métodos. A melhor precisão determinada por esses autores é do método *Random Forest*, com 0,99 e a pior do método *Naives Bayes* com 0,91. Comparando-se com a precisão determinada na área de convergência, considera-se que o resultado 0,96 é satisfatório, estando próximo do resultado de outros métodos.

Por fim, concluiu-se que o *makespan* da carga de trabalho HPC não foi impactado pela inserção das aplicações de nuvem. Isso deve-se pela habilitação do *backfilling* e pelo amplo conjunto de *slots* de tempo ocioso presente na plataforma HPC, devido à submissão de aplicações de maneira esparsa que frequentemente ocorre nessas plataformas.

Os autores [Zhang et al., 2014] apresentaram que Centrais de Dados como Google, Microsoft e Facebook, possuem características semelhantes de aplicações, com grande variação entre seus tamanhos. Com isso, entende-se que também existem aplicações candidatas à migração de nuvem para HPC em outras companhias. A proposta de área de convergência mostrou-se relevante para redução do número de violações de tempo de resposta considerando aplicações com características do Google e podem, portanto, contribuir para outras companhias também.

Com base nessas discussões apresentadas, este trabalho contribuiu com a definição de uma área de convergência em centrais de dados, uma estratégia de previsão de violação, fluxos de controle de aplicação considerando migração de nuvem para HPC, estratégias para escalonadores e uma estratégia de escalonamento nuvem-HPC, visando a redução do número de violações de tempo de resposta SLO. Não foram encontrados trabalhos similares na literatura para cada uma das estratégias apresentadas. O simulador CArS foi desenvolvido para esta proposta, sendo considerado também uma contribuição relevante. As publicações científicas [Kraemer et al., 2016, Kraemer et al., 2017] introduzem o tema de área de convergência na comunidade científica, sendo outras contribuições relevantes. O desenvolvimento do CArS e essas publicações contaram com a colaboração do MOAIS, que é responsável pelo desenvolvimento do RJMS OAR e pelo gerenciamento do Grid'5000 no INRIA. Por tais considerações, conclui-se que este trabalho apresentou contribuições significativas, sendo factível de implantação em centrais de dados.

Trabalhos futuros objetivarão o escalonamento de aplicações moldáveis e maleáveis. Para trabalhos na linha de pesquisa da nuvem Google, recomendam-se os estudos realizados por [Reiss et al., 2012, Reiss et al., 2011, Sheng et al., 2012, Sheng et al., 2013]. A respeito de violação em sistemas de computação de larga escala, recomenda-se o estudo realizado por [Hemmat e Hafid, 2016].

Referências Bibliográficas

[Abdul-Rahman e Aida, 2014] Abdul-Rahman, O. A. e Aida, K. (2014). Towards understanding the usage behavior of Google cloud users: The mice and elephants phenomenon. Em *IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom)*, páginas 272–277.

[Amani e Zamanifar, 2014] Amani, A. e Zamanifar, K. (2014). Improving the time of live migration virtual machine by optimized algorithm scheduler credit. Em *4th International Conference on Computer and Knowledge Engineering (ICCKE)*, páginas 346–351.

[Atif e Strazdins, 2009] Atif, M. e Strazdins, P. (2009). Optimizing live migration of virtual machines in SMP clusters for HPC applications. Em *Sixth IFIP International Conference on Network and Parallel Computing*, páginas 51–58.

[Badia et al., 2013] Badia, S., Carpen-Amarie, A., Lebre, A. e Nussbaum, L. (2013). Enabling large-scale testing of iaas cloud platforms on the grid5000 testbed. Em *Proceedings of the 2013 International Workshop on Testing the Cloud*, TTC 2013, páginas 7–12, New York, NY, USA. ACM.

[Barquet et al., 2013] Barquet, A. L., Tchernykh, A. e Yahyapour, R. (2013). Performance evaluation of infrastructure as a service clouds with SLA constraints. Em *Iberoamerican Journal of Research Computing and Systems (Revista Iberoamericana de Investigación "Computación y Sistemas"), vol. 17*, página 401–411.

[Becchetti et al., 2000] Becchetti, L., Leonardi, S. e Muthukrishnan, S. (2000). Scheduling to minimize average stretch without migration. Em *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, páginas 548–557, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[Bender et al., 1998] Bender, M. A., Chakrabarti, S. e Muthukrishnan, S. (1998). Flow and stretch metrics for scheduling continuous job streams. Em *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '98, páginas 270–279, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[Bender et al., 2004] Bender, M. A., Muthukrishnan, S. e Rajaraman, R. (2004). Approximation algorithms for average stretch scheduling. *Journal of Scheduling*, 7(3):195–222.

[Birkenheuer, 2012] Birkenheuer, G. (2012). *Risk Aware Overbooking for SLA Based Scheduling Systems*. Dissertação de mestrado, University of Paderborn.

[Birkenheuer e Brinkmann, 2011] Birkenheuer, G. e Brinkmann, A. (2011). Reservation-based overbooking for HPC clusters. Em *IEEE International Conference on Cluster Computing (CLUSTER)*, páginas 537–541.

[Bougeret et al., 2015] Bougeret, M., Dutot, P., Trystram, D., Jansen, K. e Robenek, C. (2015). Improved approximation algorithms for scheduling parallel jobs on identical clusters. *Theor. Comput. Sci.*, 600(C):70–85.

[Braschi e Trystram, 1994] Braschi, B. e Trystram, D. (1994). A new insight into the Coffman-Graham algorithm. *SIAM J. Comput.*, 23(3):662–669.

[Breitgand et al., 2007] Breitgand, D., Henis, E. A., Shehory, O. e Lake, J. M. (2007). Derivation of response time service level objectives for business services. Em *2nd IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM)*, páginas 29–38.

[Caglar e Gokhale, 2014] Caglar, F. e Gokhale, A. (2014). iOverbook: Intelligent resource-overbooking to support soft real-time applications in the cloud. Em *Cloud Computing (CLOUD)*, páginas 538–545.

[Cardoso e Valette, 1997] Cardoso, J. e Valette, R. (1997). *Redes de Petri*. UFSC, Florianópolis.

[Casanova et al., 2014] Casanova, H., Giersch, A., Legrand, A., Quinson, M. e Suter, F. (2014). Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917.

[Chanchio e Thaenkaew, 2014] Chanchio, K. e Thaenkaew, P. (2014). Time-bound, thread-based live migration of virtual machines. Em *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, páginas 364–373.

[Chauhan e Babar, 2012] Chauhan, M. A. e Babar, M. A. (2012). Towards process support for migrating applications to cloud computing. Em *Cloud and Service Computing (CSC)*, páginas 80–87.

[Cherkasova et al., 2007] Cherkasova, L., Gupta, D. e Vahdat, A. (2007). Comparison of the three CPU schedulers in Xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51.

[Chopra e Singh, 2013] Chopra, N. e Singh, S. (2013). Deadline and cost based workflow scheduling in hybrid cloud. Em *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, páginas 840–846.

[Costache et al., 2013] Costache, S., Parlavantzas, N., Morin, C. e Kortas, S. (2013). Merkat: A market-based SLO-driven cloud platform. Em *IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, volume 1, páginas 403–410.

[den Bossche et al., 2010] den Bossche, R. V., Vanmechelen, K. e Broeckhove, J. (2010). Cost-optimal scheduling in hybrid IaaS clouds for deadline constrained workloads. Em *IEEE 3rd International Conference on Cloud Computing*, páginas 228–235.

[Dutot et al., 2016] Dutot, P., Mercier, M., Poquet, M. e Richard, O. (2016). BatSim: a realistic language-independent resources and jobs management systems simulator. Em *20th Workshop on Job Scheduling Strategies for Parallel Processing*, Chicago, United States.

[Dutot et al., 2004] Dutot, P., Mounie, G. e Trystram, D. (2004). Scheduling parallel tasks: Approximation algorithms. Em Leung, J. T., editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, chapter 26, páginas 26–1 – 26–24. CRC Press.

[Elmroth et al., 2009] Elmroth, E., Marquez, F. G., Henriksson, D. e Ferrera, D. P. (2009). Accounting and billing for federated cloud infrastructures. Em *Eighth International Conference on Grid and Cooperative Computing*, páginas 268–275.

[Emeras et al., 2013] Emeras, J., Ruiz, C., Vincent, J. e Richard, O. (2013). Analysis of the jobs resource utilization on a production system. Em Cirne, W., Desai, N., Frachtenberg, E. e Schwiegelshohn, U., editores, *Job Scheduling Strategies for Parallel Processing*, Lecture Notes in Computer Science, Boston, United States. Springer.

[Facebook-team, 2012] Facebook-team (2012). Under the hood: Scheduling MapReduce jobs more efficiently with corona.

[Feitelson, 2016] Feitelson, D. (2016). Logs of real parallel workloads from production systems. `http://www.cs.huji.ac.il/labs/parallel/workload/logs.html`. Acessado em 15/07/2016.

[Feitelson et al., 1997] Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C. e Wong, P. (1997). Theory and practice in parallel job scheduling. Em *Proceedings of the Job Scheduling Strategies for Parallel Processing*, IPPS '97, páginas 1–34, London, UK, UK. Springer-Verlag.

[Freitas et al., 2011] Freitas, A. L., Parlavantzas, N. e Pazat, J. L. (2011). Cost reduction through SLA-driven self-management. Em *Ninth IEEE European Conference on Web Services (ECOWS)*, páginas 117–124.

[Georgiou, 2006] Georgiou, Y. (2006). *Resource and Job Management in High Performance Computing*. Tese de doutorado, University of Grenoble.

[Gibbons, 1997] Gibbons, R. (1997). *A historical application profiler for use by parallel schedulers*, páginas 58–77. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Graham et al., 1979] Graham, R., Lawler, E., Lenstra, J. e Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Em Hammer, P., Johnson, E. e Korte, B., editores, *Discrete Optimization II - Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, volume 5 de *Annals of Discrete Mathematics*, páginas 287 – 326. Elsevier.

[Graham, 1966] Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45:1563–1581.

[Gupta et al., 2013] Gupta, A., Kale, L. V., Milojicic, D. S., Faraboschi, P., Kaufmann, R., March, V., Gioachin, F., Suen, C. H. e Lee, B. (2013). The who, what, why and how of high performance computing applications in the cloud. Em *Proceedings of the 5th IEEE International Conference on Cloud Computing Technology and Science*, CloudCom '13.

[gyu Kim et al., 2012] gyu Kim, S., Eom, H. e Yeom, H. Y. (2012). Virtual machine scheduling for multicores considering effects of shared on-chip last level cache interference. Em *International Green Computing Conference (IGCC)*, páginas 1–6.

[Hafshejani et al., 2013] Hafshejani, Z. M., Mirtaheri, S. L., Khaneghah, E. M. e Sharifi, M. (2013). An efficient method for improving backfill job scheduling algorithm in cluster computing systems. *International Journal of Soft Computing and Software Engineering [JSCSE]*, páginas 422–429. 10.7321/jscse.v3.n3.64.

[Hedwig et al., 2012] Hedwig, M., Malkowski, S. e Neumann, D. (2012). Risk-aware service level agreement design for enterprise information systems. Em *45th Hawaii International Conference on System Science (HICSS)*, páginas 4552–4561.

[Hemmat e Hafid, 2016] Hemmat, R. A. e Hafid, A. (2016). SLA violation prediction in cloud computing: A machine learning perspective. *CoRR*, abs/1611.10338.

[Hindman et al., 2011] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R., Shenker, S. e Stoica, I. (2011). Mesos: A platform for fine-grained resource sharing in the data center. Em *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, páginas 295–308, Berkeley, CA, USA. USENIX Association.

[Hirales-Carbajal et al., 2010] Hirales-Carbajal, A., Tchernykh, A., Röblitz, T. e Yahyapour, R. (2010). A grid simulation framework to study advance scheduling strategies for complex workflow applications. Em *IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW)*, páginas 1–8.

[Hirales-Carbajal et al., 2012] Hirales-Carbajal, A., Tchernykh, A., Yahyapour, R., Gonzalez-Garcia, J. L., Roblitz, T. e Ramirez-Alcaraz, J. M. (2012). Multiple workflow scheduling strategies with user run time estimates on a grid. *Journal of Grid Computing*, 10(2):325–346.

[IBM, 2009] IBM, C. C. (2009). Seeding the clouds: Key infrastructure elements for cloud computing.

[Iglesias et al., 2014] Iglesias, J. O., Murphy, L., Cauwer, M. D., Mehta, D. e O'Sullivan, B. (2014). A methodology for online consolidation of tasks through more accurate resource estimations. Em *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, páginas 89–98.

[Jayathilaka et al., 2015] Jayathilaka, H., Krintz, C. e Wolski, R. (2015). Response time service level agreements for cloud-hosted web applications. Em *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, páginas 315–328, New York, NY, USA. ACM.

[Jhawar e Piuri, 2015] Jhawar, R. e Piuri, V. (2015). *Dependability-Oriented Resource Management Schemes for Cloud Computing Data Centers*, páginas 1285–1305. Springer New York, New York, NY.

[JTC, 2014] JTC (2014). Information technology – cloud computing – overview and vocabulary. Relatório Técnico ISO/IEC 17788:2014(E), Joint Technical Committee JTC.

[Kapil et al., 2013] Kapil, D., Pilli, E. S. e Joshi, R. C. (2013). Live virtual machine migration techniques: Survey and research challenges. Em *3rd IEEE International Advance Computing Conference (IACC)*, páginas 963–969.

[Kivity et al., 2014] Kivity, A., L., D., C., G., E., P., Har'El, N., Marti, D. e Zolotarov, V. (2014). OSv—optimizing the operating system for virtual machines. Em *2014 USENIX*

Annual Technical Conference (USENIX ATC 14), páginas 61–72, Philadelphia, PA. USENIX Association.

[Koller, 2010] Koller, B. (2010). Enhanced SLA management in the high performance computing domain.

[Kraemer et al., 2016] Kraemer, A., Maziero, C., Richard, O. e Trystram, D. (2016). Reducing the number of response time SLO violations by a cloud-HPC convergence scheduler. Em *2nd International Conference on Cloud Computing Technologies and Applications (CloudTech)*, páginas 293–300.

[Kraemer et al., 2017] Kraemer, A., Maziero, C. A., Richard, O. e Trystram, D. (2017). Reducing the number of response time service level objective violations by a cloud-HPC convergence scheduler. *Concurrency and Computation: Practice and Experience*, páginas e4352–n/a. e4352 cpe.4352.

[Leitner et al., 2010] Leitner, P., Michlmayr, A., Rosenberg, F. e Dustdar, S. (2010). Monitoring, prediction and prevention of SLA violations in composite services. Em *IEEE International Conference on Web Services*, páginas 369–376.

[Lifka, 1995] Lifka, D. A. (1995). The ANL/IBM SP scheduling system. Em *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, IPPS '95, páginas 295–303, London, UK, UK. Springer-Verlag.

[Liu e Cho, 2012a] Liu, Z. e Cho, S. (2012a). Characterizing machines and workloads on a Google cluster. Em *2012 41st International Conference on Parallel Processing Workshops*, páginas 397–403.

[Liu e Cho, 2012b] Liu, Z. e Cho, S. (2012b). Characterizing machines and workloads on a google cluster. Em *2012 41st International Conference on Parallel Processing Workshops*, páginas 397–403.

[Maguluri e Srikant, 2013] Maguluri, S. T. e Srikant, R. (2013). Scheduling jobs with unknown duration in clouds. Em *Proceedings IEEE INFOCOM*, páginas 1887–1895.

[Mao e Humphrey, 2011] Mao, M. e Humphrey, M. (2011). Auto-scaling to minimize cost and meet application deadlines in cloud workflows. Em *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, páginas 49:1–49:12, New York, NY, USA. ACM.

[Margery et al., 2014] Margery, D., Morel, E., Nussbaum, L., Richard, O. e Rohr, C. (2014). *Resources Description, Selection, Reservation and Verification on a Large-Scale Testbed*, páginas 239–247. Springer International Publishing, Cham.

[Mehrotra et al., 2015] Mehrotra, R., Banicescu, I., Srivastava, S. e Abdelwahed, S. (2015). *A Power-Aware Autonomic Approach for Performance Management of Scientific Applications in a Data Center Environment*, páginas 163–189. Springer New York, New York, NY.

[Mishra et al., 2010] Mishra, A. K., Hellerstein, J. L., Cirne, W. e Chita, R. (2010). Towards characterizing cloud backend workloads: Insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37(4):34–41.

[Moschakis e Karatza, 2012] Moschakis, I. A. e Karatza, H. D. (2012). Parallel job scheduling on a dynamic cloud model with variable workload and active balancing. Em *16th Panhellenic Conference on Informatics (PCI)*, páginas 93–98.

[Mounié, 2000] Mounié, G. (2000). *Ordonnancement efficace d'application parallèles: les tâches malléeables monotones*. Tese de doutorado, INP Grenoble.

[Nguyen et al., 2013] Nguyen, Q. T., Quang-Hung, N., Tuong, N. H., Tran, V. H. e Thoai, N. (2013). Virtual machine allocation in cloud computing for minimizing total execution time on each machine. Em *International Conference on Computing, Management and Telecommunications (ComManTel)*, páginas 241–245.

[Patel e Bhoi, 2014] Patel, S. J. e Bhoi, U. R. (2014). Improved priority based job scheduling algorithm in cloud computing using iterative method. Em *Fourth International Conference on Advances in Computing and Communications (ICACC)*, páginas 199–202.

[Pawar e Wagh, 2013] Pawar, C. S. e Wagh, R. B. (2013). Priority based dynamic resource allocation in cloud computing with modified waiting queue. Em *International Conference on Intelligent Systems and Signal Processing (ISSP)*, páginas 311–316.

[Perez-Mendez et al., 2014] Perez-Mendez, A., Pereniguez-Garcia, F., Marin-Lopez, R., Lopez-Millan, G. e Howlett, J. (2014). Identity federations beyond the web: A survey. *IEEE Communications Surveys Tutorials*, 16(4):2125–2141.

[Phillips et al., 1997] Phillips, C. A., Stein, C., Torng, E. e Wein, J. (1997). Optimal time-critical scheduling via resource augmentation (extended abstract). Em *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, páginas 140–149, New York, NY, USA. ACM.

[Pore et al., 2015] Pore, M., Abbasi, Z., Gupta, S. K. S. e Varsamopoulos, G. (2015). *Techniques to Achieve Energy Proportionality in Data Centers: A Survey*, páginas 109–162. Springer New York, New York, NY.

[Quezada-Pina et al., 2012] Quezada-Pina, A., Tchernykh, A., Gonzalez-Garcia, J. L., Hirales-Carbajal, A., Ramirez-Alcaraz, J. M., Schwiegelshohn, U., Yahyapour, R. e Miranda-Lopez, V. (2012). Adaptive parallel job scheduling with resource admissible allocation on two-level hierarchical grids. *Future Gener. Comput. Syst.*, 28(7):965–976.

[Raju et al., 2013] Raju, R., Babukarthik, R. G., Chandramohan, D., Dhavachelvan, P. e Vengattaraman, T. (2013). Minimizing the makespan using hybrid algorithm for cloud computing. Em *Advance Computing Conference (IACC), 2013 IEEE 3rd International*, páginas 957–962.

[Ramirez-Alcaraz et al., 2011] Ramirez-Alcaraz, J. M., Tchernykh, A., Yahyapour, R., Schwiegelshohn, U., Quezada-Pina, A., Gonzalez-Garcia, J. L. e Hirales-Carbajal, A. (2011). Job allocation strategies with user run time estimates for online scheduling in hierarchical grids. *Journal of Grid Computing*, 9(1):95–116.

[Reiss et al., 2012] Reiss, C., Tumanov, A., Ganger, G. R., Katz, R. H. e Kozuch, M. A. (2012). Towards understanding heterogeneous clouds at scale: Google trace analysis. Relatório Técnico ISTC-CC-TR-12-101, Intel Science and Technology Center for Cloud Computing, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

[Reiss et al., 2011] Reiss, C., Wilkes, J. e Hellerstein, J. L. (2011). Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA. Revisado em 20/03/2012.

[Rochwerger et al., 2009] Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I. M., Montero, R., Wolfsthal, Y., Elmroth, E., Caceres, J., Ben-Yehuda, M., Emmerich, W. e Galan, F. (2009). The Reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4):4:1–4:11.

[Rodriguez et al., 2003] Rodriguez, A. D., Tchernykh, A. e Ecker, K. H. (2003). Algorithms for dynamic scheduling of unit execution time tasks. *European Journal of Operational Research*, 146(2):403–416.

[Schwarzkopf et al., 2013] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M. e Wilkes, J. (2013). Omega: Flexible, scalable schedulers for large compute clusters. Em *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, páginas 351–364, New York, NY, USA. ACM.

[Schwiegelshohn et al., 1999] Schwiegelshohn, U., Ludwig, W., Wolf, J. L., Turek, J. e Yu, P. S. (1999). Smart bounds for weighted response time scheduling. *SIAM J. Comput.*, 28(1):237–253.

[Schwiegelshohn et al., 2008] Schwiegelshohn, U., Tchernykh, A. e Yahyapour, R. (2008). Online scheduling in grids. Em *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, páginas 1–10.

[Sefraoui et al., 2015] Sefraoui, O., Aissaoui, M. e Eleuldj, M. (2015). Management platform for cloud computing. Em *Cloud Technologies and Applications (CloudTech)*, páginas 1–5.

[Sharma e Chawla, 2013] Sharma, S. e Chawla, M. (2013). A technical review for efficient virtual machine migration. Em *Cloud Ubiquitous Computing Emerging Technologies (CUBE)*, páginas 20–25.

[Sheng et al., 2013] Sheng, D., Kondo, D. e Cappello, F. (2013). Characterizing cloud applications on a Google data center. Em *2nd International Conference Parallel Processing (ICPP)*, páginas 468–473.

[Sheng et al., 2012] Sheng, D., Kondo, D. e Cirne, W. (2012). Characterization and comparison of cloud versus grid workloads. Em *IEEE International Conference on Cluster Computing (CLUSTER)*, páginas 230–238.

[Silberschatz et al., 2015] Silberschatz, A., Gagne, G. e Galvin, P. B. (2015). *Fundamentos de Sistemas Operacionais*. LTC.

[Simarro et al., 2011] Simarro, J. L. L., Moreno-Vozmediano, R., Montero, R. S. e Llorente, I. M. (2011). Dynamic placement of virtual machines for cost optimization in multi-cloud environments. Em *International Conference on High Performance Computing and Simulation (HPCS)*, páginas 1–7.

[Singh e Singh, 2013] Singh, L. e Singh, S. (2013). Article: A survey of workflow scheduling algorithms and research issues. *International Journal of Computer Applications*, 74(15):21–28.

[Sotiriadis et al., 2012] Sotiriadis, S., Bessis, N., Xhafa, F. e Antonopoulos, N. (2012). From meta-computing to interoperable infrastructures: A review of meta-schedulers for HPC, grid and cloud. Em *Advanced Information Networking and Applications (AINA)*, páginas 874–883.

[Strunk, 2012] Strunk, A. (2012). Costs of virtual machine live migration: A survey. Em *IEEE Eighth World Congress on Services*, páginas 323–329.

[Suen et al., 2011] Suen, C. H., Kirchberg, M. e Lee, B. S. (2011). Efficient migration of virtual machines between public and private cloud. Em *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, páginas 549–553.

[Tammaro et al., 2011] Tammaro, D., Doumith, E. A., Zahr, S. A., Smets, J. P. e Gagnaire, M. (2011). Dynamic resource allocation in cloud environment under time-variant job requests. Em *IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom)*, páginas 592–598.

[Tchernykh, 2014] Tchernykh, A. (2014). Introducción a calendarización en sistemas paralelas, grids y nubes. Centro de Investigación Científica y de Educación Superior de Ensenada.

[Tchernykh et al., 2010] Tchernykh, A., Schwiegelshohn, U., Yahyapour, R. e Kuzjurin, N. (2010). On-line hierarchical job scheduling on grids with admissible allocation. *Journal of Scheduling*, 13(5):545–552.

[Tchernykh et al., 2009] Tchernykh, A., Trystram, D., Brizuela, C. e Scherson, I. (2009). Idle regulation in non-clairvoyant scheduling of parallel jobs. *Discrete Applied Mathematics*, 157(2):364 – 376.

[Tchernykh et al., 2001] Tchernykh, A., Trystram, D. e Rapin, C. (2001). Adaptive (a,b,c)-scheme strategy for on-line scheduling. Em *New trends in scheduling for parallel and distributed systems*. Marseille - Lumini - CIRM.

[Tomas e Tordsson, 2013] Tomas, L. e Tordsson, J. (2013). Improving cloud infrastructure utilization through overbooking. Em *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, páginas 5:1–5:10, New York, NY, USA. ACM.

[Tran et al., 2015] Tran, T., Zhang, P., H. Li, D. D. e Beck, J. (2015). Resource-aware scheduling for data centers with heterogeneous servers. *MISTA*.

[Trystram et al., 2015] Trystram, D., Bleuse, R., Dutot, P., Georgiou, Y., Glesser, D., Jeannot, E., Kraemer, A., Lucarelli, G., Mendonca, F., Mercier, G., Mounie, G., Poquet, M., Srivastav, A. e Wagner, F. (2015). MOEBUS: Scheduling in large scale computing platforms. Relatório técnico, ANR - L'Agence Nationale de la Recherche, France.

[Vavilapalli et al., 2013] Vavilapalli, V. K., Murthy, A. C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., Malley, O., Radia, S., Reed, B. e Baldeschwieler, E. (2013). Apache hadoop YARN: Yet another resource negotiator. Em *Proceedings of the 4th Annual Symposium on Cloud Computing*, SOCC '13, páginas 5:1–5:16, New York, NY, USA. ACM.

[Wang e Pazat, 2012] Wang, C. e Pazat, J. L. (2012). A two-phase online prediction approach for accurate and timely adaptation decision. Em *IEEE Ninth International Conference on Services Computing*, páginas 218–225.

[Wang et al., 2014] Wang, W., Li, B. e Liang, B. (2014). Dominant resource fairness in cloud computing systems with heterogeneous servers. Em *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, páginas 583–591.

[Wieder et al., 2011] Wieder, P., Butler, J. M., Theilmann, W. e Yahyapour, R. (2011). *Service Level Agreements for Cloud Computing*. Springer Publishing Company, Incorporated.

[Wu e Guo, 2015] Wu, C. e Guo, J. (2015). *Software Monitoring in Data Centers*, páginas 1209–1253. Springer New York, New York, NY.

[Wu et al., 2011] Wu, L., Garg, S. K. e Buyya, R. (2011). SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments. Em *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, páginas 195–204, Washington, DC, USA. IEEE Computer Society.

[Xiang et al., 2013] Xiang, Y., Balasubramanian, B., Wang, M., Lan, T., Sen, S. e Chiang, M. (2013). Self-adaptive, deadline-aware resource control in cloud computing. Em *IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops (SASOW)*, páginas 41–46.

[Yarmolenko e Sakellariou, 2006] Yarmolenko, V. e Sakellariou, R. (2006). An evaluation of heuristics for SLA based parallel job scheduling. Em *20th International Parallel and Distributed Processing Symposium (IPDPS)*, página 8 pp.

[Ye et al., 2009] Ye, D., Han, X. e Zhang, G. (2009). *On-Line Multiple-Strip Packing*, páginas 155–165. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Yeo e Buyya, 2005] Yeo, C. S. e Buyya, R. (2005). Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. Em *2005 IEEE International Conference on Cluster Computing*, páginas 1–10.

[Zhang et al., 2014] Zhang, Z., Li, C., Tao, Y., Yang, R., Tang, H. e Xu, J. (2014). Fuxi: A fault-tolerant resource management and job scheduling system at Internet scale. *Proc. VLDB Endow.*, 7(13):1393–1404.

[Zhuk et al., 2004] Zhuk, S., Chernykh, A., Avestiyan, A., Gaissaryan, S., Kuzjurin, N., Pospelov, A. e Grushin, D. (2004). Comparison of scheduling heuristics for grid resource broker. Em *Proceedings of the Fifth Mexican International Conference in Computer Science*, páginas 388–392.

Apêndice A

Exemplos de documentos de Acordo de Nível de Serviço

A.1 Exemplo de uma especificação SLA generalizada

O resumo de um SLA apresentado a seguir tem como base o trabalho de [Freitas et al., 2011]:

- *Parties: service customer, service provider;*
- *Terms:*
  - *description: the service funcional requeriments;*
  - *duration: seconds;*
  - *customer's obligations: should not exceed the maximum number of requests per duration;*
  - *provider's obligations: should provide the agreed service according to its QoS;*
  - *SLA types: platinum, gold, silver;*
  - *priority: low, normal, high;*
  - *price model: pay-per-use (usage dependent);*
  - *price: price per time for SLA type + price per priority choosen.*
- *QoS: throughput MB/Sec.*

A.2 Exemplo de uma especificação SLA para o contexto de computação em nuvem

O resumo de um SLA apresentado a seguir tem como base o trabalho de [Wu et al., 2011]:

- *request type: first time rent or upgrade service;*
- *product type: the software product offered: standard, professional, or enterprise (this one includes all the features of professional plus report functions);*

- *account type: it constrains the maximum number of accounts that a customer can create: group, team or department;*
- *contract length: minimum is one month;*
- *number of accounts: the actual number of accounts that a customer wants to create;*
- *number of recored: The maximum number of records a customer is able to create for each account during the transaction and this will impact the data transfer time during the service upgrade. The value of this parameter is predefined in the SLA;*
- *response time: the value of each type of response time is different and predefined in SLA. The types are: response time for first time renting of the service, response time for adding new accounts, response time for upgrading the product.*

A.3 Exemplo de uma especificação adaptada do SLA @SOI para um contexto HPC

O resumo de um SLA apresentado a seguir tem como base o trabalho de [Koller, 2010].

- *DocType: Template,Tender, Offer, Request, Counter-Offer or Preventive.*
- *Context: Third party: Role (or evaluation), Owner; Document base: Name, DocType; Validity: Start, End, Condition Time, etc; Service provider: Name, Phone, etc; Customer: Name, Phone, etc; Other.*
- *Terms:*
  - *Service Terms: service description, and service reference.*
  - *Guarantee Terms: Obligated (provider, customer, or thrird party), Service Name (Resource, Application, Service Availabiity, Data Treatment, and other), Negotiable or non-negotiable.*
  - *Business Terms.*

Usualmente, os parâmetros do elemento *Guarantee Terms* são usados como SLOs. Destacam-se a seguir alguns desses SLOs:

- *Resource terms: Candidate host; FileSystem; Exclusive execution; Operating System; CPU architecture; Individual CPU speed; Individual CPU time; Individual CPU count; Individual network bandwidth; Individual physical memory; Individual virtual memory; Individual disk space; Total CPU time; Total CPU count; Total network bandwidth; Total physical memory; Total virtual memory; Total disk space; Total resource count; Other.*
- *Availability terms: Availability rate (it defines the minimal percentage of availability for the agreed service): Minimum rate, Maximum rate, Description, Period; Response time: Minimum response time, Maximum response time, Unit; Maximum run time; Start time; End time; Invocation limit (maximum number of invocation by Customer); Duration; Other.*

- *Data Treatment: Data access: Unlimited, Special Group, Exclusive; Level of confidentiality; Data backup (backup for intermediate and result data): Full backup, Diff backup, File name, Time interval for start backup; Data staging: File system, Creation flag (append or overwrite an existing file), Delete on termination, Source (where the data can be get from), Target (name to data movement); Data storage: shared or exclusive storage; Other.*

Por fim, *Business Terms* contêm especificações das penalidades e regras de valores, como seguem:

- *Name;*
- *Guarantee name refered;*
- *Importance;*
- *Penalty: service term reference (it represents an alternative way of achieving the associated SLO);*
- *Utility: specifies the utility gained by achieving of SLO;*
- *Reward;*
- *Preference: this element specifies a list of fine-granularity business values for different alternatives;*
- *Custom business value;*
- *Price: Fixed, Variable.*

Uma estrutura de compensação é definida para *Penalty* e *Reward* considerando o intervalo de tempo avaliado, valor de cada unidade e uma expressão lógica.

Apêndice B

Detalhamento do simulador e experimentação científica

O simulador desenvolvido para representar uma área de convergência nuvem-HPC chama-se CARS (*Convergence Area Scheduler* - Escalonador de Área de Convergência). Esse programa é uma ramificação do OAR para escalonamento em lote. Detalham-se neste apêndice a arquitetura do simulador, os procedimentos para instalação e experimentação científica. Sugere-se o trabalho [Margery et al., 2014] para entendimento do OAR.

B.1 Arquitetura do sistema

As plataformas de computação simuladas utilizadas pelo CArS são criadas por um simulador base conhecido como Simgrid (*Grid Simulator* - Simulador de grids). Para prover a comunicação entre um escalonador e o Simgrid, pesquisadores do MOAIS (*prograMming and scheduling design fOr Applications in Interactive Simulation* - Projeto de Escalonamento e Programação para Aplicações em Simulação Interativa) desenvolveram o BatSim (*Batch Scheduler Simulator* - Escalonador de Aplicações em Lote). O CArS é um desses escalonadores que são incorporados por meio do BatSim. Apresenta-se a seguir uma breve caracterização desses programas.

B.1.1 Caracterização do Simgrid (*Grid Simulator*)

O Simgrid foi desenvolvido em 1999 para simular escalonamentos *offline* de gráfos [Casanova et al., 2014]. A classe de experimentação suportada por este simulador é *in silico*, caracterizada por utilizar dados reais e não apenas dados de simulações matemáticas. Os modelos de simulação implementados dentro do Simgrid são aprovados pelo INRIA (*Institut National de Recherche en Informatique et en Automatique* - Instituto Nacional de Pesquisa em Computação e Automação), onde o MOAIS atua.

Atualmente, o Simgrid tem suporte para simulação de grids, nuvens, HPC, P2P (*Peer-to-Peer* - Ponto-a-Ponto) e MPI (*Message Passing Interface* - Interface de Passagem de Mensagem). As simulações são escritas em Java, C, Lua ou Ruby. A maior parte dos seus programas é distribuída com a licença L-GPL 2.1. (*Lesser General Public License* - Licença Pública Geral Menor).

A seguir apresenta-se a Figura B.1, a fim de representar em alto nível a integração de um novo escalonador na arquitetura do Simgrid. Os passos destacados como 1º, 2º e 3º representam o ponto de vista do desenvolvedor de uma nova estratégia de escalonamento. No

1º passo, o desenvolvedor define a configuração da plataforma, podendo descrever detalhes de cada computador, como sua memória, processadores, e outros recursos. No 2º passo, que não é obrigatório, o desenvolvedor pode definir detalhes das aplicações, como o tempo de processamento, memória requerida, quais processadores são mais indicados para determinados tipos de aplicação, detalhar o comportamento de cada processador para habilitar preempção, entre outras. Por fim, no 3º passo, definem-se quais e quando as aplicações são lançadas para o escalonador, podendo alterar detalhes das aplicações definidas no 2º passo.

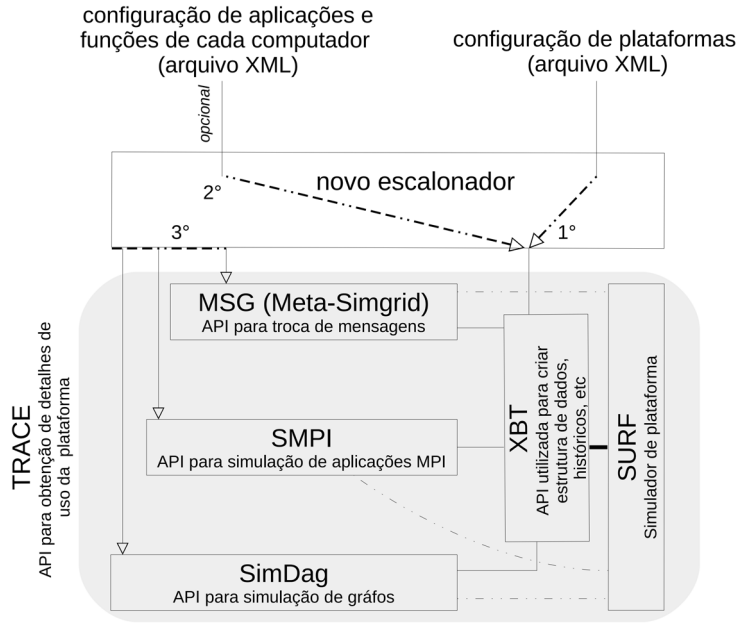


Figura B.1: Integração de um novo escalonador na arquitetura do Simgrid.

B.1.2 Caracterização do BatSim (*Batch Scheduler Simulator*)

O BatSim (*Batch Scheduler Simulator* - Escalonador de Aplicações em Lote) é um simulador de RJMS (*Resource and Job Management System* - Sistema Gerenciador de Tarefas e Recursos) que habilita incorporação de um escalonador externo [Dutot et al., 2016]. O seu desenvolvimento foi iniciado no INRIA em 2015 e atualmente está codificado em C++.

O BatSim é orientado a eventos, sendo que existem somente dois eventos tratados no sistema: um para indicar que uma nova aplicação foi recebida na fila de espera e outro para indicar o término dela. Um escalonador incorporado considera esses eventos para tomadas de decisão. De um lado, o BatSim integra-se com o Simgrid para instanciar plataformas de computação; e de outro lado, ele abre um *socket* para comunicar-se com um escalonador incorporado, que passa a gerenciar as aplicações. O CArS é um exemplo de escalonador incorporável. Essa integração tem como objetivo tornar mais flexível, em termos de linguagem de programação e definição de aplicações, o desenvolvimento e a experimentação de novos escalonadores.

A definição de uma plataforma para o BatSim é realizada utilizando o mesmo padrão XML do Simgrid, tal como o exemplo ilustrado na Figura B.2. O protocolo de comunicação entre o BatSim e um escalonador incorporado é apresentado na sequência.

O protocolo de comunicação entre o BatSim e o escalonador incorporado define o formato de uma mensagem como *"ILT | DATA: TIPO_OPERAÇÃO : OPERAÇÃO"*, onde:

- *ILT* é o intervalo na linha de tempo, com especificação de início e fim;

```
<?xml version='1.0'?>
 <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
 <platform version="3">
 <AS  id="AS0"  routing="Full">
   <host id="master_host" power="98.095Mf"/>
   <host id="Jupiter" power="76.296Mf"/>
      ...
   <link id="6" bandwidth="41.279125MBps" latency="59.904us"/>
      ...
   <route src="master_host" dst="master_host"><link_ctn id="loopback"/></route>
   <route src="Jupiter" dst="Jupiter"><link_ctn id="loopback"/></route>
   <route src="master_host" dst="Jupiter">
    <link_ctn id="6"/>
   </route>
      ...
</AS>
</platform>
```

Figura B.2: Exemplo de arquivo XML para configuração de uma plataforma no BatSim e no Simgrid.

- *DATA* é o instante exato para execução da operação;
- *TIPO_OPERAÇÃO* é um parâmetro que depende de qual programa está enviando a mensagem. Caso seja o escalonador incorporado, têm-se os parâmetros: *J* (alocar aplicação) ou *R* (rejeitar aplicação). Caso seja uma mensagem do BatSim, tem-se: *NOP* (nenhuma operação sendo executada no momento), *S* (submissão de aplicação) ou *C* (aplicação completa/finalizada), sendo *S* e *C* os únicos eventos tratados no sistema.

Para exemplificar uma mensagem, segue um caso onde o escalonador incorporado solicita a execução da aplicação de código 1 nos processadores $\{0, 1, 2\}$ e a aplicação de código 2 no processador $\{3\}$. Assim, mais de uma aplicação pode ser informada no intervalo da mensagem entre 0 e 25.836709. As aplicações de código 1 e 2 serão executadas no instante 25.836709, como representado nesta mensagem: *"0 : 25.836709 | 25.836709 : J : 1=0,1,2 ; 2=3"*.

B.1.3 Caraterização do CArS (*Convergence Area Scheduler*)

O CArS é um escalonador incorporado no BatSim, construído em linguagem Python. O desenvolvimento do CArS foi iniciado em 2016 como projeto desta tese de doutorado. Esse escalonador é uma ramificação do OAR com implementação de gerenciamento de tempo de resposta e escalonamento de aplicações em área de convergência nuvem-HPC [Kraemer et al., 2016] e [Kraemer et al., 2017]. Para uso desse sistema são necessários os arcabouços BatSim e Simgrid. A definição de plataforma é feita da mesma maneira que o arquivo XML apresentado na Figura B.2. A carga de trabalho é definida em formato JSON, tal como apresentado na Figura B.3, o que distingue-se do padrão de carga de trabalho definido em XML para o Simgrid.

B.2 Máquina virtual para replicação dos experimentos

A execução de um experimentos científicos envolvendo uma carga de trabalho que representa um cenário real de aplicações em grid pode levar horas de processamento. Isto se deve pelo elevado número de aplicações e operações de escalonamento que precisam ser tratados pelo simulador. Uma solução que pode agilizar a execução de vários experimentos, procurando

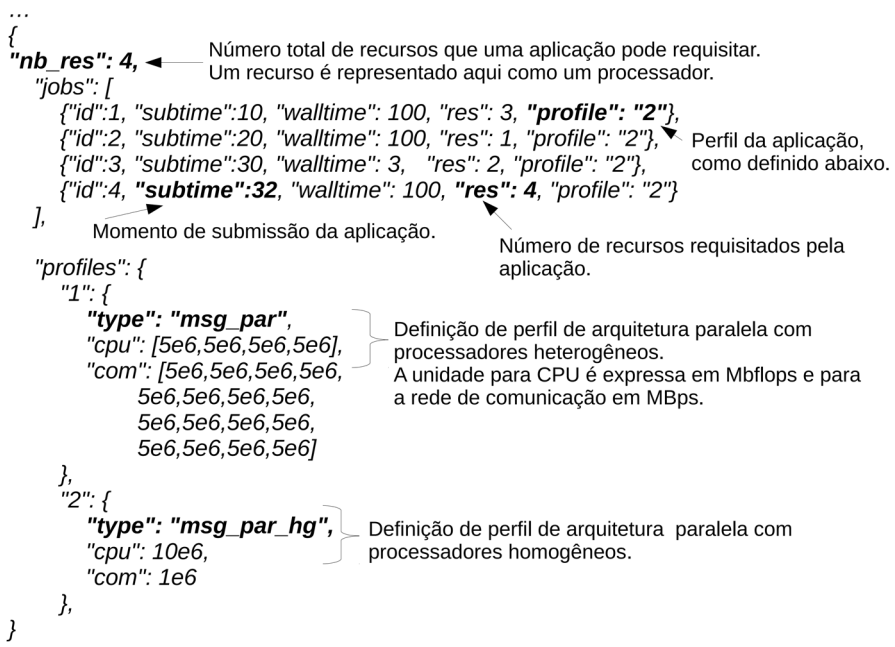


Figura B.3: Exemplo de arquivo JSON para configuração de aplicações.

executá-los ao mesmo tempo, consiste em considerar o processamento paralelo de diferentes cenários. Entretanto, a experimentação paralela no mesmo computador e sem máquina virtual é dificultosa envolvendo o BatSim, pois este cria apenas um *socket* no sistema operacional, permitindo somente um escalonador incorporado e somente uma carga de trabalho. Considerando estas barreiras, a instalação do CArS em uma máquina virtual é interessante para experimentação científica, pois permite instanciar vários experimentos isoladamente, em um mesmo computador e em paralelo. A seguir são apresentados os passos para preparação de uma máquina virtual.

Utiliza-se aqui a sigla VM (*Virtual Machine*) para representar máquina virtual. Como primeiro passo de instalação, recomenda-se executar os comandos *apt update* e *apt upgrade*. Isto atualizará a lista de aplicações nos repositórios e as bibliotecas instaladas no sistema operacional. Os passos a seguir foram executados em um instalação Linux Ubuntu 16.04 e representam a criação de uma VM KVM:

- Instalar suporte a virtualização no computador:
  - *sudo apt install virt-manager cpu-checker qemu-kvm*
  - *sudo apt install openssh-server*
- Criar uma nova VM:
  - abrir o *virt-manager*
  - solicitar nova VM
  - selecionar suporte de instalação local ISO (Linux Ubuntu Server 16.04)
  - escolher arquivo de imagem iso
  - definir quantidade de memória 4GB, 2 CPUs
  - definir armazenamento de 15GB para o disco virtual
  - definir apenas a partição raiz e sem *swap*

Por fim, são necessários os seguintes passos para criar réplicas dessa VM e executar instâncias em paralelo:

- Exportar a VM:
  - desligar a VM primeiro
  - *virsh dumpxml vm_name > vm_name.xml*
  - *sudo cp /var/lib/libvirt/images/vm_name.qcow2 .*
  - *gzip vm_name.qcow2*
  - transferir *vm_name.** para o destino
- Importar a VM principal (passo não obrigatório, sugerido apenas como forma de verificação se o *hardware* virtual é reconhecido no computador):
  - *gunzip vm_name.qcow2.gz*
  - *sudo cp vm_name.qcow2 /var/lib/libvirt/images/*
  - *virsh define vm_name.xml*
  - *rm vm_name**
- Criar réplicas:
  - *sudo cp vm_name.qcow2 vm_name_number.qcow2&*
  - *jobs* (este comando permite verificar se o processo de criação das réplicas foi finalizado)
  - abrir o *virt-manager*
  - criar novas VMs indicando seus respectivos arquivos *vm_name_number.qcow2*

B.3 Passos para instalação do CArS na máquina virtual

O primeiro passo para instalação do CArS consiste em instalar todas as suas dependências. Dividem-se as dependências em duas categorias: (tipo 1) ferramentas de compilação, bibliotecas de linguagens de programação e banco de dados; e (tipo 2) os programas Simgrid e BatSim. Apresentam-se as versões das dependências na Tabela B.1.

Seguem abaixo os comandos para instalação das dependências de tipo 1:

- *apt update*
- *apt install make cmake gcc-5 g++-5 python git*
- *apt install libboost-all-dev doxygen libgmpv4-dev libzmq5-dev libhiredis-dev libev-dev libgmpxxv4-4 libssl-dev python-pip python-sortedcontainers*
- *pip install redis sqlalchemy setuptools*

Os comandos que seguem ainda tratam de dependências do tipo 1, sendo que o código-fonte deles pode ser obtido via programa git (recomenda-se executar cada comando git no diretório */opt*):

Tabela B.1: Versão de cada programa necessário ao simulador CArS

Programa	**Versão**
make	4.1
cmake	3.5.1
gcc-5	5.3.1
gcc++-5	5.3.1
python	2.7.12
libboost-all-dev	1.58
doxygen	1.8.11
libgmpv4-dev	2:6.1
libzmq5-dev	4.1.4
libhiredis-dev	0.13.3
libev-dev	1:4.22
libgmpxxv4-4	2:6.1
libssl-dev	1.0.2g
python-sortedcontainers	2.7.11
redis	2.10.5
sqlalchemy	1.1.7
rapidjson	1.0.4
redox	0.3
oar-docker	1.4.0
python-oar-lib	0.5.0
simgrid	3.13
batsim	

- *git clone https://github.com/miloyip/rapidjson.git*
  - *cd rapidjson*
  - *mkdir build; cd build*
  - *cmake ..*
  - *make*
  - *make install*
- *git clone https://github.com/hmartiro/redox.git*

  - *cd redox*
  - *mkdir build; cd build*
  - *cmake .. -DCMAKE_INSTALL_PREFIX=../install*
  - *make -j $(nproc)*
  - *make install*
  - *cd ../install*
  - *mv -f lib64 lib*
  - *sudo cp -r ./* /usr/*

- *pip install git+https://github.com/oar-team/oar-docker.git*

- *git clone https://github.com/oar-team/oar3.git -b lib python-oar-lib*
  - *cd python-oar-lib*
  - *python setup.py install*

Para a instalação das dependências do tipo 2 (Simgrid e Batsim), copia-se primeiramente o código-fonte via git, como seguem (ambos para o diretório */opt*):

- *git clone git://scm.gforge.inria.fr/simgrid/simgrid.git*

- *git clone https://scm.gforge.inria.fr/anonscm/git/batsim/batsim.git*

Para cada um desses programas, recomenda-se criar um diretório chamado *build* (exemplo: *cd /opt/simgrid; mkdir build; cd build*). Em seguida, dentro do diretório *build*, realizam-se a checagem da configuração, compilação e instalação por meio dos seguintes comandos:

- *cmake .. -DCMAKE_CXX_COMPILER=/usr/bin/g++-5 -DCMAKE_EXE_LINKER_FLAGS=-dynamic"*

- *make*

- *make install*

O CarS utiliza a API MSG da arquitetura do simulador Simgrid. Entretanto, podem ocorrer problemas de dependência envolvendo a API SMPI, que é dedicada para simulação de aplicações MPI. A fim de evitar essas dependências que não fazem parte do CarS, para as compilações que seguem desabilitaram-se as chamadas de função SMPI, em */opt/batsim/src*. Finalmente, o CArS é obtido pelo comando *git*:

- *git clone https://scm.gforge.inria.fr/anonscm/git/cars/cars.git*

e sua instalação é feita pelo comando:

- *python setup.py install*

B.4 Passos para execução de um experimento

A experimentação científica no CArS envolve a utilização de dois arquivos de configuração: um arquivo XML contendo detalhes da plataforma, tal como apresentado na Figura B.2; e um arquivo JSON contendo detalhes das aplicações, tal como apresentado na Figura B.3. Neste capítulo apresentam-se os procedimentos para criação desses arquivos.

B.4.1 Configuração da plataforma

Utiliza-se o *script platform_generator.py* para criar um arquivo XML de partição onde todos os processadores estão conectados. Esse *script* está localizado no diretório */opt/cars/oar/kao* e sua sintaxe é como segue:

- *./platform_generator.py -i number_hosts -o outputfile*

Após a definição da plataforma é necessário validá-la no Simgrid por meio do seguinte comando:

- */opt/simgrid/tools/simgrid_update_xml.pl outputfile*

B.4.2 Validação do arquivo de carga de trabalho HPC e inserção de aplicações de nuvem

Arquivos de carga de trabalho contendo dados de aplicações reais em grids podem ser obtidos em *http://www.cs.huji.ac.il/labs/parallel/workload/*. O formato padrão para especificação de cargas de trabalho em grids é o SWF (*Standard Workload Format* - Formato Padrão de Carga de Trabalho). Entretanto, o BatSim e o CArS adotam uma especificação em formato JSON. Uma razão para isto é que no formato JSON foram acrescentados mais campos sobre aplicações, tal como o limite de tempo de resposta e o perfil da aplicação (i.e. paralela, homogênea ou heterogênea). Assim, um arquivo SWF deve ser convertido para JSON para ser utilizado no CArS. Para realizar essa conversão, utiliza-se o *script swf2json.py* localizado no diretório */opt/cars/oar/kao/workload*.

Após a criação do arquivo JSON de carga de trabalho HPC, o *script insert_cloud_job.py* pode ser utilizado para inserção de aplicações de nuvem. Para facilitar a identificação das aplicações inseridas, esse script cria um arquivo denominado *info* contendo o número total de aplicações inseridas e a identificação de cada uma delas. Esse script está localizado no diretório */opt/cars/oar/kao/workload*. Segue a sua sintaxe:

- insert_cloud_job.py -i inputfile -o outputline
  -t default_processing_time
  -r number_of_processors_required
  -f slack_factor
  -v interval_between_cloud_jobs
  -k profile
  -m max_number_of_resources_for_a_job
  -p partition

Destaca-se que o parâmetro *-m* encurta aplicações do *inputfile* caso excedam o tamanho informado. Para simulações em geral, este parâmetro contém o mesmo número de processadores do domínio do escalonador, ou seja, o tamanho definido por meio de *./platform_generator.py*.

B.4.3 Execução de uma simulação

A execução de uma simulação consiste em iniciar o BatSim e em seguida iniciar o CArS, como segue. Em um terminal de linha de comando, executa-se:

- *batsim platform_file workload_file*

Complementarmente, em outro terminal de linha de comando, executa-se:

- *cd /opt/cars/oar/kao*
- *./bataar.py workload_file*

O mesmo arquivo contendo a carga de trabalho deve ser utilizado por ambos os programas. O BatSim carrega esse arquivo em memória RAM e verifica se todos os parâmetros foram definidos corretamente. Em seguida, o CArS, por meio da execução do *script ./bataar.py*, utiliza o protocolo de comunicação para indicar qual aplicação será processada, não sendo necessário o envio de todos os detalhes da aplicação, pois o BatSim já os possui. Finalmente, os resultados da simulação são armazenados em diversos arquivos, como seguem:

- *out_jobs.csv*: dados sobre a vida da aplicação no sistema;
- *out_jobs_violated.csv*: dados sobre o número de aplicações violadas e os detalhes de cada violação;
- *out_schedule.csv*: dados sobre o número de aplicações atendidas e a vida do escalonador;
- *out_violation_prediction_error.csv*: dados sobre todas as aplicações com erro de previsão, incluindo a previsão;
- *out_violation_prediction_error_g.csv*: dados sobre as aplicações violadas que não foram previstas;
- *out_violation_prediction_error_l.csv*: dados sobre as aplicações que foram previstas como violadas mas que foram executadas sem violação.