UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS BOUSQUET BANDINI

ESCALONAMENTO DISTRIBUÍDO DE APLICAÇÕES PARALELAS AUTÔNOMAS EM AMBIENTES COMPUTACIONAIS COMPARTILHADOS

NITERÓI

2018

UNIVERSIDADE FEDERAL FLUMINENSE

MATHEUS BOUSQUET BANDINI

ESCALONAMENTO DISTRIBUÍDO DE APLICAÇÕES PARALELAS AUTÔNOMAS EM AMBIENTES COMPUTACIONAIS COMPARTILHADOS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Sistemas de Computação.

Orientador:
EUGENE FRANCIS VINOD REBELLO

Co-orientador:
MARIA CRISTINA SILVA BOERES

NITERÓI

2018

Ficha catalográfica automática - SDC/BEE

B214e Bandini, Matheus Bousquet
Escalonamento Distribuído de Aplicações Paralelas Autônomas em Ambientes Computacionais Compartilhados / Matheus Bousquet Bandini ; Eugene Francis Vinod Rebello, orientador ; Maria Cristina Silva Boeres, coorientadora. Niterói, 2018.
111 f. : il.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.d.09290480750

1. Escalonamento de Aplicações Autonômicas. 2. Compartilhamento de Recursos Compartilhados. 3. Escalonamento Descentralizado. 4. Produção intelectual. I. Título II. Rebello,Eugene Francis Vinod, orientador. III. Boeres, Maria Cristina Silva, coorientadora. IV. Universidade Federal Fluminense. Escola de Engenharia.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

MATHEUS BOUSQUET BANDINI

ESCALONAMENTO DISTRIBUÍDO DE APLICAÇÕES PARALELAS AUTÔNOMAS EM AMBIENTES COMPUTACIONAIS COMPARTILHADOS

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Sistemas de Computação.

Aprovada em 9 de julho de 2018.

BANCA EXAMINADORA

V. Rebello

Prof. EUGENE FRANCIS VINOD REBELLO - Orientador, UFF

Prof[a]. MARIA CRISTINA SILVA BOERES - Co-orientadora, UFF

Prof. ANTÔNIO AUGUSTO DE ARAGÃO ROCHA, UFF

Prof[a]. SIMONE DE LIMA MARTINS, UFF

Prof. BRUNO RICHARD SCHULZE, LNCC

Prof[a]. CRISTIANA BARBOSA BENTES, UERJ

Niterói

2018

À minha esposa Renata e à minha filha Rebeca
por serem as razões da minha vida.

Agradecimentos

Agradeço a Deus por todas as graças que me foram concedidas e que me permitiram conquistar o que conquistei até aqui.

À memória da minha mãe Vera Marina por ter me ensinado, por meio dos seus próprios atos, a fazer com carinho e dedicação todas as coisas que eu vier a fazer. Também, à memória do meu pai Waldyr por todos os valores transmitidos. Obrigado por terem moldado o meu caráter.

À minha amada esposa Renata pelo amor incondicional, pela compreensão e paciência nos momentos mais difíceis e por ter ficado ao meu lado ao longo do caminho. Obrigado por fazer parte da minha vida.

À minha amada filha Rebeca por todo o amor e por me fazer querer ser o melhor exemplo em todos os aspectos da vida.

Aos meus orientadores, professor Vinod Rebello e professora Cristina Boeres, por todo o tempo dedicado a mim e ao meu trabalho durante todo o doutorado. Agradeço também pelos conselhos que me deram e pela paciência que tiveram comigo.

Ao professor e amigo Bruno Schulze por todo o apoio, pelas oportunidades concedidas e pelos conselhos durante os mais de 10 anos que nos conhecemos. Agradeço também, é claro, por aceitar o convite para participar da minha banca de defesa de doutorado.

Aos professores Antônio Augusto Rocha, Cristiana Bentes e Simone Martins, por aceitarem participar da minha banca de defesa de doutorado.

Ao amigo Victor Oliveira pela incansável ajuda na revisão do texto da tese e pelas discussões que contribuíram para a realização do trabalho.

Ao amigo Fabrício Oliveira por ter me ajudado a entender os diversos aspectos práticos do EasyGrid.

Aos amigos Douglas Oliveira e Henrique Klôh, por terem partilhado comigo mais de 5 anos de viagens entre Petrópolis e Niterói e por terem ajudado a transformar até mesmo longos congestionamentos em momentos de descontração.

Aos amigos do laboratório ComCiDis/LNCC Jonathan Barbosa, Daniel Yokoyama, André Yokoyama, Jonatan Gall, Jonathas Almeida, Allan Santos, Mariza Ferro, Vinícius Klôh e Gabrieli Silva. Fiz questão de agradecer cada um nominalmente para enfatizar a importância do apoio e da amizade de todos.

Ao laboratório ComCidis (Computação Científica Distribuída) e ao LNCC (Laboratório Nacional de Computação Científica) pela disponibilização dos recursos computacionais utilizados para a execução dos experimentos apresentados nesta tese.

A todos os professores e alunos do Instituto de Computação da Universidade Federal Fluminense com os quais convivi durante o doutorado.

À CAPES, pela concessão da bolsa de doutorado entre 2013 e 2017, que me permitiram realizar o doutoramento.

Resumo

Supercomputadores e *Data centers* cada vez maiores são utilizados para auxiliar pesquisadores a resolverem os complexos problemas científicos. Porém, para torná-los economicamente viáveis, eles devem ser compartilhados, transformando-os em sistema multi usuários. Já se foi o tempo em que os usuários detinham acesso exclusivo aos recursos, pois em sistemas relativamente pequenos, a taxa de utilização desses recursos era baixa. Portanto, sistemas de gerenciamento de recursos agora têm a responsabilidade de prover recursos dedicados para cada usuário do ambiente, enquanto promovem também o compartilhamento dos recursos do sistema compartilhado de maneira eficiente.

A maioria dos sistemas escalonadores de aplicações empregam um esquema de fila que fazem as aplicações aguardarem até que os recursos que elas necessitam para executar estejam disponíveis. O conjunto de recursos utilizados é definido pelo usuário e é tipicamente a quantidade necessária para minimizar o tempo de execução da aplicação. Isto, no entanto, tende a causar um efeito adverso sobre a utilização do sistema, pois alguns recursos podem permanecer ociosos, e sobre o tempo de *turnaround* da aplicação, uma vez que essa abordagem pode aumentar o tempo que a aplicação precisa aguardar na fila até que os recursos solicitados fiquem disponíveis. Em contraste a essa abordagem tradicional, na qual os recursos são alocados exclusivamente a uma aplicação, esta tese propõe e avalia a adoção de uma estratégia descentralizada de escalonamento de aplicações que contempla o compartilhamento de recursos entre as aplicações. Ao invés de um ou mais gerenciadores de recursos, cada aplicação é responsável pela sua própria alocação de recursos, o que permite uma utilização mais dinâmica e aprimorada do sistema. A avaliação da estratégia proposta foi realizada por meio da execução simultânea de aplicações que compartilharam recursos de processamento. Os experimentos mostraram que a adoção da estratégia proposta permite aumentar a vazão dos recursos de processamento, o que beneficia os provedores de serviços. Ao mesmo tempo, foi possível também reduzir o tempo necessário para executar o conjunto de aplicações, resultado relevante para os provedores de serviço como para os usuários. Por ser capaz de ajustar a grau de compartilhamento entre aplicações, acreditamos que esta nova abordagem seja mais flexível no que se refere a possíveis escalonamentos de aplicações. Por ser também uma abordagem distribuída, pode tornar o problema de escalonamento de aplicações em maiores escalas mais tratável. Estas aplicações autonômicas exploram o uso dos recursos para reduzir o tempo necessário para executar um *workload* (ou conjunto de aplicações). Isso é alcançado por intermédio da capacidade da aplicação paralela de medir a sua utilização em cada um dos recursos do ambiente que ela utiliza e por meio da adoção de um esquema inovador usado para simular as políticas de fila das abordagens tradicionais.

Palavras-chave: Escalonamento de aplicações autonômicas, Compartilhamento de recursos computacionais, Escalonamento decentralizado.

Abstract

Ever larger data centers and supercomputers are being deployed to help researchers to solve the complex scientific problems of tomorrow. However, to make them economical, they must be shared and have become multiuser and multi-tenant systems. The days when a user had exclusive access are gone, since even on relative small systems, resource utilization was often very low. Therefore, resource management software now has the responsibility of providing each user with a dedicated environment while sharing the available resources of the shared system in an efficient manner.

Most job scheduling systems employ a queuing scheme to hold jobs while they wait for the resources they require for execution to become available. The quantity of resources is defined by the user and is typically the amount that minimizes the execution time of the job. This, however, tends to have an adverse affect on system utilization, as some resources may remain idle, and on job turnaround times, since this approach may increase the time that the job must wait in the queue for the solicited resources to become available. In contrast to this traditional approach, where resources are allocated exclusively to a job, we propose the adoption of a decentralized job scheduling strategy that embraces resource sharing between jobs and evaluate it. Instead of one or more separate resource brokers, each job is itself responsible for its own resource allocation, which allows for a more dynamic and improved system utilization. The evaluation of the proposed strategy was carried out through the simultaneous execution of applications that shared processing resources. Our experiments showed that the adoption of the proposed strategy allows to increase the throughput of processing resources, which benefits the service providers. At the same time, it was also possible to reduce the time required to run the application workload, Which provides a relevant result to both service providers and application users. Since it enables the adjustment of the level of resource sharing between applications, we believe that the approach may be more flexible in terms of job scheduling possibilities. Also, as it is a distributed approach, it might make the problem more tractable for job scheduling on a larger scale. These autonomic self-scheduling jobs exploit the use of idle resources to reduce the completion time of a workload (or set of jobs). This is achieved by the parallel job's ability to infer it's utilization of each of the system's resources and the adoption of a novel scheme to mimic the queuing policy of traditional approaches.

Keywords: Autonomic application scheduling, Resource sharing, Decentralized scheduling.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AMS	:	*Application Management System*;
CMPD	:	Computação Massivamente Paralela e Distribuída;
CPU	:	*Central Processing Unit*;
DoE	:	Departamento de Energia dos Estados Unidos da América;
FCFS	:	*First Come First Served*;
FJSP	:	*Flexible Job-shop Scheduling*;
GDS	:	*Global Dynamic Scheduler*;
GM	:	*Global Manager*;
HDS	:	*Host Dynamic Scheduler*;
HM	:	*Host Manager*;
HPC	:	*High Performance Computing*;
HPC4E	:	*High Performance Computing for Energy*;
MCTI	:	Ministério de Ciência, Tecnologia e Inovação;
MPI	:	*Message Passing Interface*;
NPB	:	*NAS Parallel Benchmarks*;
PBS	:	*Portable Batch System*;
PRA	:	Percentual de Recursos Consumidos por uma Aplicação;
PTR	:	Percentual Total de Consumo dos Recursos de Processamento;
SDS	:	*Site Dynamic Scheduler*;
SLURM	:	*Simple Linux Utility for Resource Management*;
SM	:	*Site Manager*;
SO	:	Sistema Operacional;
TA	:	Tempo Alvo;
TE	:	Tempo de Execução;
TET	:	Tempo Estimado de Término;
TMT	:	Tempo Médio de *Turnaround*;
TR	:	Tempo Restante;

Sumário

Capítulo 1

Introdução

Durante décadas, os avanços tecnológicos e científicos permitiram que diversas áreas do conhecimento como Ciências Naturais, Engenharias, Medicina, entre outras, produzissem resultados que contribuíram para os avanços da sociedade moderna. No entanto, problemas novos e cada vez mais complexos surgem à medida em que questionamentos passados são respondidos. Ao mesmo tempo, torna-se cada necessário que tais problemas sejam resolvidos de maneira mais rápida, precisa e eficiente. Tal necessidade resultou na união entre essas áreas e a Computação, cunhando-se assim o termo *e-Science*.

O termo *e-Science applications* (ou aplicações de e-Ciência) é empregado para referenciar as aplicações científicas computacionais de diversas áreas. Tais aplicações são responsáveis por resolver problemas complexos por meio da realização de uma quantidade elevada de cálculos numéricos e da análise de grandes volumes de dados [24]. Em razão dessas características, as aplicações de e-Ciência passaram a ser projetadas de maneira a explorar os ambientes computacionais com alta capacidade de processamento e de armazenamento, a fim de reduzir o tempo necessário para a obtenção de resultados e de aumentar a sua precisão. O termo *High Performance Computing* (HPC) é atribuído ao emprego de tais ambientes para resolver problemas computacionais complexos. Alguns exemplos de aplicações científicas que demandam alta capacidade de processamento e armazenamento são:

- As pesquisas que envolvem a investigação de fatores genéticos que causam câncer de intestino, realizadas entre o *Edinburgh Parallel Computing Centre* (EPCC) e o *Colon Cancer Genetics Group* (CCGG), ambos da Universidade de Edimburgo [41]. O processamento das informações cruzadas de mais de 500.000 marcadores genéticos de mais de 2.000 pessoas resultou em um algoritmo sequencial que, mesmo após ser

submetido a otimizações, leva em torno de 130 dias para executar. Contudo, a solução paralela executou em aproximadamente 6,5 horas em um supercomputador BlueGene/L com 128 processadores.

- O acelerador de partículas do CERN é capaz de colidir partículas aproximadamente 600 milhões de vezes por segundo. Isso faz com que cerca de 30 *petabytes* de dados sejam gerados por ano para serem analisados [80]. A aplicação da computação de alto desempenho ao processamento desses dados tornou possível, entre outras coisas, a confirmação do bóson de Higgs [1].

- A simulação de tráfego urbano em grandes cidades envolve a análise de comportamento de milhares de veículos em milhares de quilômetros de ruas e rodovias. Turek apresenta esse problema em [77] como uma oportunidade para explorar o uso da Computação de Alto Desempenho, uma vez que o tempo de execução das simulações em computadores sequenciais são elevados demais a ponto de não poderem ser utilizados. Além disso, o autor também discute a necessidade de aperfeiçoar a definição do problema de simulação e do uso da infraestrutura disponível. Com isso, o objetivo é aumentar tanto a escalabilidade do problema em função do maior volume de dados de entrada, como a qualidade da solução, por meio do aumento de recursos computacionais de processamento utilizados.

O uso de supercomputadores e de *data centers*, responsáveis por dar suporte às aplicações de e-Ciência, demanda o treinamento de pessoal qualificado e a reserva de espaço físico adequado. Ademais, existe também o alto custo financeiro envolvido na adquisição, na operação e na manutenção de equipamentos da infraestrutura física e no treinamento de profissionais. Em muitos casos, esse custo pode extrapolar os limites de investimento, impondo restrições à pesquisa a ser realizada. Um levantamento realizado por Wilson *et al.* [79] aponta que o valor de aquisição de apenas um servidor com 36 núcleos divididos em duas *Central Processing Unit*s (CPUs) e com 64 GB de memória é de US$ 14.375,00. Além disso, é considerado um custo mensal de US$ 596,16 para uma carga de utilização média de 85% e para manter a refrigeração a 50%. Os valores citados desconsideram gastos com a aquisição/locação do espaço físico e com treinamento de pessoal. Em uma escala ainda maior, o investimento realizado para a aquisição do supercomputador Santos Dumont [48], concretizada em 2016 pelo Ministério de Ciência, Tecnologia e Inovação (MCTI) do Brasil, totalizou R$ 60 milhões. A esse valor, acrescenta-se R$ 8 milhões anuais com custos energéticos e de equipamentos de apoio para manter o supercomputador em operação.

O uso convencional do escalonamento de aplicações (ou *jobs*) de e-Ciência em ambientes de HPC envolve a sua execução utilizando parte da infraestrutura dedicada a ela. Na abordagem tradicional de escalonamento, uma fração dos recursos da infraestrutura é utilizada de modo exclusivo por uma aplicação, por um determinado período de tempo. Tal abordagem certamente faz sentido, uma vez que, analisando-se o desempenho, deseja-se obter o menor tempo de execução possível. No entanto, o aumento da complexidade e da quantidade dos problemas e das aplicações que buscam resolvê-los torna as infraestruturas de HPC incapazes de atender simultaneamente a todos os requisitos mínimos de hardware que algumas delas precisam para executar. Devido a isso, é necessário que diferentes políticas de fila sejam implementadas para organizar o fluxo de execução e para garantir que todas as aplicações tenham os seus requisitos mínimos de hardware satisfeitos. Ademais, a existência de filas de execução pode levar a situações nas quais a infraestrutura de HPC seja subutilizada. Um exemplo dessa subutilização é observada no cenário hipotético apresentado a seguir:

- uma infraestrutura dispõe de 1024 núcleos de processamento;
- em um instante de tempo t, uma aplicação app_1 é submetida à essa infraestrutura com a necessidade de utilizar 512 núcleos de processamento;
- em um instante de tempo $t+x$, uma segunda aplicação app_2 também é submetida à infraestrutura, requisitando 256 núcleos para executar. Como os requisitos de app_2 podem ser atendidos, a mesma entra em estado de execução;
- em um instante de tempo $t+y$, com $y > x$, uma terceira aplicação app_3 é submetida à infraestrutura com o requisito de 384 núcleos de processamento para executar. Considerando que as aplicações app_1 e app_2 não concluíram suas execuções, não há núcleos suficientes para atender aos requisitos de app_3. Logo, a mesma deve ser colocada em uma fila até que existam núcleos suficientes para satisfazer os seus requisitos de execução.

Ao realizar uma breve análise do cenário descrito, é possível observar que 256 núcleos de processamento (1/4 da infraestrutura) permanecem ociosos, mesmo havendo uma aplicação aguardando na fila. Também é possível concluir que a terceira aplicação deverá aguardar o término de uma das outras duas aplicações, o que causará o acréscimo do seu tempo de término. A situação descrita utiliza o modelo de alocação *Job Scheduling* [81, 18, 32], que ainda é amplamente utilizado em ambientes de HPC, e cujo objetivo é maximizar a quantidade de aplicações executadas em um período de tempo, porém sem

permitir que elas compartilhem os recursos de processamento disponíveis. Uma estratégia que permita o compartilhamento dos recursos e que resulte na maior vazão de execução de aplicações é benéfica tanto para os usuários como para os provedores de serviço.

Melhores resultados podem ser obtidos, tanto no tocante ao melhor aproveitamento da infraestrutura computacional, como na redução do tempo em fila de espera. Para que isso seja possível, é necessário que os requisitos de recursos para a execução de cada aplicação (quantidade de núcleos, memória, entre outros) sejam definidos pelo usuário. Algumas aplicações, classificadas como moldáveis [31], podem ser executadas com diferentes quantidades de processos, embora tal quantidade não varie durante a sua execução. Em resumo, a ordem de execução das aplicações pode causar um aumento do tempo médio de espera em fila e da quantidade de recursos computacionais ociosos, ao mesmo tempo em que privilegia o tempo de execução individual das aplicações.

As aplicações que fazem uso de ambientes de HPC nem sempre consomem integralmente os recursos computacionais que são alocados para elas durante todo o período que estão em execução [83]. Ao mesmo tempo, o uso exclusivo desses recursos por apenas uma aplicação durante a sua execução contribui para a sua subutilização. Modelos como a computação em nuvem [15, 35, 51] já exploram o uso de recursos computacionais compartilhados para oferecer diferentes tipos de serviço, com o propósito de maximizar a utilização da infraestrutura e minimizar os custos associados à sua manutenção. Ao considerar que os ambientes de HPC nem sempre são utilizados em sua plenitude, vislumbrou-se a possibilidade de aplicar um modelo que explore o compartilhamento de recursos também nesses ambientes.

O relatório Magellan [27] e a pesquisa realizada por Luszczek *et al.* [49] apresentam resultados do uso da virtualização e de conceitos de computação em nuvem no âmbito de HPC. As conclusões de ambos convergem para o uso de recursos computacionais na nuvem apenas como complemento aos de HPC. A justificativa desses trabalhos é fundamentada nas questões relativas à perda de desempenho causada pela virtualização e pelo acesso concorrente dos recursos pelas aplicações. Em contrapartida, o trabalho de Liu *et al.* [47] demostra que o uso da virtualização e do compartilhamento de recursos não pode ser descartado apenas porque houve aumento do tempo de execução, pois é possível que o aumento dos tempos de execução sejam aceitáveis, desde que a infraestrutura seja melhor aproveitada. Para isso, é necessário também que os mecanismos de escalonamento convencionais sejam revistos e que sejam capazes de lidar com o compartilhamento de recursos e forma eficiente. Ao mesmo tempo, o relatório do Departamento de Energia

dos Estados Unidos da América (DoE) [38] e o projeto *High Performance Computing for Energy* (HPC4E) [39] também apontam para o uso de recursos compartilhados, porém, como uma abordagem para a redução do consumo energético em ambientes de computação em exaescala.

Sistemas escalonadores amplamente utilizados, como o HTCondor [74] e o *Portable Batch System* (PBS) [11] visam garantir a alocação exclusiva dos recursos, com o objetivo de minimizar o *makespan* de cada aplicação executada. Logo, tais sistemas não foram projetados para lidar com o compartilhamento de recursos [52]. Embora não seja possível reduzir o *makespan* de uma aplicação ao introduzir o compartilhamento de recursos, outros benefícios pode ser alcançados, como a redução do *makespan* do *workload* ouou o aumento da vazão de processamento do ambiente computacional. Portanto, se faz necessário o desenvolvimento de novas estratégias de escalonamento de aplicações, com a finalidade de viabilizar o compartilhamento de recursos para melhor aproveitar a infraestrutura disponível, ao mesmo tempo em que sejam oferecidas contrapartidas relacionadas ao tempo de execução das aplicações.

O desenvolvimento de uma nova estratégia de escalonamento que considere o compartilhamento de recursos envolve o cumprimento de múltiplos objetivos. Enquanto os sistemas tradicionais de escalonamento visam apenas a minimização do *makespan* de uma aplicação, estratégias de escalonamento que admitem o compartilhamento de recursos acrescentam apenas o aumento da taxa de utilização desses recursos como um novo objetivo [59]. No entanto, a redução do tempo médio de *turnaround*[1] e do tempo de espera em fila também podem ser definidos como novos objetivos para o escalonamento [59].

O escalonamento de aplicações em ambientes distribuídos é um problema desafiador. As estratégias existentes enfrentam limitações relativas à escalabilidade, uma vez que elas tentam gerenciar todas as informações acerca do estado do ambiente [5]. Essas limitações impedem que tais estratégias sejam utilizadas para escalonar aplicações multitarefas em ambientes computacionais que possuem milhares ou milhões de *hosts*, tais como grades e nuvens computacionais [60]. Portanto, estratégias descentralizadas tornam-se alternativas atrativas para realizar o escalonamento eficiente das aplicações em ambientes com essas características.

É importante destacar que alguns dos objetivos de escalonamento podem ser antagônicos quando há o compartilhamento de recursos. Por exemplo, aumentar a taxa de utilização do ambiente, com o propósito de atender a um maior número de aplicações

[1] Somatório do tempo de espera na fila e do tempo de execução de cada aplicação.

simultaneamente, pode fazer com que os tempos de execução dessas aplicações aumente significativamente. Em contrapartida, privilegiar apenas o tempo de execução pode levar à subutilização do ambiente de HPC [47, 52]. Portanto, é necessário que novas estratégias de alocação definam um conjunto de objetivos e que permita escolher aquele que deve ser alcançado a cada novo escalonamento realizado.

Esta tese apresenta uma estratégia descentralizada de escalonamento, na qual as próprias aplicações[2] gerenciam a execução de suas tarefas. A estratégia tem o objetivo de abordar o problema de escalonamento de aplicações que compartilham os recursos computacionais de processamento, e que talvez possua maior potencial para lidar melhor com a escalabilidade inerente a sistemas de grades e nuvens. A estratégia foi estruturada de maneira que as aplicações escalonadas se comportem como uma sociedade, isto é, buscando encontrar o melhor escalonamento para si, mas evitando prejudicar a execução das demais. Cada aplicação é capaz de realizar o gerenciamento da sua própria execução, de acordo com um conjunto de comportamentos previamente definidos, além de serem cientes do consumo de recursos que elas utilizam.

A estratégia proposta nesta tese foi elaborada com a finalidade de ser capaz de fornecer diferentes resultados de escalonamento para diferentes execuções de um mesmo conjunto de aplicações (*workload*). Os diferentes escalonamentos são alcançados por meio do ajuste de um parâmetro que representa a prioridade de cada aplicação. Os diferentes ajustes do parâmetro de cada aplicação permite que elas alterem os seus comportamentos, aumentando ou reduzindo a intensidade de uso dos recursos, ao detectarem que os mesmos encontram-se ociosos ou utilizados por outras aplicações. Assim, o sistema de escalonamento opera como um sistema multiagentes, no qual as aplicações cooperam entre si para compartilhar os recursos de forma a aumentar a utilização da infraestrutura, ao mesmo tempo em que buscam reduzir o makespan do *workload*. Os desdobramentos da abordagem de escalonamento proposta são relevantes para dois grupos de interesse específicos: os usuários das aplicações obtém os seus resultados mais rapidamente, devido à redução do tempo médio de espera. Já os gestores dos recursos (provedores de serviços, por exemplo) são capazes de atender mais aplicações simultaneamente, além de aumentar a taxa de utilização do ambiente e de reduzir o tempo total necessário para executar um determinado *workload*.

[2]Nesta tese, uma aplicação possui a mesma conotação de um *job*. Cada aplicação é composta por um conjunto de tarefas que devem ser escalonadas e executadas.

1.1 Motivação

As limitações dos escalonadores tradicionais em lidar de forma eficiente com a alocação de recursos compartilhados para executar as aplicações torna-se um fator restritivo para a sua adoção. Além disso, esses escalonadores muitas vezes não conseguem sequer cumprir com o seu real propósito com eficiência, pois os resultados podem levar a tempos de espera em fila muito elevados. Com a finalidade de corroborar esta afirmação, foi realizada uma análise sobre experimentos de alocação de aplicações em um supercomputador. As conclusões obtidas a partir dessa análise serviram como justificar e motivar a elaboração desta tese por meio de dados empíricos.

Os experimentos foram executados no supercomputador Santos Dumont [48] e seguiram a sua política atual de alocação exclusiva de *hosts*, definida pelo escalonador SLURM *workload manager* [42]. Essa política determina que todos os 24 núcleos de processamento de um determinado *host* são utilizados somente pelos processos da aplicação para a qual ele foi escalonado. Ou seja, se porventura uma aplicação utilizar somente um núcleo de um *host*, o demais 23 permanecerão ociosos durante todo o período em que ela se encontrar em execução.

O experimento envolveu a submissão de um total de 12 instâncias da aplicação *Block Tri-diagonal solver* para execução. Ela é parte integrante do conjunto de aplicações de dinâmica de fluidos *NAS Parallel Benchmarks* (NPB) [6], utilizado para avaliar o desempenho de supercomputadores. Os tamanhos dos problemas NPB são pré-definidos e divididos em diferentes classes [53]. Seis instâncias foram configuradas para utilizar entrada da classe B. As outras 6 instâncias utilizaram entrada da classe C. Cada instância foi submetida ao supercomputador Santos Dumont com requisitos específicos referentes á quantidade de *hosts*. Na Tabela 1.1 são apresentados os tempos de execução e de espera na fila de cada uma das 12 instâncias, de acordo com a classe de dados de entrada e a quantidade de *hosts* requisitada.

A análise dos dados apresentados na Tabela permitiu concluir que o tempo médio de espera na fila de execução foi de aproximadamente 7 dias. Por outro lado, o tempo médio de execução das aplicações com classe B foi de 26 segundos, enquanto os problemas de classe C apresentaram tempo médio de execução igual a 4 minutos e 40 segundos. Isso mostra como um escalonamento inadequado pode causar tempos de espera em fila muito elevados, quando comparados ao tempo de execução das aplicações. Além disso, não há garantias que todos os núcleos de processamento dos *hosts* são utilizados, uma vez que isso

Tabela 1.1: Tempos de execução e de espera em fila de cada uma das 12 instâncias da aplicação *Block Tri-diagonal solver* executadas de acordo com a classe de dados de entrada e com a quantidade de *hosts* alocados

		2 *hosts*	3 *hosts*	11 *hosts*	23 *hosts*	43 *hosts*	85 *hosts*
Classe B	Espera em fila	6 dias e 21 horas	6 dias e 21 horas	6 dias e 22 horas	6 dias e 23 horas	6 dias e 23 horas	7 dias e 2 horas
	Tempo de execução	46 segs	28 segs	11 segs	14 segs	21 segs	37 segs
Classe C	Espera em fila	6 dias e 22 horas	6 dias e 23 horas	6 dias e 23 horas	7 dias e 1 hora	7 dias e 1 hora	7 dias e 3 horas
	Tempo de execução	14 mins e 15 segs	8 mins e 41 segs	2 mins e 16 segs	1 min e 8 segs	51 segs	52 segs

depende do modelo de programação adotado no desenvolvimento da aplicação paralela. Também não há garantias de que a aplicação tenha consumido os recursos de processamento de forma a justificar a alocação exclusiva dos recursos. Portanto, fica evidente que a subalocação de um ambiente de HPC pode ocorrer, mesmo em um supercomputador pertencente à lista Top 500[3] e que utiliza um escalonador amplamente difundido.

1.2 Objetivos

O principal objetivo desta tese de doutorado é propor uma nova estratégia descentralizada de escalonamento de aplicações autonômicas em ambientes computacionais compartilhados. A intenção é melhor aproveitar os recursos de processamento disponíveis, por meio da redução ou da completa eliminação dos períodos de tempo em que eles permanecem ociosos. Com isso, almeja-se aumentar a capacidade de atendimento do ambiente, ao permitir que um maior número de aplicações sejam executadas simultaneamente. A estratégia proposta deve ainda alcançar a redução do tempo necessário para executar um conjunto de aplicações ou, no pior dos casos, não permitir que ele aumente de forma significativa. Esta abordagem difere daquelas implantadas por estratégias convencionalmente adotadas, as quais buscam reduzir o tempo de execução individual de cada aplicação, causando frequentemente a subutilização da infraestrutura computacional disponível.

Outro objetivo da estratégia de escalonamento proposta é a definição de um modelo de transição de estados que permita resolver o problema descrito de forma distribuída,

[3]Lista dos 500 supercomputadores com maior capacidade computacional: www.top500.org - Classificação de junho de 2017.

sem comunicação para tornar a solução escalável de implementação simples. Cada estado determina um comportamento que as aplicações podem adotar durante as suas execuções. Os comportamentos definem quantas tarefas[4] a aplicação deve executar, e são baseados na relação entre a quantidade de processamento remanescente da aplicação, o tempo estimado para término, o tempo considerado ideal para que a execução da aplicação seja concluída e a capacidade computacional disponível para realizar o trabalho.

Um parâmetro denominado **Tempo Alvo (TA)** é empregado para representar a prioridade de uma aplicação do *workload* em relação à utilização dos recursos. Logo, esse parâmetro é responsável por guiar o escalonamento das tarefas da aplicação, que continua durante todo o período no qual ela se encontra em execução. Uma vez que o ambiente computacional é compartilhado, as tarefas de cada aplicação são executadas de acordo com a sua prioridade e de acordo com o nível da utilização do ambiente naquele momento. Sendo assim, se uma aplicação possui prioridade baixa e o sistema encontra-se com alta taxa de utilização de recursos de processamento, a quantidade de tarefas executadas é reduzida. Por outro lado, caso a prioridade seja alta, o AMS da aplicação aumenta a quantidade de tarefas executadas. Portanto, o resultado do escalonamento de uma aplicação no ambiente compartilhado, em diferentes execuções, varia conforme o valor especificado para o parâmetro **Tempo Alvo**. Os valores distintos que podem ser atribuídos para as prioridades das aplicações concorrentes refletem em diferentes escalonamentos de um mesmo *workload*. Isso permite que a estratégia alcance uma maior taxa de utilização dos recursos e uma maior redução do *makespan* de um *workload*.

O uso de um *broker* centralizado apresenta limitações em ambientes cuja quantidade de recursos computacionais pode variar e alcançar a ordem de milhares ou milhões [3]. Com o objetivo de eliminar tais limitações, a estratégia de escalonamento proposta foi elaborada de uma forma em que cada aplicação é capaz de adotar um comportamento próprio. Em virtude disso, o escalonamento proposto considera que as aplicações são autônomas e que elas realizam as tomadas de decisão a respeito do escalonamento das suas próprias tarefas. Para tornar isso possível, foi admitido: i) que cada aplicação possui o seu próprio AMS, responsável por controlar as variações de comportamento de acordo com a prioridade da aplicação e com a taxa de utilização dos recursos de processamento do ambiente; e ii) o uso de aplicações maleáveis [31].

A validação da estratégia de escalonamento foi realizada por meio do uso de um

[4]Nesta tese, uma tarefa se refere a um processo que deve ser executado. Cada aplicação (ou *job*) é composta por um conjunto de tarefas. Um *workload* é composto por um conjunto de aplicações.

conjunto de aplicações paralelas maleáveis[5] do tipo *bag of tasks*, de modo que elas compartilhassem os recursos de processamento durante as suas execuções. A escolha desta classe de aplicações se dá pelo fato de que suas tarefas não dependem do término de outras tarefas para iniciar suas execuções. Com isso, os únicos requisitos relacionados à execução das tarefas desta classe de aplicações são referentes aos recursos que elas necessitam, razão pela qual elas podem entrar em execução tão logo encontrem-se no estado de "prontas". Por outro lado, alcançar melhores resultados quanto à redução do *makespan* de conjuntos de aplicações deste tipo é um desafio. Em virtude disso, é necessário que a coordenação dos recursos computacionais do ambiente seja feito na forma mais eficientes para garantir que o compartilhamento dos recursos não comprometa o desempenho das aplicações, mesmo em situações com elevado grau de concorrência.

Por fim, a estratégia de escalonamento proposta tem o objetivo de promover o melhor aproveitamento dos ambientes computacionais de alto desempenho, ao permitir que aplicações compartilhem os seus recursos de processamento. Os possíveis desdobramentos deste trabalho envolvem a redução de custos relativos à manutenção do ambiente e ao consumo energético, inclusive possibilitando o seu uso em ambientes de computação em nuvem. Isso pode ser alcançado com a adoção de uma estratégia de escalonamento capaz de escalonar um mesmo *workload* de formas diferentes, com os objetivos de reduzir o seu *makespan* e de aumentar a taxa de utilização do ambiente computacional.

1.3 Contribuições

O trabalho relatado nesta tese apresenta as seguintes contribuições relacionadas ao escalonamento de aplicações que compartilham os recursos de uma infraestrutura computacional:

- A elaboração, o desenvolvimento e a validação de uma estratégia descentralizada de escalonamento de aplicações autonômicas que compartilham recursos computacionais de processamento. Por meio da escolha adequada do valor de apenas um parâmetro, a estratégia em questão é capaz de realizar escalonamentos distintos de um mesmo *workload*, o que simplifica o uso da estratégia para diferentes situações de escalonamento. Os diferentes escalonamentos que podem ser alcançados buscam reduzir o tempo necessário para executar o conjunto de aplicações, ao mesmo tempo em que proporciona o aumento da taxa de utilização da infraestrutura.

[5] Aplicações configuradas para iniciar a executar com uma quantidade de tarefas. No entanto essa quantidade pode variar em tempo de execução

- A definição de um conjunto de comportamentos que são adotados pelas aplicações e a possibilidade de variá-los durante as suas execuções. Esses comportamentos estendem o conceito de aplicações autônomas, ao permitir que decisões sejam tomadas visando beneficiar a execução do conjunto de aplicações. Com isso, cada aplicação pode sofrer variações de comportamento de maneira a reduzir o *makespan* do *workload*, ou de aumentar a taxa de utilização dos recursos do ambiente.

- Por fim, a maior contribuição desta tese está na constatação de que é possível mudar o paradigma de escalonamento de aplicações em ambientes compartilhados. Os resultados obtidos por meio dos experimentos mostraram a aplicabilidade da estratégia e que é possível alcançar ganhos significativos com a sua adoção. No entanto, os estudos a respeito do tema devem ser aprofundados, com a finalidade de explorar o completo potencial deste novo paradigma, além de identificar quais são as situações mais adequadas para utilizá-lo.

- Os diferentes escalonamentos resultantes, proporcionados pela estratégia proposta, buscam beneficiar dois grupos de interesse distintos: os usuários das aplicações, ao reduzir o tempo de espera na fila e, consequentemente, o tempo de resposta para obter os resultados das aplicações; e os gestores dos recursos, ao permitir que um maior número de usuários e aplicações sejam atendidos simultaneamente.

Adicionalmente, a pesquisa realizada e o trabalho desenvolvido também contribuíram direta e indiretamente com trabalhos relacionados ao escalonamento de máquinas e *clusters* virtuais em ambientes de nuvem.

1.4 Síntese do capítulo e organização do texto da tese

Este capítulo foi escrito com o propósito de expressar a motivação que levou à realização da pesquisa apresentada no decorrer desta tese. Além disso, foram apresentados também os objetivos e as contribuições alcançadas com o trabalho realizado.

O Capítulo 2 apresenta a revisão da literatura sobre asuntos correlatos ao problema de escalonamento de aplicações em ambientes computacionais compartilhados. O Capítulo 3 apresenta o histórico e o aprofundamento do estudo sobre o problema de escalonamento de aplicações em ambientes compartilhados. Nele, é apresentado também o problema de escalonamento considerado nesta tese e a solução proposta para resolvê-lo. O Capítulo 4 é usado para descrever o conjunto de experimentos realizados para validar a solução pro-

posta e para discutir os resultados obtidos. Por fim, o Capítulo 5 apresenta as conclusões da tese, os trabalhos futuros identificados para dar continuidade à pesquisa iniciada na tese e a produção científica associada.

Capítulo 2

Trabalhos correlatos ao tema abordado

Este capítulo é dedicado a apresentar trabalhos relacionados ao tema de escalonamento que é abordado nesta tese de doutorado. É realizada também uma análise crítica acerca de cada um deles, além de apontar os aspectos relevantes para a tese.

O tema da pesquisa envolve o escalonamento e a execução de aplicações autonômicas que utilizam os recursos de ambientes computacionais compartilhados. Assim, trabalhos relacionados ao escalonamento de aplicações autonômicas, à interferência sobre o desempenho de aplicações concorrentes e à competição por recursos, entre outros, são apresentados e discutidos.

2.1 Evolução das Infraestruturas Computacionais

Em 1965, a declaração do cofundador e então presidente da Intel, Gordon E. Moore de que a quantidade de transistores de *chips* de processamento aumentaria em 100%, a cada 18 meses, com um mesmo custo, foi mais do que um marco inicial da evolução tecnológica do que iria acontecer nos 50 anos seguintes. O que passou a ser conhecida como Lei de Moore serviu como uma regra a ser seguida pela indústria de eletrônicos, e que promoveu contribuições nas área tecnológicas, econômicas e sociais [40].

Durante aproximadamente 30 anos, a partir da década de 1960 até o início da década de 1990, a Lei de Moore continuou a ser uma meta a ser perseguida pelos fabricantes de microprocessadores, com o objetivo de manter a indústria competitiva. Durante o período citado, o tamanho dos transistores nos *chips* de computadores continuou diminuindo exponencialmente até atingir a escala de nanômetros.

Porém, o aumento da densidade dos *chips* de computadores enfrenta limites físicos

[43]. Ao considerar somente um *chip* de processador, tais limites físicos impactam sobre a continuação da evolução tecnológica prevista pela Lei de Moore. No entanto, a partir de meados da década de 1980, a pesquisa e a elaboração de arquiteturas computacionais inovadoras permitiram o projeto e o uso de um conjunto de processadores que atuam de forma conjunta, com o objetivo de executar uma mesma aplicação.

Dongarra *et al.* [28] aponta a introdução dos sistemas de computadores vetoriais como sendo o marco inicial da Supercomputação moderna. Um computador vetorial é uma CPU que implementa um conjunto de instruções que operam sobre vetores unidimensionais, enquanto que processadores escalares operam sobre dados unitários [22]. Sistemas de computadores vetoriais apresentavam desempenho superior em pelo menos uma ordem de magnitude, se comparado aos computadores convencionais da época. O desempenho, que até então era considerado o único argumento que pesava a favor desta arquitetura, passou a ser considerado menos importante no início de década de 1980, devido à integração dos sistemas vetoriais com ambientes computacionais convencionais. Somente os fabricantes que forneceram ambientes de programação, sistemas operacionais e aplicações específicas conseguiram prosperar ao atrair usuários de grande potencial, como indústrias e centros de pesquisa. Apesar dos esforços, o aumento de desempenho foi basicamente através do desenvolvimento tecnológico na produção de *chips* e da produção de sistemas multiprocessados de memória distribuída.

Durante a primeira metade da década de 1980, uma série de programas governamentais norte americanos incentivou o desenvolvimento de sistemas paralelos com memória distribuída. Os principais objetivos eram superar as limitações de escalabilidade dos sistemas de memória compartilhada com a possibilidade de crescimento contínuo da capacidade computacional e a redução do custo de obtenção e de manutenção de sistemas de alto desempenho [28]. A evolução de arquiteturas e técnicas com base em sistemas de memória distribuída proporcionaram a implantação e a difusão do paradigma de computação paralela, o que foi determinante para que a demanda gerada por problemas cada vez mais complexos continuasse sendo atendida por sistemas computacionais de alto desempenho e de baixo custo.

Ademais, também com o propósito de continuar permitindo o aumento da capacidade computacional, apesar das limitações físicas existentes, *clusters* computacionais continuam sendo utilizados até os dias atuais. Trata-se de um conjunto de computadores, preferencialmente homogêneos, interconectados por uma rede de alta velocidade. Essa infraestrutura permite que as aplicações sejam executadas de acordo com o modelo de

programação paralela com memória distribuída e a comunicação ocorre através do uso de interfaces de trocas de mensagens [7]. Os *clusters* são amplamente utilizados para viabilizar e reduzir o tempo de execução de aplicações complexas, que possuem grandes quantidades de dados de entrada e que requerem elevada capacidade de processamento.

A partir da primeira metade da década de 1990, a ampla adoção de sistemas distribuídos passou a contribuir de modo a difundir o conhecimento, compartilhar recursos e proporcionar alguma espécie de cooperação entre a academia e a indústria. A integração de sistemas heterogêneos, compostos por centenas ou milhares de computadores conectados através de uma rede global, é de fundamental importância para o paradigma de computação atualmente praticado [20]. Tal paradigma está presente em aplicações diversas, que podem acessar ou ser acessadas por uma grande variedade de recursos, desde dispositivos móveis até os supercomputadores.

As grades computacionais exercem um papel histórico importante, no que se refere à colaboração entre participantes de um sistema distribuído através do compartilhamento de seus recursos. As grades computacionais são definidas por Foster *et al.* [34] como um conjunto de serviços, protocolos e ferramentas que têm o objetivo de suportar a escalabilidade característica de sistemas distribuídos. Os recursos oferecidos por um ambiente de grade incluem soluções de segurança para o gerenciamento de credenciais e de políticas, em situações que envolvem múltiplos usuários; serviços e protocolos para o gerenciamento de recursos para garantir o acesso remoto seguro aos dispositivos e dados que fazem parte do ambiente; protocolos para busca de informações sobre o estado de recursos e serviços; e serviços para o gerenciamento de dados, que garantem a integridade dos dados que trafegam entre os dispositivos que fazem parte da grade [34, 33].

Por ser de natureza heterogênea, não é pouco comum encontrar ambientes de grades computacionais que possuam um ambiente de alto desempenho de memória distribuída como um de seus recursos computacionais. Para que isso seja possível, é necessário que haja uma integração entre os ambientes que seja transparente para os usuários. A abstração da heterogeneidade dos recursos de rede e de *hardware*, dos Sistemas Operacionais e das linguagens de programação é realizada por uma camada de *software* denominada *middleware* [20].

Os ambientes computacionais dedicados para o alto desempenho, como os *clusters* e até mesmo as grades computacionais, possuem elevado custo de obtenção e esforço de manutenção. Por isso, não são muitas as empresas e centros de pesquisa capazes de manter tais infraestruturas. Sendo assim, a partir do início dos anos 2000, um modelo de com-

putação começou a ser idealizado de modo a difundir o acesso a recursos computacionais de alta vazão, com custo reduzido, e cuja manutenção fica sob a responsabilidade de uma empresa contratada, conforme sugerido por Foster *et al.* em [35]. Em *Buyya et al.* [15], é sugerido ainda que tal modelo fosse fornecido como uma utilidade, tal como energia elétrica.

O modelo de computação em nuvem, de acordo com o NIST (*National Institute for Standards and Technologies*) [51], foi idealizado de modo a proporcionar o acesso remoto e sob demanda a um conjunto compartilhado de recursos computacionais diversos. A infraestrutura disponibilizada através do modelo de nuvens pode ser rapidamente alocada ou desalocada, de acordo com a demanda.

A diversidade de ambientes de alto desempenho, capazes de atender à demanda por recursos computacionais de um conjunto de aplicações científicas igualmente vasto, faz com que seja necessário o uso de mecanismos responsáveis por conduzir a execução de tais aplicações, desde a escolha do recurso mais adequado para o processamento, até que elas terminem de executar. Em função dessa necessidade, as políticas de escalonamento (ou alocação) de recursos devem considerar, de acordo com os objetivos a serem atingidos, as combinações mais apropriadas entre aplicações e recursos a serem realizadas.

2.2 Escalonamento de aplicações e métricas

O planejamento e a correta execução de um projeto envolve a alocação adequada das tarefas a serem executadas, que possuem necessidades específicas, a um conjunto de recursos capazes de satisfazê-las. Esse problema, conhecido como escalonamento de tarefas, está presente em diversas áreas, tais como produção industrial, gerenciamento de projetos, prestação de serviços e na computação, entre outros. Independente da área à qual o escalonamento de tarefas esteja associado, é uma característica comum deste problema o fato de não existir uma solução algorítmica eficiente, capaz de encontrar uma solução ótima em tempo polinomial [18].

De maneira geral, o problema de escalonamento de aplicações é descrito por Silberschatz et al. em [68] como uma forma de sempre manter a maior quantidade de processos computacionais em execução. Os sistemas com um único núcleo de processamento se apresentam como o cenário menos complexo de escalonamento, uma vez que nunca haverá mais de uma aplicação em execução em um mesmo recurso computacional, em um mesmo período de tempo. Ainda assim, não é um problema que admita solução em tempo

polinomial, fazendo parte da classe de problemas NP-completo [50].

2.2.1 Escalonamento de aplicações sequenciais

Na computação, o *job-shop scheduling* é um dos problemas de escalonamento de tarefas mais conhecidos e estudados. De acordo com Chen *et al.* [18], este problema considera um conjunto de tarefas e outro conjunto de máquinas (ou recursos). Cada tarefa consiste de uma sequência de instruções que devem ser executadas de modo ininterrupto em um determinado recurso. Cada recurso processa apenas uma operação de uma tarefa por vez. Um escalonamento é uma alocação das operações nos intervalos de tempo dos recursos disponíveis e o objetivo é encontrar um escalonamento de duração mínima [18, 81].

Para que o escalonamento de uma aplicação resulte na sua execução correta, é necessário que cada tarefa seja escalonada somente após todas as suas dependências terem sido executadas corretamente.

Segundo Mastrolilli e Svensson [50] e Anand e Panneerselvam [4], as principais métricas do problema *Job-Shop Scheduling* consideradas pela literatura, dado que C_j é o tempo necessário para completar a execução da última tarefa de uma aplicação j, são:

- *Makespan*: o *Makespan* (L_{max}) é o tempo de conclusão da última aplicação a ser completada. Esta métrica indica o tempo que um determinado conjunto de n aplicações precisa para ser concluído. Desta forma, o objetivo é minimizá-lo de maneira que $L_{max} = min(C_j)$.

- Tempo de execução: é o intervalo de tempo compreendido entre o início da execução de uma aplicação e o seu término. É considerado tanto o tempo de processamento, como o tempo necessário para a realização das operações de entrada e saída.

- Tempo de atendimento: considera os tempos necessários para a submissão e para a execução de uma aplicação, somado ao tempo de espera na fila de escalonamento;

- *Deadline* de uma aplicação: é o tempo máximo dentro do qual se deseja finalizar a execução de todas as tarefas de uma aplicação. O *deadline* de uma aplicação geralmente é estabelecido pelo próprio usuário ou entidade que o submete. A extrapolação do *deadline* deve ser evitada para que o usuário obtenha o resultado da aplicação dentro do período de tempo esperado.

- Meta ou prazo (*Due date*): é o tempo limite dentro do qual uma aplicação deve ser

completada. Esse tempo pode ser reajustado pelo escalonador, desde que o *deadline* não seja comprometido.

- *Stretch*: métrica definida por Bender *et al.* [10] e adotada por Casanova *et al.* [16] para a análise de desempenho de aplicações maleáveis que competem pelos recursos de uma infraestrutura computacional. No contexto de escalonamento *offline*, consiste da razão entre o *makespan* de uma aplicação em ambientes com concorrência e o *makespan* da mesma aplicação em ambientes dedicados para a sua execução. Contudo, a adoção desta métrica para escalonamento *online* é viável ao adotar os tempos de execução das aplicações em detrimento do *makespan*.

É importante ressaltar que a escolha das métricas que serão adotadas como objetivos a serem alcançados depende to tipo de escalonamento a ser realizado [30]. Por exemplo, em situações nas quais o escalonamento é *online* (quando aplicações que precisam ser escalonadas continuam chegando ao sistema indefinidamente) é comum trabalhar sobre métricas associadas a cada aplicações, como a meta, o *deadline* ou o tempo de execução. Em sistemas *offline*, por sua vez, utiliza-se o *makespan* como uma das principais métricas, uma vez que é conhecido todo o conjunto de aplicações, além do tempo de execução individual e do tempo de chegada de cada uma delas.

As métricas citadas são consideradas pela literatura para sistemas computacionais nos quais um recurso computacional processa apenas uma aplicação por vez [50]. No entanto, elas podem ser inseridas para a análise de situações em que haja a concorrência por recursos computacionais [16]. As técnicas que visam otimizar tais sistemas não consideram a execução de aplicações que competem por recursos compartilhados, causando a subutilização dos recursos. Por essa razão, torna-se necessário definir novas métricas e estratégias de escalonamento que considerem máquinas capazes de processar duas ou mais aplicações em um mesmo instante de tempo [84].

2.2.2 Escalonamento de aplicações paralelas

Os sistemas multi-núcleos ou de memória distribuída possibilitam a divisão do processamento de uma mesma aplicação em dois ou mais recursos. Surge também a possibilidade de processar duas ou mais aplicações simultaneamente. Em razão desta possibilidade, é estabelecido um problema de escalonamento diferente daquele que envolve somente um único núcleo. Os objetivos desse problema envolvem o arranjo das aplicações de maneira a reduzir o tempo total para executar todas as aplicações (*makespan*), em sistemas *offline*,

e reduzir o tempo de execução das aplicações, em sistemas *online*.

A busca por estratégias de escalonamento de aplicações mais eficientes implica na necessidade de identificar os comportamentos destas em relação à variação da utilização dos recursos computacionais durante o tempo em que elas se encontram em execução [76].

Quanto às características dos dispositivos computacionais utilizados por um escalonador, podem ser consideradas informações estáticas, tais como a quantidade total de núcleos de processamento e a quantidade de memória compartilhada por estes núcleos. Também podem ser consideradas as informações dinâmicas, tais como a taxa de utilização de cada núcleo de processamento, o total de núcleos ocupados, ou o tempo necessário para que um determinado recurso computacional consiga atender todas as aplicações escalonadas para ele, entre outras.

No entanto, características como a quantidade de núcleos alocados para uma aplicação podem ser consideradas como estáticas ou dinâmicas, dependendo da lógica de programação adotada. Aplicações estáticas são aquelas que independente da quantidade de núcleos de processamento disponíveis, sempre demandará uma mesma quantidade, uma vez que isso foi definido no código da aplicação. Existem também aquelas que podem se adequar à disponibilidade de núcleos disponíveis, mas que, uma vez iniciada a execução, devem utilizar a quantidade de núcleos especificada até o fim da execução. A estas aplicações, dá-se o nome de moldáveis. Por fim, existem aplicações que são capazes de variar, em tempo de execução, a quantidade de núcleos de processamento utilizados. Tais aplicações são definidas como maleáveis por Feitelson e Rudolph em [31].

2.2.3 Escalonamento de aplicações em ambientes compartilhados

Uma variação do problema de escalonamento em sistemas paralelos e distribuídos surge da necessidade de aumentar a taxa de ocupação dos recursos computacionais, com o objetivo de melhor aproveitar a infraestrutura disponível. O problema de escalonamento de aplicações que competem por recursos compartilhados é visto na literatura atual como uma proposta para abordar a alta demanda por recursos de alto desempenho [62, 26, 63]. Os objetivos, além daqueles especificados para o problema *Job-Shop Scheduling* clássico, envolvem também a melhora da utilização dos recursos e a redução de eventuais perdas de desempenho das aplicações, que podem ocorrer devido à concorrência entre elas [84]. A análise é realizada através da comparação dos resultados das execuções em ambientes compartilhados, nos quais um recurso é capaz de executar mais de uma aplicação simultaneamente, e ambientes dedicados, nos quais cada recurso executa apenas uma aplicação

por vez.

O primeiro problema relatado pela literatura, no que se refere ao escalonamento de aplicações com competição, é o aumento do tempo de execução, quando comparado ao tempo de execução em ambientes livres de concorrência [85, 29]. Isso se deve ao comportamento das aplicações em relação ao acesso e à utilização dos recursos computacionais e à variação que seus comportamentos sofrem devido à concorrência com outras aplicações. No entanto, mais importante do que o tempo de execução individual de uma aplicação, é o tempo total necessário para escalonar um conjunto de aplicações [31].

Polo *et al.* [59] apresenta uma estratégia de escalonamento adaptativo para aplicações *MapReduce*. O objetivo é analisar o escalonamento concorrente entre aplicações *MapReduce* e de banco de dados transacionais, sem que os impactos decorrentes da concorrência influenciem negativamente nas execuções de ambas as aplicações.

Apesar dos resultados positivos apresentados, os autores de [59] ressaltam que, embora seja teoricamente possível aplicar a estratégia apresentada para diferentes aplicações, não foram realizados experimentos que explorassem a competição entre mais do que duas aplicações. Também não foram consideradas características diferentes daquelas presentes por aplicações *MapReduce* e de bancos de dados transacionais. Essa observação sugere questionamentos sobre como características diferentes, tais como o uso de CPU e o acesso à memória, influenciam na execução concorrente das aplicações.

2.3 Escalonamento autonômico de aplicações

Salehi e Tahvildari [66] definem a computação autonômica como um paradigma voltado para o projeto, o desenvolvimento, a implantação e a manutenção de ambientes computacionais, cuja inspiração é oriunda de sistemas biológicos capazes de lidar com fatores complexos como a heterogeneidade e a incerteza. Um sistema autonômico possui quatro características: auto-configuração (*self-configure*), auto-cura (*self-healing*), auto-otimização (*self-optimize*) e auto-proteção (*self-protect*). Por sua vez, Schanne *et al.* apresenta duas visões para a computação autonômica em [67]. A primeira é uma "Visão da Administração", na qual sistemas computacionais são capazes de gerenciar a si mesmos, e que é adotada pelos sistemas de escalonamento autonômico de aplicações. A segunda é uma "Visão de Engenharia de *Software*", na qual as funcionalidades autonômicas são padronizadas, proporcionando que os esforços sejam concentrados na melhoria e no desenvolvimento de funcionalidades.

Com o objetivo de melhor aproveitar os recursos computacionais disponíveis, diversos trabalhos na literatura, tais como [73], [64], [13], [12], [54] e [9], defendem a possibilidade das aplicações serem capazes de alocar as suas próprios tarefas, de maneira automática e em tempo de execução. Esta abordagem é denominada *Dynamic and Self Scheduling* (Escalonamento Dinâmico e Autonômico) e é utilizada de diferentes formas, sobre diferentes arquiteturas e com diferentes propósitos.

Os esforços descritos por Tang *et al.* [73] estão voltados para uma estratégia de escalonamento autonômico dos *loops* de aplicações paralelas e o impacto que a mudança da ordem de execução destes *loops* pode causar. Na prática, são realizadas variações em relação às filas de execução, com o propósito de analisar o impacto sobre o tempo de atraso de execução dos *loops* das aplicações paralelas. A estratégia proposta no trabalho, denominada *Shortest-delay Scheduling*, que consiste em agrupar *loops* de menor tempo de execução, se mostrou mais eficiente do que as demais estratégias existentes até então. Contudo, não é considerada a possibilidade de escalonar aplicações concorrentes, devido às limitações arquiteturais existentes na época da publicação do trabalho em questão.

Rudolph e Polychronopoulos [64], por sua vez, apresentam uma estratégia de escalonamento autonômico de aplicações que são executadas em sistemas computacionais de memória distribuída. Para isso, os autores discutem, implementam e confrontam os resultados obtidos a partir de estratégias de escalonamento utilizadas para sistemas de memória compartilhada em sistemas de memória distribuída. Como principal contribuição, os autores concluem que a estratégia denominada *Guided Self-Scheduling* apresentou os melhores resultados, devido à sua característica de alocar primeiro os maiores blocos de iterações, passando para os menores, até que todas os blocos de iterações tenham sido alocados. Embora uma estratégia semelhante seja usada por [9] em um contexto mais atual e abrangente, [64] limita-se apenas a esta análise comparativa restrita a duas ferramentas de *Benchmark*, e sem explorar a concorrência entre as aplicações.

Os trabalhos [73] e [64] apresentam abordagens de escalonamento sobre arquiteturas com memória compartilhada e distribuída, respectivamente. Os dois trabalhos estão entre os primeiros esforços que visam a melhoria dos resultados do escalonamento autonômico das aplicações, em vista dos recursos computacionais empregados.

Uma abordagem de escalonamento autonômico para *loops* paralelos não uniformes em recursos computacionais de memória distribuída é proposta por Liu e Saletore [46]. Os autores discutem métodos de distribuição de dados, utilizando duplicação parcial, o que se mostrou favorável para o escalonamento autonômico para as referidas aplicações, de

modo a permitir que o tamanho do problema cresça linearmente em relação à quantidade de processadores. A abordagem permite ainda que seja feita uma busca prévia dos dados a serem utilizados, devido à execução base que serve como comparação, reduzindo o *overhead* de escalonamento.

Embora a abordagem proposta pelos autores apresente uma melhora de 79% em relação à não adoção da mesma, apenas o escalonamento *offline* de aplicações é considerado. Por essa razão, é incerto garantir que os resultados sejam significativos também para o escalonamento *online*, uma vez que há forte dependência de execuções prévias das aplicações.

O *middleware EasyGrid* é apresentado por Boeres *et al.* [13, 12] e permite a execução de aplicações paralelas em ambientes distribuídos. Sua hierarquia de escalonamento é dividida em três níveis (global, local e de máquina), através de um sistema gerenciador de aplicações. Em [54], Nascimento introduz ao *EasyGrid* uma forma de permitir que as aplicações se tornem cientes do estado de um ambiente de Grade Computacional e que se comportem de modo autonômico, isto é, reagir por conta própria de acordo com eventos que ocorrem no sistema. Tais características possibilitam que políticas de escalonamento sejam melhor especificadas de acordo com as necessidades de cada aplicação. Além disso, surge também a possibilidade de simplificar a escalabilidade dos ambientes de Grade, através da distribuição do gerenciamento dos recursos entre as próprias aplicações.

Os benefícios proporcionados pelo *EasyGrid* são alcançados através da implementação de uma estratégia híbrida de escalonamento que, embora considerada pouco intrusiva, encontra-se embutida nas próprias aplicações. Isso permite que elas sejam capazes de criar ou eliminar processos dinamicamente, de modo a melhor explorar a disponibilidade dos recursos, além de realizar o balanceamento de carga de processamento entre os processos. O *EasyGrid* possui ainda um mecanismo de tolerância a falhas, graças à baixa granularidade dos processos e à capacidade de recriá-los, caso eles falhem. Por fim, é possível considerar que o *EasyGrid* realize o escalonamento de tarefas que competem por recursos, embora não seja feita a análise de interferência entre as tarefas. Essa análise possibilitaria escolher quais tarefas concorrentes resultariam em um escalonamento mais apropriado quanto às métricas de escalonamento de aplicações que competem por recursos computacionais.

A intrusão do *EasyGrid* nas aplicações, no entanto, pode ser interpretada como uma limitação, dado que existe a necessidade do acesso à codificação das aplicações para a inclusão da lógica do componente responsável por realizar o gerenciamento, denominado AMS (*Application Management System*). Essa necessidade pode dificultar, ou até inviabi-

lizar o uso desta estratégia em ambientes de larga escala, nos quais não há controle sobre as aplicações executadas no ambiente distribuído. No entanto, a estratégia de escalonamento autonômico adotada pelo *EasyGrid* pode servir como ponto de partida para a elaboração de uma estratégia semelhante, que seja ainda menos intrusiva para as aplicações. Além disso, outros trabalhos na literatura, tais como [54] e [24], utilizam o *EasyGrid* como base para o escalonamento autonômico de aplicações paralelas e de e-Ciência, respectivamente.

O termo e-Ciência (*e-Science*) refere-se às aplicações científicas que necessitam e fazem uso da computação de alto desempenho para que seja possível obter resultados corretos em tempo hábil. Essa necessidade se dá em função do grande volume de dados e da complexidade computacional da aplicação. Neste contexto, justifica-se o esforço de introduzir o conceito de autonomia nesse grupo de aplicações, uma vez que elas geralmente apresentam variações de comportamento quanto à utilização de recursos [57]. Neste contexto, Passos [24] apresenta uma adição ao *middleware EasyGrid* de modo a tornar autonômicas as aplicações da e-Ciência. Desse modo, a autora apresenta como suas principais contribuições a conclusão de que é possível executar aplicações da e-Ciência de forma benéfica em sistemas distribuídos e que as próprias aplicações são capazes de realizar o auto gerenciamento.

Embora apresente contribuições significativas para a e-Ciência, em especial sobre o modo de realizar o gerenciamento autonômico, é importante ressaltar que Passos [24] não introduz novas métricas para a análise do escalonamento das referidas aplicações, assim como não faz referência à competição entre aplicações além daquelas já contidas no *EasyGrid*. A relevância da análise e da criação de novas métricas e estratégias de escalonamento autonômico de aplicações científicas, que competem por recursos, se dá pelo fato de que os sistemas distribuídos permitem o uso compartilhado dos seus recursos por diferentes aplicações.

Uma proposta de escalonamento autonômico para arquiteturas multiprocessadas heterogêneas é apresentada por Belviranli *et al.* [9]. O objetivo é possibilitar a execução de *loops* de iterações nos sistemas com diferentes arquiteturas pertencentes ao sistema distribuído. Para alcançar o seu objetivo, foi criado um algoritmo que obtém informações sobre a capacidade de processamento de cada processador em sua primeira etapa de execução, denominada etapa de adaptação. Na etapa denominada finalização, a segunda de sua execução, o conhecimento obtido na etapa de adaptação é utilizado para escalonar o restante das aplicações do *job* de maneira autonômica.

A estratégia de escalonamento descrita por [9] foi projetada de forma a permitir que

o conhecimento acerca do comportamento da execução das aplicações, assim como dos ambientes heterogêneos nos quais ocorre o escalonamento, sejam obtidos sem a necessidade de uma execução prévia. No entanto, como principais fatores limitantes, estão: i) a necessidade de incluir a lógica que implementa a estratégia no código das aplicações, o que pode dificultar a sua ampla adoção; e ii) a pequena abrangência da análise realizada, dado que somente a influência do tamanho dos blocos de instruções foi levada em consideração como fator de variação do desempenho nos experimentos realizados.

2.4 Tipos de aplicações paralelas

Durante décadas e até nos dias atuais, programas de computadores foram e são escritos seguindo o modelo de programação sequencial, no qual a solução computacional de um problema é dividido em um conjunto discreto de instruções que, por sua vez, são executadas uma após a outra. As instruções são executadas em um único processador (ou núcleo de processamento), em instantes distintos de tempo [8]. No entanto, com o advento e a evolução das arquiteturas computacionais multitarefas e multi-núcleos, vislumbrou-se a possibilidade de explorar tais avanços, ao custo de adotar um paradigma de programação paralela, a fim de reduzir o tempo necessário para resolver problemas complexos.

O projeto de uma aplicação paralela, contudo, deve ser realizada de modo a considerar características anteriormente inexistentes. A divisão do trabalho da aplicação entre os recursos computacionais responsáveis por processá-la deve ser feita de modo a considerar o tipo de arquitetura computacional à disposição: podendo esta ser multitarefa ou multi-núcleos e com memória compartilhada ou distribuída. Além disso, as características da aplicação devem ser consideradas a fim de estabelecer corretamente a coordenação entre os processos paralelos e a forma de comunicação a ser realizada entre eles. De acordo com as características de coordenação entre os processos, a classificação dos modelos de aplicações paralelas apresenta-se da seguinte forma:

- Aplicações *bag of tasks*: são aplicações paralelas compostas por tarefas sequenciais e independentes, ou seja, que não possuem dependência e não realizam comunicação entre si [69]. Os dados de entrada são arquivos que servem para uma ou mais tarefas e cada tarefa gera o seu próprio conjunto de arquivos de dados de saída. Exemplos de aplicações *bag of tasks* incluem as simulações Monte Carlo[1], aplicações

[1] O método de Monte Carlo é uma ferramenta utilizada em diversas áreas para simular o comportamento complexo de problemas científicos e industriais [24].

de manipulação de imagens e aplicações de mineração de dados. As aplicações *bag of tasks* também são conhecidas na literatura como *parameter-sweep applications* [69] *apud* [17].

- Aplicações mestre/trabalhador: segundo Silva e Buyya [70], as aplicações paralelas mestre/trabalhador são aquelas compostas por dois tipos de processos com funções específicas. O processo mestre é responsável por decompor o domínio do problema em pequenas tarefas e distribuí-las aos processos trabalhadores. Além disso, o processo mestre tem a função de receber os resultados parciais e consolidá-los de modo a apresentar o resultado final do processamento. Os processos trabalhadores recebem do processo mestre as especificações dos dados a serem processados, realizam o processamento e enviam seus resultados parciais de volta ao processo mestre. Neste modelo, é comum que a comunicação ocorra somente entre o processo mestre e os trabalhadores e vice-versa.

- *Workflows* científicos: por definição, um *workflow* científico é composto por um conjunto de aplicações independentes, mas que possuem relações de precedência entre si, ou seja, os dados de saída de uma aplicação podem servir como dados de entrada para uma ou mais aplicações do mesmo *workflow*. Sua adoção permite a construção e a execução de experimentos científicos complexos em diversas áreas como a Biologia, Engenharias, Manufatura, Processos de Negócios, entre outras [23, 61]. A natureza do paralelismo de tais aplicações está em dois aspectos: o primeiro é que aplicações que tenham suas precedências satisfeitas podem ser executadas simultaneamente, caso exista a disponibilidade de recursos computacionais. O segundo aspecto está associado às características individuais de cada aplicação do *workflow*, que podem ser individualmente consideradas como aplicações paralelas convencionais, sejam elas *bag of tasks* ou mestre/trabalhador [71]. Em virtude das dependências existentes entre as aplicações, torna-se complexa a gerência do *workflow*, tarefa essa realizada por ferramentas gerenciadoras especializadas [21].

A classificação de aplicações paralelas, no entanto, não se dá apenas de acordo com o modelo de coordenação entre os processos. Elas podem ser classificadas também de acordo com o dinamismo de criação dos processos paralelos. No que se refere a esse critério, Feitelson e Rudolph [31] as classificam como rígidas, moldáveis, maleáveis, evolutivas e adaptativas, com as seguintes definições:

- Rígidas: são aplicações paralelas que possuem um conjunto fixo de processos defi-

nido no instante da execução. Não há qualquer forma de interação da aplicação com agentes externos, de modo que não é possível realizar qualquer variação da quantidade de processos em execução ou da quantidade de recursos de processamento associados às tarefas.

- Moldáveis: aplicações paralelas que admitem execução para diferentes quantidades de processos paralelos. Essa característica é viável desde que o programa seja escrito de forma a dividir os dados de entrada entre os processos participantes da execução paralela. Contudo, a quantidade de processos de uma aplicação moldável não pode ser alterada durante a sua execução, o que impede que a mesma se adapte em função da carga de trabalho ou da disponibilidade de recursos.

- Maleáveis: são aplicações capazes de se adaptar a mudanças da quantidade de processos em tempo de execução. Apesar desta característica, estas aplicações não são consideradas autonômicas [31], uma vez que a decisão de aumentar ou reduzir a quantidade de processos fica a critério de um agente externo, como um escalonador, por exemplo.

- Evolutivas: as aplicações paralelas evolutivas são aquelas que apresentam a capacidade de auto gerenciamento, de acordo com a variação da carga de trabalho de suas tarefas e da disponibilidade dos recursos computacionais, e sem a intervenção de um agente externo. Para que isso seja possível, estas aplicações devem ser desenvolvidas de modo a se tornarem cientes do estado dos recursos e das demais aplicações que competem por recursos, além de terem autonomia para criar ou eliminar processos.

- Adaptativas: são aplicações paralelas que podem apresentar comportamento maleável ou evolutivo, agindo de modo autonômico ou com a influência de um agente externo.

2.5 Competição entre aplicações

Certos domínios do mundo real, como serviços de socorro em situações de desastres, apresentam uma natureza dinâmica, de modo a demandar uma alocação igualmente dinâmica dos recursos disponíveis para que sejam capazes de atender às requisições feitas a eles. Adicionado a isso, está o fato de que os recursos são finitos e, frequentemente, estão distribuídos entre diversos agentes que trabalham em contextos diferentes. Este cenário é introduzido por Wang *et al.* [78] como motivação para um trabalho que apresenta

uma abordagem baseada em leilões para que serviços diferentes sejam capazes de coordenar a utilização dos recursos pelos quais eles competem. Os autores mostram que o uso do agente desenvolvido por eles, que aplica técnicas baseada em leilões, auxilia na coordenação de tarefas distribuídas que utilizam recursos escassos compartilhados. Embora a solução proposta pelos autores seja genérica, focada em operações de resgate, para qualquer situação de compartilhamento de recursos e tarefas que competem entre si, os resultados experimentais sustentam a aplicabilidade para o problema de escalonamento de tarefas concorrentes. Para o problema analisado, foi reportado o aumento da utilização dos recursos compartilhados em, aproximadamente, 60%, quando comparada com a alocação que não adota a estratégia desenvolvida.

A execução de aplicações que competem por recursos compartilhados em ambientes de HPC é contemplada também no trabalho de Oliveira [55]. Nesse trabalho são consideras aplicações paralelas do tipo *bag of tasks* que possuem um prazo. O prazo é um marco de tempo que pode ser flexibilizado, mas de modo a não extrapolar o *deadline* da aplicação. A estratégia definida pelo autor segue o conceito de uma sociedade altruísta, na qual as aplicações são cientes do estado do sistema e das aplicações concorrentes. De posse desse conhecimento, as aplicações são capazes de ceder ou intensificar o uso de recursos com os propósitos de que todas as aplicações finalizem suas execuções dentro do prazo especificado e de que a taxa de ocupação dos recursos seja maximizada.

O trabalho de Oliveira [55] introduz ainda um diagrama de estados de comportamento adotado pelas aplicações durantes suas execuções. Tais estados estados são utilizados com a finalidade de permitir que o compartilhamento dos recursos de processamento seja feito de maneira a garantir que as aplicações com menor prazo tenham maior prioridade sobre a utilização dos recursos. No entanto, os resultados apresentados pelo autor mostram uma subalocação dos recursos em relação à quantidade de tarefas executadas por cada aplicação, causada por um elevado tempo de reação das aplicações para iniciar a execução de suas tarefas. Ademais, o comportamento altruísta das aplicações é experimentado apenas para duas aplicações, o que, para aplicações cujas cargas de processamento não sejam elevadas, não garante a maximização de ocupação da infraestrutura.

Oliveira [57] apresenta uma proposta de alocação e migração de máquinas virtuais em ambientes de nuvem e de Computação Massivamente Paralela e Distribuída (CMPD), com base no monitoramento e no aprendizado dos perfis de consumo de recursos pelas aplicações em execução. A estratégia adotada pelo autor considera que as máquinas virtuais competem pelos recursos das máquinas reais e aplica os conceitos de interferência

e de afinidade entre aplicações, com o objetivo de reduzir o impacto causado pela execução concorrente, a partir de informações obtidas com o monitoramento prévio das aplicações. O monitoramento do perfil das aplicações permite ainda que ações sejam executadas durante a execução da aplicação virtualizada, de modo a manter a execução no mesmo servidor real ou migrar para outro, de acordo com a análise da afinidade entre as aplicações e do tempo necessário para realizar a migração.

Os experimentos conduzidos por Oliveira [57] abrangeram três tipos de alocação: uma FCFS (*First-Come First-Served*), uma *Round-Robin* e uma denominada *Affinity*, que é proposta por ele. Os resultados mostram que a estratégia *Affinity* proporcionou uma redução aproximada do *makespan* de 33% sobre a estratégia *Round-Robin* e de 45% sobre a FCFS. Com isso, o autor valida as hipóteses de que ações cientes do perfil das aplicação e das afinidades entre elas impactam sobre o resultado do escalonamento em situações nas quais existe a competição por recursos compartilhados. Contudo, é necessário enfatizar que as conclusões do autor são baseadas na métrica de *makespan*, o que direciona a estratégia adotada apenas para situações de escalonamento *offline*, isto é, aquelas em que todas as aplicações são conhecidas no momento em que o escalonamento é iniciado.

Allesandro *et al.* [2] abordam o problema do escalonamento de aplicações submetidas por duas entidades distintas, denominadas agentes, para serem executadas de modo concorrente em um conjunto compartilhado de recursos de processamento. Os autores afirmam que, para esse problema, cada agente deve minimizar/otimizar uma determinada função objetivo, que varia de acordo com o tempo necessário para executar cada aplicação. A principal premissa adotada pelos autores é que o escalonamento considerado "melhor" para um determinado objetivo do primeiro agente necessariamente irá resultar em um escalonamento "pior" para o mesmo objetivo do segundo agente. Tal premissa, denominada escalonamento não-dominante ou Pareto-ótimo, é provada como sendo falsa pelos próprios autores e por outros na literatura, como [45] e [82]. Na prática, a conclusão de que duas aplicações podem ser escalonadas de modo satisfatório para ambas é a maior contribuição de [2], embora os autores não deixem isso claro, além de não apresentarem uma metodologia experimental para comprovar as próprias hipóteses. por fim, o trabalho considera apenas o compartilhamento de recursos de processamento.

2.6 Síntese do capítulo

Os trabalhos correlatos apresentados neste capítulo foram discutidos devido ao fato do trabalho desenvolvido nesta tese de doutorado considerar as aplicações científicas paralelas que executam sobre um ambiente computacional compartilhado. A estratégia de escalonamento proposta envolve o uso de aplicações autonômicas adaptativas, cujas tomadas de decisão são realizadas por um *Application Management System* (AMS). Tais decisões são ponderadas de acordo com métricas associadas às aplicações (prazo de término e cargas de processamento e de comunicação) e à infraestrutura (taxa de utilização dos recursos computacionais).

Capítulo 3

Uma nova abordagem para resolver um antigo problema

No Capítulo 2 foram apresentados trabalhos sobre áreas correlatas ao problema de escalonamento de aplicações que compartilham recursos computacionais de processamento. Foram apresentadas as principais contribuições de cada trabalho e a sua relação com esse problema. Este capítulo apresenta uma pesquisa mais direcionada e aprofundada a respeito de abordagens que possuem o objetivo de resolver o problema de escalonamento de aplicações. É apresentado um breve histórico sobre o assunto, começando pelo problema clássico *Job-shop Scheduling*, passando pelo problema de escalonamento de aplicações em ambientes compartilhados, para chegar até a forma como a alocação de recursos é feita atualmente, com a finalidade de atender às demandas por alta capacidade de processamento das aplicações de e-Ciência. Além disso, são relatados os principais problemas enfrentados por estratégias atuais e amplamente utilizadas, que servem como motivação para apresentar a nova abordagem de escalonamento proposta nesta tese.

3.1 A evolução do escalonamento de aplicações

O problema de escalonamento consiste na tomada de decisões necessárias para atribuir recursos para executar um conjunto de tarefas, dentro de um determinado período de tempo, com o objetivo de otimizar um ou mais objetivos. Soluções para este problema são adotadas regularmente em áreas que variam desde a manufatura de produtos até o provimento de serviços [58]. O *Job-shop Scheduling* é um dos problemas mais conhecidos e estudados. Nele, um conjunto de *jobs* deve ser completado no menor tempo possível [19].

Em sua definição formal, o *Job-shop Scheduling* considera um conjunto de m máquinas

que deve ser utilizado para escalonar e executar *n jobs* diferentes. Cada *job* é um conjunto de operações que devem ser executadas em uma sequência específica [18]. Para executar, cada operação requer uma máquina específica e uma quantidade fixa de processamento, além de não poder ser interrompida. Outras restrições do *Job-shop Scheduling* determinam que: uma operação é executada em apenas uma máquina; não há relações de precedência entre operações que pertencem a *jobs* diferentes; cada máquina pode processar apenas uma operação por vez; e não há a definição de tempo limite para a execução de um *job*. O objetivo do *Job-shop Scheduling* é determinar o mapeamento da execução das operações nas máquinas requeridas por elas, de modo a minimizar o tempo total necessário (*makespan*) para executar todas as operações do *job*. Uma característica comum a vários problemas de escalonamento, entre eles o *Job-shop Scheduling*, é que eles fazem parte da categoria de problemas NP-completo, onde não existe um algoritmo conhecido capaz de gerar escalonamentos ótimos em tempo polinomial [36].

Como uma extensão do problema *Job-shop Scheduling* clássico, o problema *Flexible Job-shop Scheduling* (FJSP) permite que uma operação seja processada em qualquer máquina ou em um determinado subconjunto de máquinas. O problema consiste em atribuir e ordenar as operações nas máquinas nas quais elas serão executadas de maneira a minimizar o *makespan* do *job* [65]. Uma vez que o problema FJSP assume a execução de uma operação em múltiplas máquinas, ele se torna uma alternativa viável para lidar com o escalonamento de *jobs* em sistemas computacionais compostos por múltiplas máquinas interconectadas.

Com o surgimento dos sistemas computacionais de alto desempenho (HPC) na década de 1980, as aplicações passaram a ser executadas de forma diferente. Nesses sistemas, as aplicações executam como *batch jobs*, isto é, as requisições pelos recursos necessários para executá-las devem ser atendidas dentro de um limite de tempo [3]. Além disso, múltiplos *batch jobs* podem compartilhar o uso de um mesmo conjunto de computadores, o que torna possível a execução de várias aplicações simultaneamente em um mesmo ambiente. Assim, surgiu uma nova vertente do problema de escalonamento, na qual precisa-se atribuir um conjunto de recursos computacionais fixo para executar cada aplicação (ou *job*), de modo a minimizar os seus tempos de execução e maximizar a utilização dos recursos.

Os principais sistemas escalonadores utilizados em ambientes de HPC abordam o problema de alocação das aplicações para executar utilizando os recursos disponíveis, além de adotarem políticas de fila. Uma vez submetidas, as aplicações são inseridas no fim de uma fila e aguardam até que o escalonador as coloquem em execução, de acordo com

as políticas de fila adotadas. Por exemplo, a política *First Come First Served* (FCFS) executa as aplicações de acordo com a ordem de chegada. Por outro lado, a política *Backfilling* permite que a ordem seja alterada e que uma aplicação tenha a sua execução antecipada, caso os recursos disponíveis não sejam suficientes para executar as aplicações que chegaram na fila antes dela. Além disso, os modelos de alocação exclusiva e não exclusiva de servidores podem ser adotados, seja para garantir que as aplicações utilizem os recursos do servidor de forma dedicada durante as suas execuções, seja para habilitar o compartilhamento do máquina. A maioria dos sistemas escalonadores amplamente utilizados, tais como o *Simple Linux Utility for Resource Management* (SLURM) [42] e o Torque PBS [72], permitem a adoção de políticas de fila e modelos de alocação de recursos diferentes para escalonar as aplicações.

Embora os sistemas escalonadores tradicionais permitam o compartilhamento de servidores quando um modelo de alocação não exclusiva é adotado, recursos computacionais como os núcleos de processamento são alocados exclusivamente para uma aplicação se eles não estiverem atribuídos a outras. A Figura 3.1 ilustra um exemplo de escalonamento de duas aplicações em uma única máquina que possui 8 núcleos disponíveis. A primeira requer 4 núcleos para executar de forma ideal, enquanto a segunda pode executar apenas se 5 núcleos estiverem disponíveis. Embora o modelo de alocação não exclusiva permita que a máquina seja compartilhada, isso não se estende aos seus núcleos. Logo, a segunda aplicação precisa aguardar até que a execução da primeira seja concluída. Esta abordagem pode acarretar em um tempo de espera na fila elevado para a aplicação APP2, enquanto o compartilhamento de um dos núcleos talvez não causasse prejuízos para a execução de nenhuma das aplicações. Esta abordagem pode levar ainda à subtilização dos recursos, uma vez que um subconjunto de núcleos permanece ocioso durante todo o tempo.

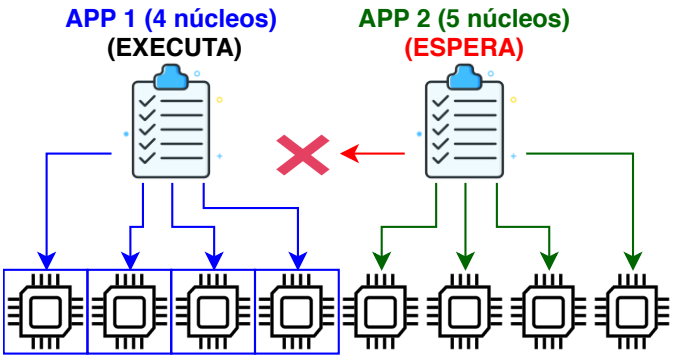


Figura 3.1: Exemplo típico do escalonamento de aplicações que compartilham uma mesma máquina realizado por escalonadores tradicionais

A execução das aplicações ilustrada na Figura 3.1 considera aplicações rígidas, isto é,

aquelas que podem ser executadas apenas com uma quantidade específica de processos. Porém, ao considerar aplicações moldáveis, a aplicação APP2 poderia também ser executada com 4 processos. Embora isso pudesse causar o aumento do seu tempo de execução, ela poderia ser iniciada imediatamente, eliminando assim o tempo de espera na fila. Por fim, as aplicações poderiam executar simultaneamente compartilhando todos um conjunto de núcleos de processamento. Tal abordagem também eliminaria o tempo de espera em fila, além de aumentar a vazão dos núcleos utilizados. Em contrapartida, a competição por recursos leva ao aumento dos tempos de execução individuais de ambas as aplicações.

Os sistemas escalonadores SLURM e Torque PBS também enfrentam problemas relacionados à escalabilidade, quando precisam gerenciar recursos de sistemas distribuídos de larga escala. Isso ocorre porque eles utilizam um gerente (*broker*) centralizado para realizar o gerenciamento dos recursos e o escalonamento das aplicações [60]. Além do mais, as aplicações que são gerenciadas por esses sistemas geralmente possuem diferentes requisitos de recursos para executar [3]. Portanto, estratégias tradicionais de escalonamento tornam-se ineficientes ao escalonar aplicações que compartilham os recursos de um sistema distribuído [75].

O compartilhamento de recursos é uma medida necessária para permitir que mais aplicações sejam executadas simultaneamente. Somado a isso, vantagens de modelos como a computação em nuvem que oferecem recursos sob demanda, elasticidade e customização tornam-se uma alternativa economicamente atrativa para atender aos requisitos de alguns usuários e de suas aplicações [37]. É de amplo conhecimento que alguns objetivos da computação em nuvem e da computação de alto desempenho sejam antagônicos. A primeira prioriza a redução de custos e o aumento da taxa de utilização da infraestrutura. Já a segunda tem o objetivo de aumentar o desempenho das aplicações, por intermédio da redução dos seus tempos de execução. No entanto, talvez a razão pela qual os objetivos se oponham seja porque os escalonadores usados em ambientes de HPC desenvolvidos até então não tenham sido projetados para lidar com características diferentes daquelas inerentes a eles. Quando sistemas distribuídos e compartilhados são levados em consideração, fica claro que a evolução das estratégias de escalonamento não acompanhou o avanço das arquiteturas computacionais.

Os sistemas escalonadores existentes tentam adaptar suas estratégias para lidar com diferentes arquiteturas computacionais e com as características intrínsecas a elas, ao invés de pensar de forma mais ousada para criar uma nova solução. Com o objetivo de propor uma nova solução, esta tese apresenta uma estratégia descentralizada de escalo-

namento de aplicações autonômicas, que tira proveito do compartilhamento dos recursos de processamento. A estratégia proposta é capaz de escalonar um mesmo conjunto de aplicações de formas diferentes, por meio do ajuste de apenas uma parâmetro de execução de cada aplicação. Os diferentes escalonamentos resultantes buscam beneficiar dois grupos de interesse distintos: os usuários das aplicações, ao reduzir o tempo de espera na fila e, consequentemente, o tempo de resposta para obter os resultados das aplicações; e os gestores dos recursos, ao permitir que um maior número de usuários e aplicações sejam atendidos simultaneamente.

3.2 O EasyGrid AMS

O EasyGrid AMS [54, 25] é um sistema gerenciador de aplicações paralelas em ambientes computacionais distribuídos. As aplicações são definidas na forma de Grafos Acíclicos Direcionados (GADs) que expressam as relações de precedência entre as suas tarefas. O EasyGrid AMS é embutido nas aplicações paralelas MPI e implementa a criação dinâmica de tarefas segundo o modelo *1PTask* (um processo para cada tarefa) e permite o auto gerenciamento de suas execuções.

O EasyGrid AMS adota uma arquitetura hierárquica em três níveis, de modo a realizar o gerenciamento de processos. Cada um dos níveis é composto por cinco camadas com propósitos específicos em relação das tarefas: gerenciamento de processos, monitoramento da aplicação, escalonamento dinâmico, tolerância a falhas e maleabilidade. Cada aplicação MPI possui o seu próprio sistema de gerenciamento em três níveis hierárquicos. O *Global Manager* (GM), no nível mais alto da hierarquia, é responsável por supervisionar os *sites* do ambiente distribuído nos quais a aplicação está executando. Também é responsabilidade de um GM realizar o escalonamento dinâmico entre os seus *sites* com a finalidade de balancear a carga de trabalho entre eles. Cada um dos *sites* possui um *Site Manager* (SM), que é encarregado da alocação dos processos da aplicação nos seus recursos disponíveis. De forma análoga ao GM, um SM é responsável por balancear a carga de trabalho entre os *hosts* pertencentes ao *site* que ele gerencia. Em cada um dos *hosts* do *site* existe um *Host Manager* (HM). Cada HM é responsável por criar e executar os processos MPI de uma aplicação, por monitorar o consumo de recursos do *host* pela aplicação e por colocar em práticas as políticas de tolerância a falhas. Cada um dos três níveis da hierarquia possui um escalonador dinâmico para a gerência da execução dos processos na arquitetura: um *Global Dynamic Scheduler* (GDS) no nível global, um *Site Dynamic Scheduler* (SDS) no nível de um *site* e um *Host Dynamic Scheduler* (HDS) no

nível do *host*. A Figura 3.2 ilustra a arquitetura em três níveis do EasyGrid AMS.

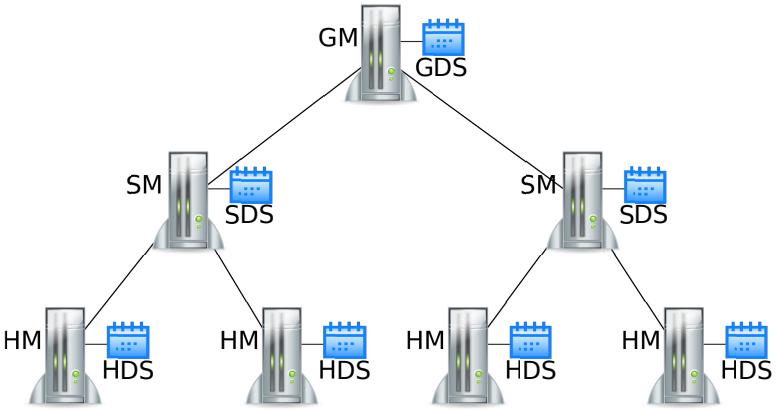


Figura 3.2: Arquitetura hierárquica do EasyGrid AMS

O EasyGrid AMS foi o sistema adotado para gerenciar as aplicações e torná-las autonômicas. O EasyGrid é uma metodologia eficiente e robusta para executar programas *Message Passing Interface* (MPI) em ambientes computacionais distribuídos. A integração de um AMS a cada aplicação permite que os recursos computacionais sejam utilizados de forma eficiente [13]. Além disso, cada AMS é responsável por atender aos requisitos da aplicação, conforme a disponibilidade dos recursos. O EasyGrid AMS também permite que tarefas previamente alocadas para executar em um determinado *host* sejam realocadas para executar em outro, com a finalidade de manter a carga de trabalho balanceada entre os recursos utilizados. O EasyGrid AMS exerce um papel importante na estratégia de escalonamento proposta, uma vez que é por meio dele que as aplicações são capazes de realizar o autoescalonamento de suas tarefas e de adquirirem conhecimento sobre a utilização do ambiente.

O trabalho proposto nesta tese tira proveito das características do EasyGrid AMS relacionadas ao escalonamento dinâmico das aplicações e do balanceamento de carga entre os *hosts* que realizam o processamento. Além disso, o conceito de sociedade de aplicações altruístas, introduzido por Bueno em [14] e aperfeiçoado por Oliveira em [55], também foi utilizado. No entanto, o foco desta tese está voltado para permitir que seja possível alcançar diferentes resultados de escalonamento por meio do ajuste de apenas um parâmetro de cada aplicação. Este foco foi responsável por mudanças na implementação da lógica do módulo HM do EasyGrid AMS, referentes ao monitoramento das aplicações e dos recursos, além de uma análise mais estrita sobre as métricas utilizadas e a forma como elas devem ser calculadas. Tais mudanças são propagadas para os demais níveis da hierarquia do EasyGrid AMS, uma vez que as decisões tomadas sobre o escalonamento de tarefas em um *host* afetam o escalonamento de tarefas em todos os *hosts* utilizados

pela aplicação. Por fim, também foram realizadas análises mais aprofundadas sobre os resultados obtidos, agora, considerando o escalonamento de aplicações que compartilham recursos de processamento.

3.3 Definição do problema e uma proposta de solução

O problema abordado nesta tese de doutorado é o escalonamento de aplicações que compartilham os recursos de processamento do ambiente computacional. Esse problema também é referenciado por autores na literatura como escalonamento competitivo (*competing scheduling*) [62, 26]. O problema também envolve escalonar as aplicações de maneira a aproveitar os períodos em que os recursos do ambiente encontram-se ociosos, com o objetivo de maximizar a sua utilização, pois as aplicações não utilizam os recursos em sua totalidade durante todo o período em que elas se encontram em execução [59]. Isso se deve a situações nas quais as suas tarefas realizam trocas de mensagens, ou por encontrarem-se em estado de espera, dentre outras razões que as impossibilitem de utilizar os recursos computacionais em questão.

Sistemas de escalonamento utilizados, como o SLURM [42] e o Torque PBS [72] iniciam a execução das aplicações somente se todos os seus requisitos de recursos forem satisfeitos. Conforme já discutido, essa abordagem pode levar a longos períodos de espera, além de causar a subutilização dos recursos. Por outro lado, o uso de recursos compartilhados não é considerado como uma alternativa eficiente em ambientes de HPC, principalmente por causar o aumento dos tempos de execução das aplicações [44]. Assim, as soluções tradicionais priorizam a redução dos tempos de execução e do aumento do tempo de espera em fila, por entenderem que isso leve à redução do tempo necessário para executar todas as aplicações (*makespan*), mesmo que tal abordagem leve também à subutilização dos recursos.

Com base nisto, a estratégia proposta nesta tese aborda o problema de escalonar aplicações que compartilham recursos de processamento. O principal objetivo da estratégia é maximizar o uso dos recursos e, em consequência disso, reduzir o *makespan* do *workload*, isto é, do conjunto de aplicações submetidas, mesmo que isto cause o aumento dos tempos de execução individuais das aplicações. Além disso, busca-se eliminar por completo o tempo de espera em filas de execução e, com isso, reduzir o tempo médio de resposta das aplicações (tempo médio de *turnaround*). Os objetivos são relevantes para os usuários, que podem obter resultados parciais e totais em um tempo reduzido, como também para

os provedores de serviço, ao ter maximizados tanto o uso dos seus recursos como a sua capacidade de atendimento.

Antes de introduzir a solução proposta para resolver o problema de escalonar aplicações que compartilham os recursos de processamento de um ambiente, alguns termos que são utilizados durante a tese devem ser formalmente definidos. Um *workload* $W = \{J_1, J_2, \ldots, J_n\}$ é um conjunto de n aplicações que devem ser escalonadas, executadas e que podem ter diferentes tempos de submissão. Cada aplicação (ou *job*) $J_i \in W$ é um conjunto de k tarefas, tal que $J_i = \{t_1, t_2, \ldots, t_k\}$. Cada tarefa $t_j \in J_i$ possui um custo de execução dado por $\varepsilon(t_j)$. O ambiente computacional compartilhado é definido como um conjunto $H = \{h_1, h_2, \ldots, h_m\}$ de m *hosts* heterogêneos, no qual cada *host* h_w possui uma quantidade de núcleos de processamento c_w. Com isso, torna-se possível escalonar as n aplicações do *workload* W nos m *hosts* do ambiente computacional H, de modo que as tarefas das aplicações possam compartilhar os *hosts* e seus núcleos de processamento disponíveis em H. Isso deve ser feito com a finalidade de reduzir o *makespan* e o tempo médio de *turnaround* do *workload* e aumentar a taxa de utilização dos recursos de processamento da infraestrutura.

A solução proposta aborda o problema por meio de uma estratégia de escalonamento descentralizado das aplicações, na qual cada uma delas é responsável por criar e executar as próprias tarefas. Logo, cada aplicação autônoma possui um AMS próprio, responsável por obter periodicamente as informações necessárias para realizar o escalonamento e por executar as suas tarefas. Essas informações envolvem métricas como o **Tempo Estimado de Término (TET)** e a **Tempo Restante (TR)** de uma aplicação, além do **Percentual Total de Consumo dos Recursos de Processamento (PTR)** utilizados por ela, cujas definições e cálculos serão apresentadas ainda neste capítulo. Uma vez de posse dessas informações, cada aplicação toma as ações necessárias para executar as suas tarefas nos recursos compartilhados, mas de forma a evitar comprometer o seu próprio desempenho e o desempenho das demais aplicações que compartilham os recursos de processamento.

A estratégia proposta foi estruturada ainda de forma a admitir que um parâmetro de execução seja fornecido, ou pelo usuário de cada aplicação J_i que compõe o *workload*, ou por um sistema capaz de determiná-lo de forma automática. Este parâmetro, denominado **Tempo Alvo (TA)** e denotado por $TA(J_i)$, representa uma métrica indireta de prioridade que é adotada na implementação da estratégia. Assim, quanto menor for a diferença entre o **Tempo Estimado de Término** da aplicação e o seu **Tempo Alvo**, mais urgente se torna a execução das tarefas desta aplicação, o que faz com que o seu AMS aumente

a quantidade de tarefas executadas. Uma vez que aplicações com diferentes valores de Tempo Alvo executam de maneira a compartilhar os recursos, suas ações de escalonamento são realizadas com a finalidade de concluir suas execuções no menor tempo praticamente possível, porém, sempre tentando minimizar o impacto sobre a execução de aplicações que se encontrem mais atrasadas.

É importante ressaltar que a estratégia de escalonamento proposta não foi elaborada com a finalidade de reduzir o tempo de execução individual das aplicações escalonadas, e sim analisar os tempos associados ao *workload* como um todo. Ao mesmo tempo, busca-se também alcançar o melhor aproveitamento dos recursos de processamento da infraestrutura. Assim, a estratégia proposta foi idealizada de forma a permitir que diferentes escalonamentos sejam realizados a partir de um mesmo *workload*. Os diferentes escalonamentos são realizados por meio do ajuste de um único parâmetro de cada aplicação, o que permite que diferentes objetivos, como a redução do makespan do *workload* ou a redução do tempo médio de *turnaround* de cada aplicação, sejam alcançados. A possibilidade de estabelecer diferentes objetivos de escalonamento para um mesmo *workload* de forma simples é uma característica que pode beneficiar os usuários em diferentes situações: seja ao executar todas as aplicações, seja ao obter o resultado de uma aplicação específica mais rapidamente. Por fim, os provedores de serviço também são beneficiados ao terem os seu recursos melhor aproveitados, o que pode levar à redução do consumo energético.

3.3.1 Métricas de avaliação da estratégia proposta

Com a finalidade de avaliar os resultados provenientes da adoção da estratégia de escalonamento proposta, duas métricas foram definidas: o *makespan* e o tempo médio de *turnaround* do *workload*. As definições destas duas métricas são necessárias em razão de suas utilizações considerarem características diferentes daquelas definidas na literatura e amplamente utilizadas. No contexto desta tese, ambas as métricas estão relacionadas ao *workload* e não a cada aplicação individualmente. Além disso, o tempo médio de *turnaround* desconsidera o tempo de espera na fila, uma vez que o uso da estratégia proposta torna desnecessário aguardar pela disponibilidade de um recurso para alocá-lo de forma exclusiva a uma única aplicação. As duas métricas estão definidas da seguinte forma:

- ***Makespan*** do *workload*: o *makespan* do *workload* W que utiliza um subconjunto de *hosts* $hu \subset H$ é definido pelo tempo de término da última aplicação $J_i \in W$ a

concluir a sua execução, tal que:

$$makespan(W, hu) = max(TT(J_i)),\ i = \{1, \ldots, n\} \tag{3.1}$$

onde $TT(J_i)$ representa o tempo de término de uma aplicação J_i e n é a quantidade de aplicações do *workload* W.

- **Tempo Médio de *Turnaround* (TMT)** de um *workload*: a estratégia de escalonamento proposta permite que as aplicações compartilhem os recursos. Logo, não há a necessidade de aguardar para utilizá-los de forma exclusiva. O cálculo do tempo de *turnaround* de uma aplicação J_i adotado nesta tese desconsidera o seu tempo de espera em fila, uma vez que ele é igual a zero. Assim, o tempo médio de *turnaround* $TMT(W, hu)$ de um *workload* W que utiliza um subconjunto de *hosts* $hu \subset H$ é dado por:

$$TMT(W, hu) = \frac{\sum_{i=1}^{n} TE(J_i)}{n}, \forall\ J_i \in W \tag{3.2}$$

onde $TE(J_i)$ é o tempo de execução de uma aplicação J_i e n é a quantidade de aplicações do *workload* W.

3.3.2 Métricas calculadas

A estratégia de escalonamento proposta é fundamentada na capacidade de cada aplicação escalonar dinamicamente as próprias tarefas. Além disso, as decisões de escalonamento tomadas por uma aplicação podem impactar sobre o desempenho das aplicações que compartilham os recursos de processamento. Em sua forma natural, o EasyGrid AMS não monitora informações sobre o uso dos recursos por processos de outras aplicações, pelo menos não de maneira a considerá-las ao realizar o escalonamento, embora ele realize a coleta de informações acerca da disponibilidade dos recursos. Assim, algumas modificações foram feitas no EasyGrid AMS para refinar o monitoramento das informações ou para obter aquelas que não eram monitoradas previamente, seja a respeito das próprias aplicações, seja a respeito dos recursos que elas utilizam. São elas:

- **Tempo Restante (TR)** das tarefas de uma aplicação alocadas em um determinado *host*: baseia-se no tempo estimado de execução das tarefas de uma aplicação $J_i \in W$ a atribuídas a um *host* $h_j \in H$. Seja $nt \subset J_i$ o conjunto de l tarefas ainda não executadas alocadas a h_j, a quantidade de trabalho restante de J_i em h_j é obtida

por:

$$TR(J_i, h_j) = \frac{\sum_{i=1}^{l}(\varepsilon(t_i))}{c_j}, \forall\ t_i \in nt_i \tag{3.3}$$

onde $\varepsilon(t_i)$ representa o custo de processamento de cada tarefa t_i e $nt_i \subset J_i$ é o conjunto de l tarefas ainda não executadas alocadas a h_j.

É importante relembrar que o EasyGrid AMS permite que as aplicações sejam executadas em ambientes computacionais distribuídos, cujos recursos podem possuir características heterogêneas. Por essa razão, a quantidade de núcleos (c_j) em cada um dos *hosts* pode ser diferente. A possibilidade de existirem *hosts* heterogêneos quanto à quantidade de núcleos é levada em consideração para o cálculo do tempo restante de execução das tarefas de uma aplicação.

- **Tempo Estimado de Término (TET)** das tarefas de uma aplicação alocadas em um determinado *host*: o AMS de uma aplicação coordena a execução das suas tarefas em cada um dos *hosts* utilizados por ela. O $TET(J_i, h_j)$ é o tempo estimado de término de uma aplicação $J_i \in W$ em um *host* $h_j \in H$, tal que:

$$TET(J_i, h_j) = tm(J_i, h_j) + TR(J_i, h_j) \tag{3.4}$$

onde $tm(J_i, h_j)$ é o instante de tempo no qual a medição foi realizada pelo AMS da aplicação J_i no *host* h_j e $TR(J_i, h_j)$ é a sua quantidade de trabalho restante naquele *host*.

É importante ressaltar que uma aplicação pode executar simultaneamente em um ou mais *hosts*, com a coordenação da execução e o balanceamento da carga de processamento da aplicação sendo feitos pelo seu AMS. Sendo assim, é possível que existam tempos estimados de término diferentes para uma mesma aplicação em cada um dos *hosts* utilizados por ela, o que reflete em diferentes estados em cada *host*.

- **Percentual Total de Consumo dos Recursos de Processamento (PTR)**: é o percentual total de utilização dos recursos de processamento gerado pelo conjunto de processos p_j que se encontram em execução no *host* $h_j \in H$. O valor de $PTR(h_j)$ é recalculado a cada evento de monitoramento. O valor de $PTR(h_j)$ é resultado da razão entre o somatório do consumo de CPU do conjunto de processos p_j todos os p processos em execução naquele *host* no instante em que o evento de monitoramento é realizado, e a sua quantidade de núcleos de processamento c_j, tal que:

$$PTR(h_j) = \frac{\sum_{t \in p_j} pu(t)}{c_j} \tag{3.5}$$

onde $pu(t)$ representa o consumo de CPU de cada processo t do conjunto p_j em execução no *host* h_j. O cálculo de $PTR(h_j)$ é, na prática, um valor médio de utilização dos núcleos de processamento existentes naquele *host*.

- **Percentual de Recursos Consumidos por uma Aplicação (PRA)**: é o total de processamento consumido pelo subconjunto $pt_i \subset J_i$ de tarefas da aplicação J_i que estão em execução no *host* $h_j \in H$. O cálculo de $PRA(J_i, h_j)$ é dado por:

$$PRA(J_i, h_j) = \frac{\sum_{t \in pt_i} pu(t)}{c_j} \tag{3.6}$$

onde $pu(t)$ é o percentual de processamento consumido por cada tarefa da aplicação J_i. Nota-se que o cálculo de $PRA(J_i, h_j)$ é uma média do consumo de CPU gerado pelas tarefas de J_i nos núcleos de h_j.

A obtenção dos dados para o cálculo das métricas e os cálculos em si são realizados por lógicas de monitoramento inseridas no EasyGrid AMS. Uma vez que o AMS de uma aplicação obtém os valores das métricas especificadas, o mesmo torna-se capaz de determinar se a execução da aplicação J_i está adiantada ou atrasada, em relação ao seu **Tempo Alvo** $TA(J_i)$ especificado pelo usuário. No entanto, o AMS deve ser capaz de adotar um comportamento apropriado, de acordo com a situação de atraso ou adianto da aplicação. Além disso, as decisões a respeito do escalonamento dependem também do consumo de recurso gerado pela própria aplicação e por outras que compartilham os recursos.

3.3.3 Métricas decisórias

Esta tese propõe uma extensão do diagrama de estados proposto em [55], no qual uma aplicação J_i admite diferentes estados de atraso em cada *host* $h_j \in H$ nas quais suas tarefas são executadas. No trabalho aqui apresentado, a diferença $TET(J_i, h_j) - TA(J_i)$ é definida como o fator determinante da quantidade de tarefas que o AMS da aplicação J_i deve executar no *host* h_j. Quanto menor for a diferença, maior a quantidade de tarefas que devem ser criadas e executadas. No entanto, o percentual total de consumo dos recursos de processamento $PTR(h_j)$ de cada *host* h_j nos quais a aplicação J_i encontra-se em execução também devem ser levados em consideração. Caso contrário, as decisões de escalonamento de uma aplicação podem afetar negativamente o desempenho de outras que estejam compartilhando os recursos de processamento. De maneira geral, o AMS de J_i decide a quantidade máxima de tarefas que devem ser criadas e executadas em cada *host* h_j conforme a Tabela 3.1.

Tabela 3.1: Quantidade de tarefas criadas pelo AMS de uma aplicação em função das relações entre o Tempo Alvo e o Tempo Estimado de Término da aplicação J_i e o Percentual Total de Consumo dos Recursos de Processamento do *host* h_j

	Diferença $TET(J_i, h_j) - TA(J_i)$	
PTR	baixa	alta
baixa	muitas tarefas	muitas tarefas
alta	poucas tarefas	muitas tarefas

Tabela 3.2: Definição dos estados de adianto/atraso de uma aplicação de acordo com a relação entre o Tempo Alvo (TA) e o Tempo Estimado de Término (TET) de uma aplicação J_i

ID	Estado de adianto/atraso	Relação $TA(J_i) \times TET(J_i, h_j)$
1	INDEFINIDO	N/A
2	MUITO ADIANTADO	$TET(J_i, h_j) < TA(J_i) \times \alpha$
3	ADIANTADO	$TA(J_i) \times \alpha \leq TET(J_i, h_j) < TA(J_i) \times \beta$
4	NO TEMPO	$TA(J_i) \times \beta \leq TET(J_i, h_j) \leq TA(J_i) \times \gamma$
5	ATRASADO COM CHANCE	$TA(J_i) \times \gamma < TET(J_i, h_j) \leq TA(J_i) \times \delta$
6	ATRASADO COM POUCA CHANCE	$TA(J_i) \leq TET(J_i, h_j) \times \delta$
7	ATRASADO SEM CHANCE	$TE(J_i) > TA(J_i)$

A relação $TA(J_i)$ e $TET(J_i)$ de uma aplicação J_i define os estados de comportamento que ela pode adotar durante a sua execução. Os estados 2, 3 e 4 são *não críticos* e indicam que a aplicação J_i está adiantada ou dentro do prazo em relação ao TA(J_i) em um determinado *host* h_j. Os estados 5, 6 e 7 são *críticos* e indicam que aplicação está atrasada em relação ao seu Tempo Alvo. A cada medição periódica realizada pelo AMS, a aplicação J_i pode apresentar apenas um estado em cada *host*. Na Tabela 3.2, é apresentado o conjunto de estados que uma aplicação pode adotar. São apresentadas também as respectivas relações entre $TET(J_i, h_j)$ e $TA(J_i)$, além de um número identificador (ID), usado para simplificar a identificação de cada estado. Por fim, deve-se destacar que o estado `ATRASADO SEM CHANCES` (7) é detectado quando o **Tempo de Execução** $TE(J_i)$ da aplicação J_i excede $TA(J_i)$.

Os valores adotados nesta tese para as constantes $\alpha = 0,5$, $\beta = 0,8$, $\gamma = 1,02$ e $\delta = 1,5$ foram obtidos empiricamente, por meio de experimentos. O objetivo foi alcançar valores que melhor representassem o limite de transição entre cada par de estados de comportamento.

Além das métricas que são recalculadas a cada medição, o AMS de cada aplicação J_i leva um conjunto de constantes em consideração para determinar o estado de atraso/a-

dianto no qual ela se encontra e a quantidade de tarefas que devem ser executadas. As constantes podem ser definidas com valores diferentes para cada aplicação que compõe o *workload*. São elas:

- `LIMIT`: usada para definir se um *host* h_j apresenta consumo de CPU reduzido ($PTR(h_j) \leq$ `LIMIT`) ou elevado ($PTR(h_j) >$ `LIMIT`);
- `MAX`: representa a quantidade máxima de tarefas de uma mesma aplicação J_i que podem executar simultaneamente em um mesmo *host* h_j;
- `FEW`: indica que uma pequena quantidade de tarefas devem ser executadas. Nesta tese, foi definido que o valor da constante `FEW` é igual ao teto 5% do valor definido para a constante `MAX`, ou seja, $FEW = \lceil MAX \times 0,05 \rceil$;
- `NONE`: constante utilizada para indicar ao AMS de uma aplicação que nenhuma nova tarefa deve ser executada;

A variação da quantidade de tarefas da aplicação J_i criadas e executadas pelo seu AMS a cada evento de monitoramento ocorre de acordo com os comportamentos descritos no Algoritmo 1. Antes de uma aplicação ter a sua execução iniciada, o $PTR(h_j)$ de cada *host* $h_j \in H$ utilizado é medido pelo seu AMS. Isso ocorre porque o AMS de J_i não possui informações suficientes para calcular o seu Tempo Estimado de Término neste momento, o que o impede de definir se a aplicação está atrasada ou não. Nesses casos, o estado `INDEFINIDO` (1) é detectado e a quantidade de tarefas executadas depende exclusivamente do $PTR(h_j)$. Todos os outros estados dependem do $TET(J_i)$, do $PTR(h_j)$ e do $PRA(J_i, h_j)$. Uma vez identificado qualquer estado diferente de `INDEFINIDO`, o AMS de J_i cria e executa a quantidade de tarefas definida na constante `MAX`, caso o $PTR(h_j)$ seja inferior ao fator de utilização de CPU do *host*, definido pela constante `LIMIT`. Caso contrário, as tarefas são criadas de acordo com o estado no qual a aplicação J_i se encontra. Para os estados não críticos `MUITO ADIANTADO` (2) e `ADIANTADO` (3), o AMS executa tarefas conforme o especificado nas constantes `NONE` e `FEW`, respectivamente. O estado não crítico `NO TEMPO` (4) faz com que o AMS crie e execute uma tarefa a mais a cada vez que este estado é detectado. Os estados críticos `ATRASADO COM CHANCE` (5) e `ATRASADO COM POUCA CHANCE` (6) forçam o AMS a criar e executar o máximo de tarefas permitido, definido na constante `MAX`. Por fim, o estado crítico `ATRASADO SEM CHANCE` (7) faz com que o AMS crie uma tarefa a menos a cada medição que este estado é detectado, até o limite definido pela constante `FEW`. O objetivo desse comportamento é garantir que uma aplicação não comprometa o desempenho de outras aplicações, caso $TE(J_i)$ exceda $TA(J_i)$.

Algoritmo 1: Lógicas adotadas pelo AMS de uma aplicação para mudar o comportamento e a quantidade de tarefas que devem ser executadas

```
while tarefa.pronta() is TRUE do
    calcula.TET(J_i, h_j);
    calcula.PTR(h_j);
    calcula.PRA(J_i, h_j);
    estado.define(J_i, h_j);
    if estado == INDEFINIDO then
        if PTR(h_j) > LIMIT then
            tarefa.executa(FEW);
        else
            tarefa.executa(MAX);
    if PTR(h_j) − PRA(J_i, h_j) ≤ LIMIT then
        tarefa.executa(MAX);
    else
        if estado == MUITO ADIANTADO then
            tarefa.executa(NONE);
        else if estado == ADIANTADO then
            tarefa.executa(FEW);
        else if estado == NO TEMPO then
            if prev.tarefas.executadas(J_i, h_j) < MAX then
                tarefa.executa(prev.tarefas.executadas(J_i, h_j) + 1);
        else if estado == ATRASADO SEM CHANCE then
            if prev.tarefas.executadas(J_i, h_j) > 1 then
                tarefa.executa(prev.tarefas.executadas(J_i, h_j) - 1);
        else
            tarefa.executa(MAX);
```

Em complemento aos comportamentos que podem ser adotados pelas aplicações durante as suas execuções, foi elaborada também uma política denominada `COMPLETA CARGA`. Esta política é responsável por monitorar e detectar ciclos ociosos de processamento nos núcleos dos *hosts*. Cada vez que ciclos ociosos são detectados, uma tarefa adicional é criada pelo AMS naquele *host*, mesmo que a quantidade total de tarefas ultrapasse o limite especificado pela constante `MAX`. Os objetivos desta política envolvem melhor aproveitar os recursos computacionais de processamento e antecipar a execução de tarefas que encontram-se prontas, mas que precisariam aguardar até uma próxima medição para serem iniciadas.

Ademais, a política `COMPLETA CARGA` proposta em [25] garante que o tempo de término

da aplicação será o mesmo em todos os *hosts* utilizados, salvo atrasos causados pelo custo da comunicação intrínsecos ao EasyGrid AMS. Essa política, aliada ao fato de que a estratégia proposta nesta tese garante 100% de utilização dos *hosts*, o tempo de execução não pode ser menor, mesmo que haja o atraso de outras aplicações.

3.4 Síntese do capítulo

Neste capítulo foi apresentada a evolução do escalonamento de aplicações, com os argumentos que motivaram a elaboração da estratégia de escalonamento proposta. Em seguida, o problema de escalonar aplicações que compartilham recursos computacionais foi apresentado e formalmente definido. Por fim, a estratégia proposta foi apresentada em detalhes.

A estratégia proposta foi projetada com a finalidade de melhorar significativamente os resultados do escalonamento de aplicações em ambientes compartilhados. As melhorias são traduzidas na redução do *makespan* e do tempo médio de *turnaround* do *workload*, além de praticamente eliminar os períodos em que os recursos de processamento permanecem ociosos. Além disso, as lógicas adotadas para realizar a troca de comportamento e a variação da quantidade de tarefas, aliadas ao uso do EasyGrid AMS, tornaram a estratégia extremamente simples. Os experimentos e resultados apresentados e discutidos no Capítulo 4 comprovam não apenas a simplicidade de uso da estratégia para escalonar um mesmo *workload* de formas diferentes, mas também os benefícios que são alcançados por meio da sua adoção.

Capítulo 4

Avaliação experimental da estratégia de escalonamento

Neste capítulo são apresentados os experimentos realizados com a finalidade de validar a estratégia proposta para o escalonamento de aplicações que compartilham recursos computacionais de processamento. São detalhados a infraestrutura computacional física e os recursos utilizados, os conjuntos de experimentos que foram executados e as discussões a respeito dos resultados obtidos.

Os experimentos e resultados foram organizados de acordo com os seus propósitos. Primeiramente, são apresentados os experimentos base, realizados com o intuito de obter valores de referência para validar a estratégia. Em seguida, foram realizados experimentos para validar o uso do parâmetro **Tempo Alvo** como critério de prioridade sobre o uso dos recursos de processamento por parte das aplicações e, com isso, ser capaz de escalonar o mesmo *workload* de maneiras diferentes. Foram realizados também experimentos com o objetivo de explorar os diferentes escalonamentos possíveis e avaliar o efeito de cada um deles sobre o *makespan* do *workload*. Por fim, foram realizados experimentos que tiveram a finalidade de avaliar o comportamento da estratégia proposta, além dos benefícios alcançados com a sua adoção, ao resolver o problema do compartilhamento de recursos comumente enfrentado por sistemas escalonadores tradicionais.

4.1 O ambiente computacional utilizado

A infraestrutura computacional utilizada nos experimentos é composta por quatro *hosts*, sendo um mestre e três de trabalho. O *host* mestre possui a responsabilidade de realizar o balanceamento de carga dos *hosts* de trabalho, por meio do uso do EasyGrid AMS. Os

hosts de trabalho, por sua vez, são responsáveis por executar as tarefas das aplicações. Essa infraestrutura foi utilizada com o propósito de validar a estratégia de escalonamento proposta em um ambiente controlado e dedicado para a realização dos experimentos. A execução das aplicações no ambiente se dá por meio da adoção do modelo de programação paralela utilizado com memória distribuída, no qual os processos comunicam-se por meio de trocas de mensagens. Cada *host* possui as seguintes características:

- Processamento: 2x Intel® Xeon CPU X5650 2,67 GHz (6 núcleos), com os recursos de *Hyperthread* desabilitados. Em um primeiro momento, não é objetivo do trabalho analisar a concorrência causada pela virtualização dos núcleos, e sim aquela gerada pelas tarefas.
- Memória: 6x 4GB Samsung DIMM DDR3 1333 MHz.
- Disco: 1x *Seagate Constellation* ES 500 GB 7200RPM SATA 3Gb/s 32 MB Cache 3.5 Inch *Internal Hard Drive* ST3500514NS.
- Redes de comunicação: as trocas de mensagens entre os *hosts* de processamento são realizadas por meio de interfaces *Gigabit Ethernet*.
- Sistema Operacional (SO): o Sistema Operacional adotado foi o Linux Ubuntu 14.04 LTS. Não foram realizadas configurações adicionais em relação ao escalonamento feito pelo SO;
- Interface de troca de mensagens: foi adotada a implementação LAM MPI 7.1.4 como interface paralela de troca de mensagens entre as aplicações. Ele foi escolhido por oferecer recursos de tolerância falhas e gerência dinâmica de processos. Esses dois recursos são amplamente explorados pelo AMS EasyGrid , com a finalidade de tornar as aplicações autogerenciáveis.

Foi utilizada uma aplicação paralela MPI sintética do tipo *bag of tasks* (que será referida como *app*), que realiza a multiplicação de valores de ponto flutuante gerados aleatoriamente. A aplicação foi projetada de modo que seja possível definir a quantidade de tarefas que ela irá executar e qual a duração de cada tarefa. Contudo, a mesma é maleável, isto é, a sua quantidade de tarefas pode variar em tempo de execução [31].

A escolha de uma aplicação do tipo *bag of tasks* é justificada pela necessidade de ter conhecimento completo sobre o comportamento da execução. Adicionalmente, a escolha também se deve à ausência de dependências entre suas tarefas. Tal característica faz com

que as tarefas não precisem aguardar o término de outras, podendo iniciar a sua execução a qualquer momento. Este cenário favorece às análises desejadas, uma vez que buscou-se avaliar a capacidade do AMS de gerenciar as aplicações e suas tarefas, independentemente da existência de qualquer relação de precedência.

4.2 Descrição dos experimentos

Os experimentos foram estruturados e realizados de modo que as aplicações fossem executadas imediatamente após suas submissões, eliminando assim a necessidade de gerenciar filas de execução. Isso foi feito com a intenção de avaliar a capacidade da estratégia proposta de executar diferentes quantidades de aplicações concorrentemente.

Os conjuntos de experimentos foram projetados com o propósito de avaliar e validar a estratégia de escalonamento de aplicações que compartilham os recursos de processamento do ambiente computacional. Para isso, foram analisados os diferentes escalonamento de um mesmo *workload* e os benefícios que podem ser alcançados por meio deles. Tais benefícios favorecem a dois grupos de interesse. O primeiro grupo é composto pelos donos ou gestores dos recursos computacionais, que podem alcançar o aumento da taxa de utilização do ambiente. O segundo grupo é formado pelos usuários que utilizam o ambiente para executar as aplicações, uma vez que a estratégia proposta possível reduzir o tempo necessário para executar os seus *workloads*. Os conjuntos de experimentos e os seus propósitos são descritos a seguir:

1. Experimentos base: conjunto de experimentos realizados com o propósito de obter os resultados de referência da execução da aplicação app_i utilizando os três *hosts* disponíveis. Foram considerados cenários nos quais a aplicação é executada com e sem o uso do EasyGrid AMS, a fim de mostrar o custo de incluir um gerenciador de aplicações para monitorar a sua execução. Ademais, foram realizados experimentos com o EasyGrid AMS e diferentes valores de $TA()$, porém sem o compartilhamento de recursos com outras aplicações. Esse experimento foi realizado para demostrar que o **Tempo Alvo** apenas influencia na execução de uma aplicação quando há o compartilhamento de recursos que acarreta em concorrência.

2. Avaliação do **Tempo Alvo** para prover diferentes resultados de escalonamento para o mesmo *workload*: conjunto de experimentos no qual duas ou mais aplicações foram executadas compartilhando os recursos de processamento. São usados diferentes valores de $TA()$, com a finalidade de estabelecer a ordem em que as aplicações devem

ter a sua execução finalizada e, com isso, definir qual a prioridade de utilização dos recursos compartilhados.

3. Avaliação do impacto da estratégia de escalonamento proposta sobre o *makespan* do *workload*: neste conjunto de experimentos, duas ou mais aplicações foram executadas de forma a compartilhar os recursos de processamento. Os valores diferentes de $TA()$ foram experimentados com o objetivo de avaliar o impacto da estratégia proposta e do compartilhamento de recursos sobre o *makespan* do *workload*. Para isso, foram executadas instâncias da mesma aplicação, porém variando o **Tempo Alvo**, com o intuito de o impacto causado em diferentes situações.

4. Avaliação dos benefícios da estratégia proposta em um cenário comum de escalonamento: este conjunto de experimentos foi elaborado para avaliar o comportamento da estratégia proposta em um cenário no qual há o compartilhamento parcial dos recursos. Para isso, cada aplicação foi configurada de modo a requerer apenas 2 dos 3 *hosts* para executar. Foi realizada a comparação dos resultados do escalonamento realizado por sistemas tradicionais de escalonamento e aqueles resultantes pela adoção da estratégia proposta. Os resultados mostram que é possível reduzir o *makespan* do *workload* e o tempo médio de *turnaround*, ao mesmo tempo em que reduz-se ou até mesmo elimina-se por completo a ociosidade dos recursos de processamento.

Para todos os experimentos realizados, cada aplicação foi composta por 512 tarefas. Cada tarefa foi configurada de forma a levar 5 segundos para executar quando não há concorrência, ou seja, seu custo é igual a 5. As medições e as tomadas de decisão também são realizadas pelo AMS a cada 5 segundos. Os AMSs de todas as aplicações tiveram suas contantes `MAX` configuradas com valores iguais a 18 tarefas simultâneas. Esse valor foi adotado ao considerar que a quantidade máxima de tarefas concorrentes em um mesmo *host* é igual a 1,5 vezes a sua quantidade de núcleos de processamento, com a finalidade de causar avaliar o tratamento de tarefas concorrentes feita pela estratégia, mesmo quando há apenas uma aplicação em execução. Assim, ao menos metade dos núcleos de um *host* podem ser compartilhados pelas tarefas de uma mesma aplicação.

4.3 Análise dos resultados

Nesta seção, são apresentados os resultados e as respectivas análises para cada um dos conjuntos de experimentos realizados. Além do experimento base, são apresentados os

resultados dos experimentos nos quais os valores de **Tempo Alvo** das aplicações foram ajustados com a finalidade de executar diferentes escalonamentos para o mesmo *workload*. São apresentados e comentados também os resultados de avaliação do impacto da estratégia proposta sobre o *makespan* do *workload*. Por fim, é realizada a análise da adoção da estratégia de escalonamento proposta, em relação aos benefícios gerados tanto para os gestores dos recursos como para os usuários das aplicações.

4.3.1 Experimentos base

Os experimentos base foram realizados com o propósito de obter valores de referência para a comparação e validação da estratégia de escalonamento. Este conjunto de experimentos considerou duas situações. Na primeira, a aplicação paralela usada nos experimentos foi executada sem o gerenciamento do EasyGrid AMS. Na segunda, a aplicação teve a sua execução gerenciada pelo do EasyGrid AMS com diferentes valores de **Tempo Alvo**, e sem compartilhar recursos com outras aplicações. O objetivo deste experimento foi averiguar o efeito do EasyGrid AMS com a estratégia proposta e com diferentes valores de **Tempo Alvo** sobre o tempo de execução das aplicações, quando não há o compartilhamento de recursos. Para isso, foram usados 4 *hosts*, sendo 1 mestre (Ilios) e 3 de trabalho (Dorado, Hanamura e Hollywood). Os resultados destes experimentos são exibidos na Tabela 4.1 e os valores de tempo são expressos em segundos. Os tempos de término considerados como base foram os maiores dentre aqueles obtidos em cada um dos *hosts* (valores em negrito).

Tabela 4.1: Resultados dos experimentos base com e sem o AMS e diferentes valores de Tempo Alvo na execução de uma aplicação app_1 com 512 tarefas

Hosts	Sem AMS	Valores de $TA(app_1)$				
		40 segs	80 segs	160 segs	240 segs	360 segs
Dorado	**74,982**	75,370	75,957	75,803	75,486	75,538
Hanamura	74,971	**75,381**	**75,975**	**75,814**	75,467	**75,552**
Hollywood	74,969	75,356	75,945	75,784	**75,495**	75,522

Por meio da análise dos resultados, é possível observar que a escolha de valores distintos de **Tempo Alvo** é indiferente, quando se trata de um ambiente no qual não há o compartilhamento de recursos. A maior variação entre os tempos médios de execução foi de aproximadamente 0,6 segundos entre as configurações de 40 e 80 segundos de **Tempo Alvo**. Ao mesmo tempo, foi possível observar que o uso do AMS impactou em um aumento de aproximadamente 1 segundo, para o $TA(app_1) = 80$ segundos. O aumento do tempo médio de execução era esperado em razão das chamadas das rotinas de monitora-

mento do EasyGrid AMS. Concluiu-se que o aumento de no máximo 1 segundo do tempo de execução é pouco significativo e, uma vez que não é detectada concorrência, a estratégia apresenta um tempo de execução muito próximo ao da execução paralela convencional.

Os resultados apresentados foram obtidos por meio da média aritmética de 30 execuções de cada um dos experimentos com intervalo de confiança de 95%. Para os experimentos que usaram o AMS, foram escolhidos diferentes valores de **Tempo Alvo** múltiplos de 80 segundos. Tais valores foram escolhidos em razão do tempo de execução médio de 74,982 segundos sem o AMS. Sendo assim, $TA(app_1) = 80$ segundos é considerado suficiente para executar a aplicação.

A título de comparação, foi considerado, para o experimento base, que um conjunto com uma dada quantidade de aplicações é executado de acordo com as especificações de sistemas escalonadores de aplicações HPC convencionais como o SLURM [42] e o TORQUE [72], que utilizam o modelo de alocação exclusiva[1]. Assim sendo, o cálculo do *makespan* do *workload* foi realizado considerando que as aplicações paralelas são executadas uma após a outra. Por fim, foram considerados conjuntos compostos por 2, 3, 4 e 5 instâncias da mesma aplicação com as mesmas 512 tarefas. Os valores do *makespan* do *workload* são exibidos na tabela 4.2.

Tabela 4.2: Resultados de *makespan* do experimento base para quatro *workloads* com quantidades diferentes de instâncias da aplicação *app*

	2 aplicações	3 aplicações	4 aplicações	5 aplicações
Makespan (s)	149,964	224,946	299,928	374,910

Para o cálculo do *makespan* do *workload* do experimento base, foi utilizado como referência o tempo de execução da aplicação sem o uso do EasyGrid AMS. O objetivo é validar a estratégia de escalonamento desenvolvida comparando o seu resultado com aquele obtido por meio da execução convencional de uma aplicação paralela. O *makespan* foi calculado por meio do somatório do tempo de execução de todas as aplicações do conjunto, considerado como o instante de término da última aplicação.

[1]Modelo de alocação em que uma aplicação só inicia a sua execução quando todos os seus requisitos de CPU são atendidos e alocados exclusivamente para ela.

4.3.2 Validação do Tempo Alvo como critério de prioridade: uma única estratégia, diferentes escalonamentos

O objetivo deste conjunto de experimentos é validar o uso do **Tempo Alvo** para estabelecer a prioridade da execução das aplicações, em relação à utilização dos recursos compartilhados da infraestrutura. Com isso, visa-se alcançar a possibilidade de escalonar um mesmo *workload* de formas diferentes por meio apenas do ajuste do parâmetro $TA()$. A prioridade sobre o uso dos recursos é estabelecida pela relação entre $TA(app_i)$ e o $TET(app_i)$, de acordo com os estados de adianto/atraso de cada aplicação app_i. A validação da capacidade da estratégia proposta realizar diferentes escalonamentos é importante para avaliar os ajustes ideais de $TA()$ para cada aplicação, seja para reduzir a ociosidade dos recursos, seja para reduzir o *makespan* do *workload*.

Foram projetados quatro cenários de experimentos, envolvendo duas (cenários 1, 2 e 3) ou três aplicações (cenário 4) que iniciam suas execuções simultaneamente, com intuito de permitir a análise dos seus comportamentos durante os períodos em que elas se encontrassem em execução. A exceção é o cenário 3, no qual as aplicações são iniciadas em instantes diferentes. Os experimentos foram estruturados de modo que as aplicações utilizassem todos os núcleos de processamento disponíveis nos 3 *hosts* utilizados. Esta análise da prioridade foi realizada por intermédio da verificação de que o EasyGrid AMS de uma aplicação com menor $TA(app_i)$ (maior prioridade) executou mais tarefas do que outra aplicação com maior $TA(app_j)$ (menor prioridade). Portanto, analisou-se a capacidade do EasyGrid AMS de cada aplicação de aumentar e reduzir a quantidade de tarefas em execução, de acordo com a variação da prioridade. Os valores de **Tempo Alvo** adotados para cada experimento foram configurados com a finalidade de permitir que as aplicações obtivessem diferentes prioridades de acesso aos recursos de processamento, o que resultou em diferentes escalonamentos.

É importante ressaltar que o **Tempo Alvo** é sugerido pelo usuário da aplicação, e que o mesmo interfere no resultado do escalonamento. As análises foram realizadas para avaliar o comportamento das aplicações de um mesmo *workload*, em função de diferentes valores de $TA(app_i)$ escolhidos por um usuário. Para as análises, os valores de $TA(app_i)$ são considerados curtos ou longos dependendo da sua relação com o tempo de execução das aplicações no experimento base.

4.3.2.1 Cenário 1: duas aplicações com Tempo Alvo curto e iniciando no mesmo momento

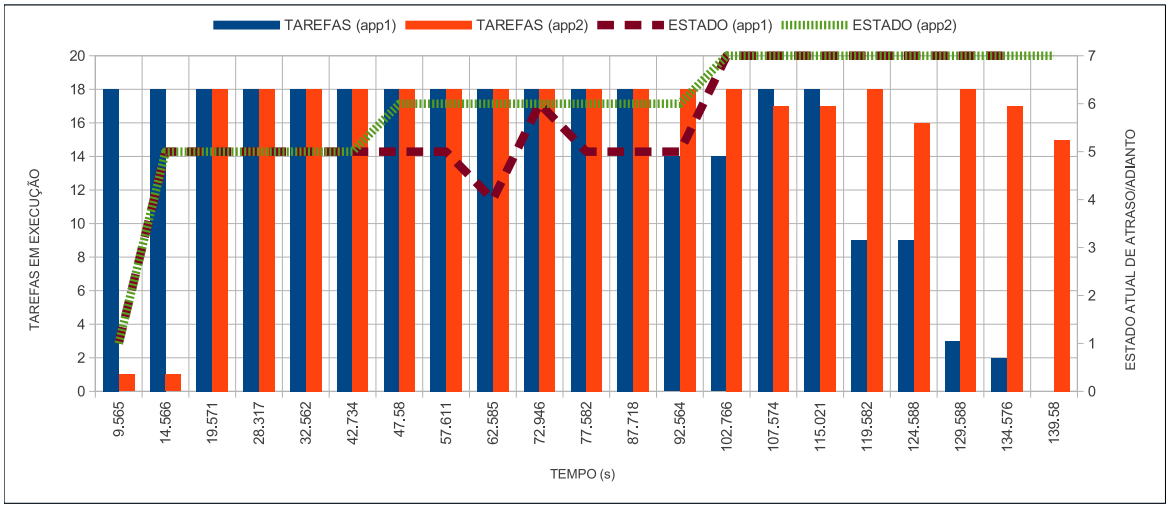
O primeiro cenário contempla duas aplicações app_1 e app_2 com valores de $TA(app_1) = TA(app_2) = 100$ segundos. O propósito deste cenário é analisar o comportamento da estratégia de escalonamento em uma situação na qual as duas aplicações possuem prioridade alta. Essas prioridades altas são expressas por meio de um $TA(app_i)$ curto em relação ao tempo de execução obtido no experimento base. Em virtude disso, esperava-se que os dois AMSs competissem fortemente pelos recursos compartilhados, ao criar a maior quantidade possível de tarefas. As variações da quantidade de tarefas em execução e do comportamento das aplicações em cada um dos *hosts* podem ser observadas nos gráficos da Figura 4.1, enquanto a identificação dos estados segue aquela definida na Tabela 3.2.

Ao analisar o comportamento das aplicações neste cenário, foi possível observar que o AMS da aplicação app_1 não detectou ocupação de recursos de processamento no início da execução. Isso fez com que ele iniciasse a execução explorando o limite máximo de 18 tarefas em cada um dos *hosts*, devido à especificação do estado `INDEFINIDO` (1). Em contrapartida, o AMS de app_2 detectou uma elevada taxa de utilização, gerada por app_1, no início da sua execução. Com isso, o seu AMS criou apenas uma tarefa para a execução enquanto app_2 encontrava-se no estado `INDEFINIDO`.

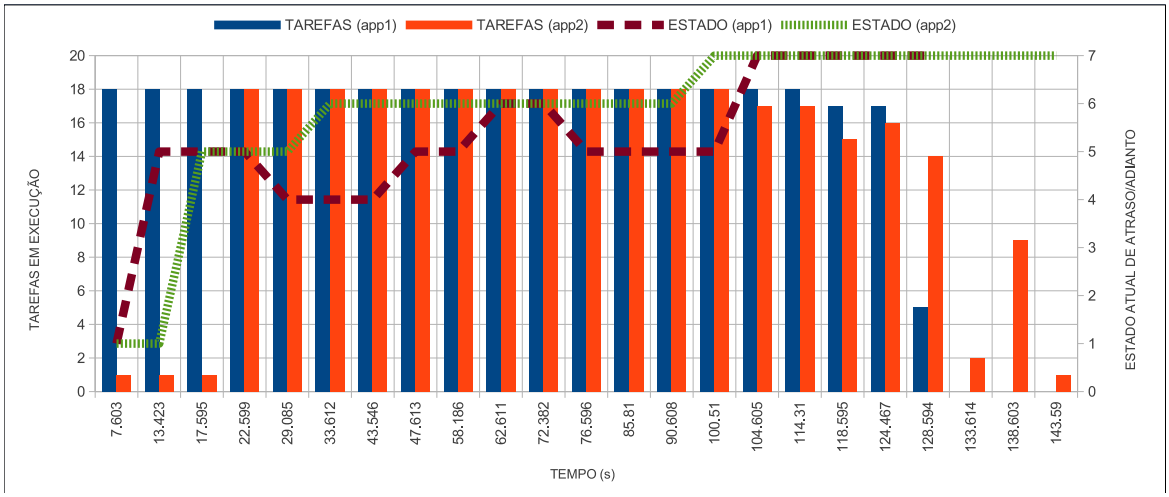
Em razão de ter executado uma maior quantidade de tarefas durante o período em que ficou em estado `INDEFINIDO` (1), app_1 permaneceu entre os estados `NO TEMPO` (4) e `ATRASADO COM CHANCE` (5) durante todo o período em que esteve em execução, até o instante em que o estado `ATRASADO SEM CHANCE` (7) foi identificado. O fato de ambas as aplicações possuírem valores curtos e iguais de **Tempo Alvo** fez com que a quantidade de tarefas executadas por elas fosse elevada durante a maior parte do período de execução.

A aplicação app_2 apresentou comportamento variando entre os estados `ATRASADO COM CHANCE` (5) e `ATRASADO COM POUCA CHANCE` (6) após deixar o estado `INDEFINIDO` (1). Esses comportamentos fizeram com que o seu AMS criasse a quantidade máximo de 18 tarefas, até que o limite do seu **Tempo Alvo** fosse atingido, pois tais estados são considerados críticos.

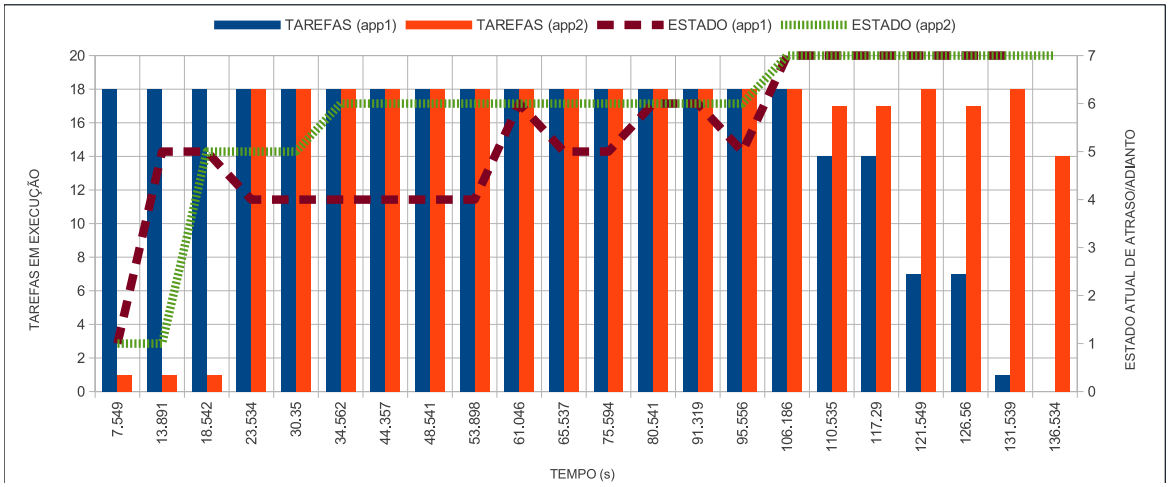
Uma vez que o **Tempo Alvo** de ambas as aplicações foi excedido, seus AMSs começaram a reduzir gradativamente a quantidade de tarefas, pois continuavam a detectar a competição pelo uso dos recursos. Com o término de app_1, o AMS de app_2 providenciou a execução das suas tarefas restantes.



(a) Quantidade de tarefas e comportamento das duas aplicações: *host* Dorado



(b) Quantidade de tarefas e comportamento das duas aplicações: *host* Hanamura



(c) Quantidade de tarefas e comportamento das duas aplicações: *host* Hollywood

Figura 4.1: Duas aplicações com 512 tarefas cada compartilhando os três *hosts*, ambas com valores curtos de Tempo Alvo, iniciando a execução no mesmo instante

No que diz respeito ao uso dos recursos, os resultados mostram que as duas aplicações foram executadas com prioridades altas, estabelecidas por meio da configuração de valores de **Tempo Alvo** iguais e curtos. Isso comprova que quanto menor o **Tempo Alvo**, maior é a prioridade da aplicação. Além disso, os resultados mostram também que, nessas circunstâncias, as aplicações concorrem de maneira mais intensa pelos recursos.

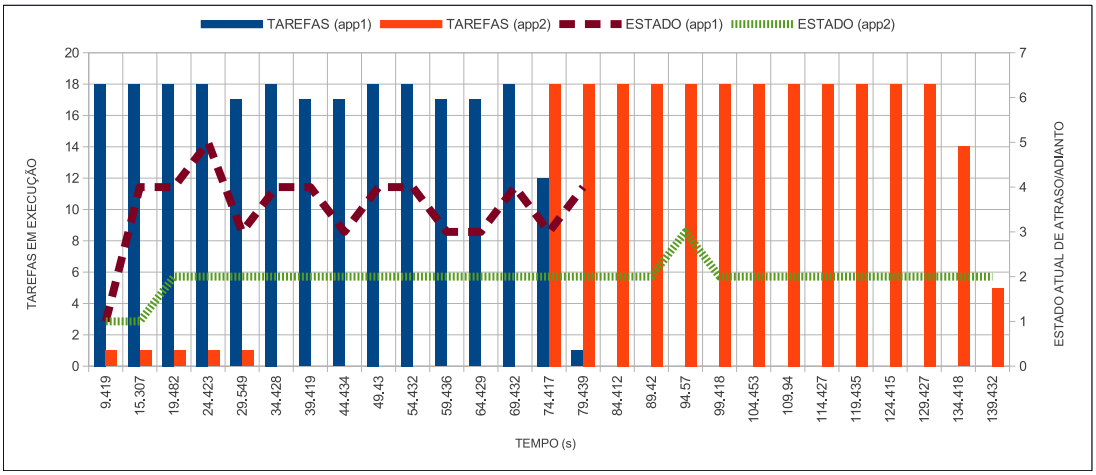
As configurações de $TA()$ utilizadas neste cenário resultaram em tempos de execução iguais a 136,925 segundos para app_1 e 144,564 segundos para app_2, cujo instante de término representa também o *makespan* do *workload*. Esse resultado representa uma redução de 3,6% em relação ao experimento base. É importante ressaltar que os valores de **Tempo Alvo** não foram escolhidos com o objetivo de reduzir o *makespan*, mas sim para analisar o comportamento da estratégia quando duas aplicações apesentam valores reduzidos para $TA()$. Tal redução ocorreu naturalmente apenas pela adoção da estratégia proposta, em relação ao *makespan* do *workload* o experimento base.

4.3.2.2 Cenário 2: uma aplicação com Tempo Alvo curto e outra com Tempo Alvo longo iniciando no mesmo momento

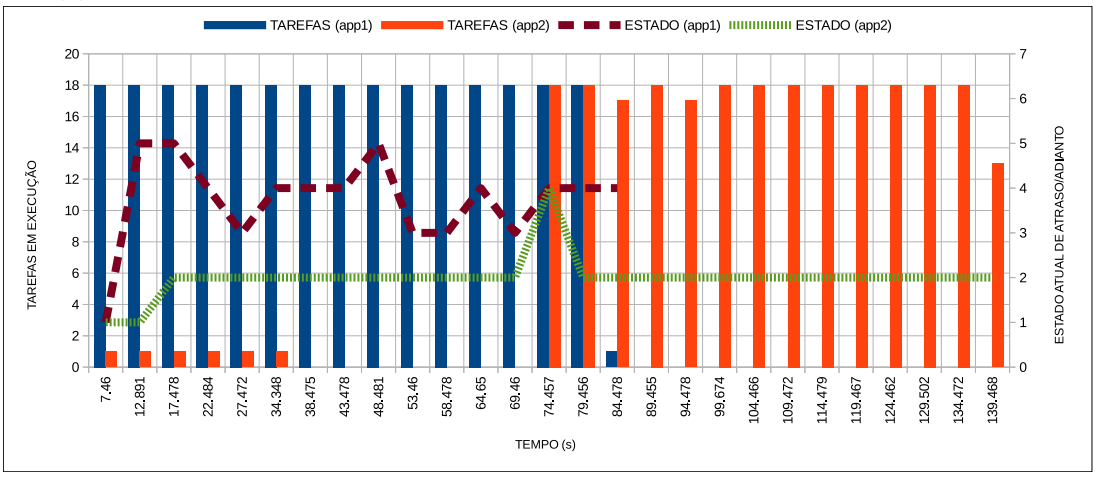
Este cenário foi projetado com a finalidade de verificar o comportamento de duas aplicações com configurações de **Tempo Alvo** diferentes, sendo $TA(app_1) = 100$ segundos (curto) e $TA(app_2) = 400$ segundos (longo). A diferença entre os dois valores de **Tempo Alvo** foi estabelecida para garantir que a aplicação que deve terminar primeiro realmente tenha prioridade sobre o uso dos recursos em relação à outra aplicação que concorrente e, com isso, alcançar um resultado de escalonamento diferente daquele obtido no cenário 1.

Nos gráficos da figura 4.2, são apresentadas as medições realizadas pelos AMSs das duas aplicações. A cada medição são obtidas informações sobre a quantidade de tarefas em execução naquele momento e qual o estado de adianto/atraso em que a aplicação se encontra. As medições foram realizadas a cada 5 segundos, conforme configuração do EasyGrid AMS já mencionada, em cada um dos três *hosts* de trabalho utilizados.

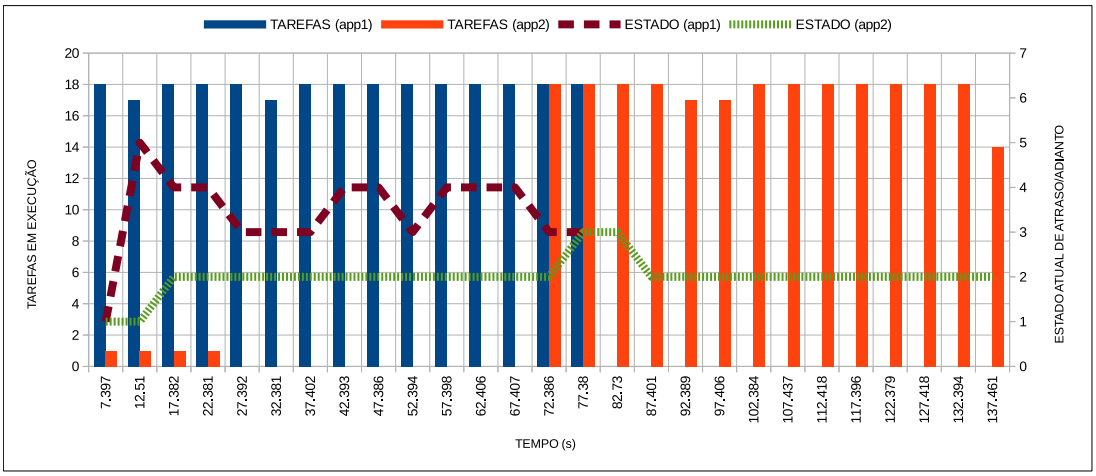
Por meio da análise dos resultados deste cenário, foi possível observar que app_1 obteve prioridade sobre o uso dos recursos em relação a app_2, por possuir um **Tempo Alvo** menor, o que fez sua execução tornar-se mais crítica. O AMS da aplicação app_1 identificou baixa utilização dos recursos durante o período em que permaneceu no estado `INDEFINIDO` (1). Com isso, foi adotado o comportamento de executar 18 tarefas, de acordo com a configuração utilizada para a quantidade máxima de tarefas executadas por aplicação em cada *host*. Uma vez que o **Tempo Alvo** de 100 segundos é suficiente, porém justo, para



(a) Quantidade de tarefas e comportamento das duas aplicações: *host* Dorado



(b) Quantidade de tarefas e comportamento das duas aplicações: *host* Hanamura



(c) Quantidade de tarefas e comportamento das duas aplicações: *host* Hollywood

Figura 4.2: Duas aplicações com 512 tarefas cada compartilhando os três *hosts*, uma com Tempo Alvo curto e outra com Tempo Alvo longo, iniciando a execução no mesmo instante

a execução de app_1 com compartilhamento de recursos, o seu AMS continuou a explorar o limite máximo de tarefas simultâneas executadas. Isso ocorreu apesar de estados `ATRASADO`

`COM CHANCE` (5) ou de maior atraso serem registrados em apenas 6,17% das medições em todos os *hosts* e também porque app_2 possui menor prioridade e não concorreu pelos recursos.

A prioridade de acesso aos recursos dada à app_1 durante a sua execução só foi possível devido à configuração de $TA(app_2) = 400$ segundos para app_2. Ao estabelecer um maior **Tempo Alvo** e uma vez identificada a presença de outra aplicação acessando os recursos compartilhados de processamento, o AMS de app_2 forçou a execução de apenas uma tarefa durante o período no qual esteve em estado `INDEFINIDO` (1). Em seguida, o estado mudou para `MUITO ADIANTADO` (2), o que fez com que o AMS de app_2 não criasse novas tarefas para execução enquanto o uso dos recursos por outras aplicações continuasse sendo identificado, ou até que houvesse alguma mudança para estados de atraso.

Os valores de **Tempo Alvo** empregados resultaram em um escalonamento que reduziu o *makespan* do *workload*. A aplicação app_1, com $TA(app_1) = 100$ segundos, apresentou um tempo de execução igual a 82,235 segundos, enquanto app_2, com $TA(app_2) = 400$ segundos, foi executada em 141,96 segundos, sendo este tempo o *makespan* do *workload*. Uma redução de 5,34% em relação ao experimento base. O conjunto de resultados deste cenário indica que a redução do *makespan* do conjunto de aplicações pode ser alcançado por meio da adoção de valores de **Tempo Alvo** diferentes, quando comparado com aquele obtido com o experimento base..

A análise do comportamento das aplicações neste cenário, confrontada com os resultados do cenário 1, comprovam a eficiência do **Tempo Alvo** como parâmetro para especificar as prioridades de uso dos recursos compartilhados entre as aplicações. Além disso, foi possível confirmar que a estratégia é capaz de realizar escalonamentos diferentes de um mesmo *workload*, apenas com o ajuste de um único parâmetro. No entanto, diferentes situações podem ocorrer em ambientes reais de execução. Logo, os cenários 3 e 4 foram estruturados com a finalidade de validar o **Tempo Alvo** como critério de prioridade em tais situações.

4.3.2.3 Cenário 3: uma aplicação com Tempo Alvo curto e outra com Tempo Alvo longo iniciando em momentos diferentes

O propósito deste cenário foi verificar o comportamento da estratégia proposta em situações nas quais uma aplicação com alta prioridade inicia sua execução e encontra outra aplicação com prioridade mais baixa já em execução. Para isso, foram utilizadas duas aplicações com **Tempo Alvo** diferentes iniciando suas execuções em momentos distin-

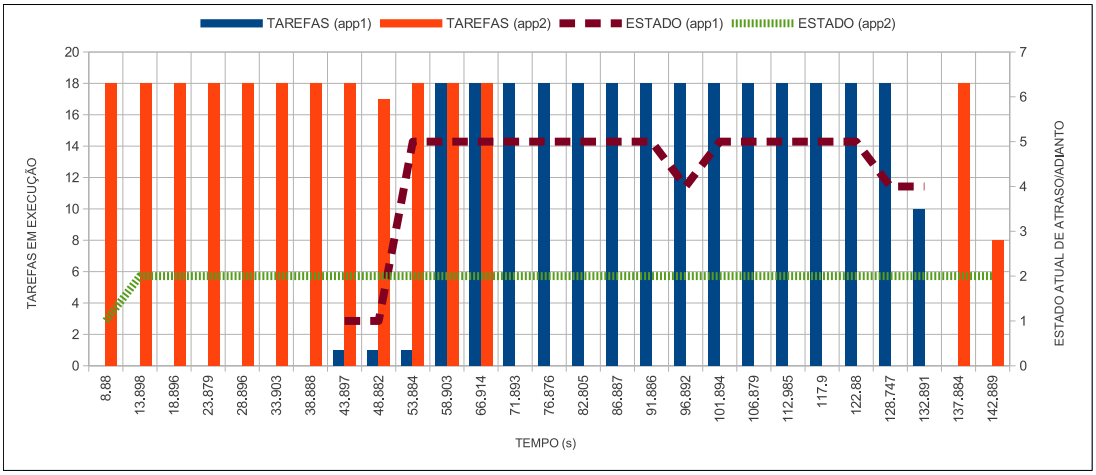
tos: app_2, com $TA(app_2) = 400$ segundos, teve sua execução iniciada no instante zero, enquanto app_1, com $TA(app_1) = 100$ segundos, iniciou 35 segundos após o início da execução de app_2.

Neste cenário, o AMS de app_2 tem ciência de que há tempo mais do que suficiente para concluir a execução. Quando uma outra aplicação (app_1) com maior prioridade é detectada, o AMS de app_2 reduz ou cessa a criação de novas tarefas, permitindo que a aplicação com maior prioridade (app_1) utilizasse os recursos com um nível reduzido de concorrência. Por meio da análise dos gráficos da Figura 4.3, é possível confirmar que o comportamento esperado foi de fato o obtido.

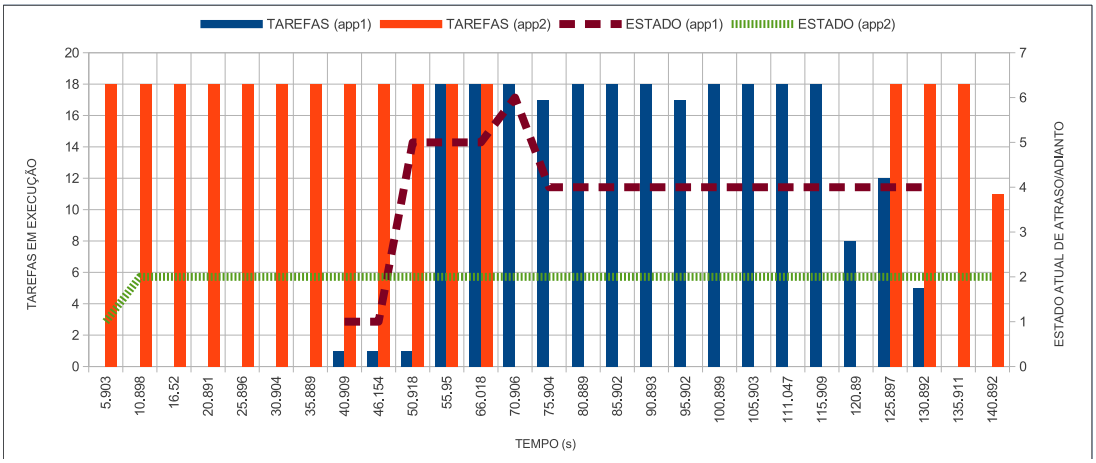
Uma vez iniciada a execução de app_2, o seu AMS não detectou qualquer utilização dos recursos por outras aplicações, o que o fez explorar o limite máximo de tarefas simultâneas em execução durante o período em que estava no estado `INDEFINIDO` (1). Em seguida, embora o AMS de app_2 tenha especificado o seu estado de adianto/atraso como `MUITO ADIANTADO` (2), também foi verificado que não havia utilização de recursos por outras aplicações. Assim, novas tarefas continuaram sendo executadas até que o início da execução de app_1 foi detectado. A partir do momento em que o AMS de app_2 verificou que a utilização de recursos por outra aplicação não reduziu, e visto que ainda persistia o estado `MUITO ADIANTADO` (2), a criação de novas tarefas foi interrompida. Somente após o término da execução de app_1 e a constatação de que os recursos não estavam mais sendo utilizados, o AMS de app_2 retomou a criação de tarefas. Além disso, o AMS de app_2 aguardou o término de app_1 apenas porque havia tempo suficiente para isso. É importante destacar que o comportamento descrito para app_2 ocorreu em todos os *hosts* nos quais ela foi executada.

O AMS de app_1, por sua vez, identificou a utilização de recursos enquanto encontrava-se no estado `INDEFINIDO` (1). Isso fez com que fosse adotado o comportamento de criar apenas uma tarefa enquanto permanecesse nesse estado. Entretanto, a primeira mudança de estado de app_1 foi para `ATRASADO COM CHANCE` (5) em todos os *hosts* e, em medições posteriores, ao estado `ATRASADO COM POUCAS CHANCE` (6) nos *hosts* Hanamura e Hollywood. Isso fez com que o seu AMS criasse uma quantidade elevada de tarefas até o término da execução da aplicação.

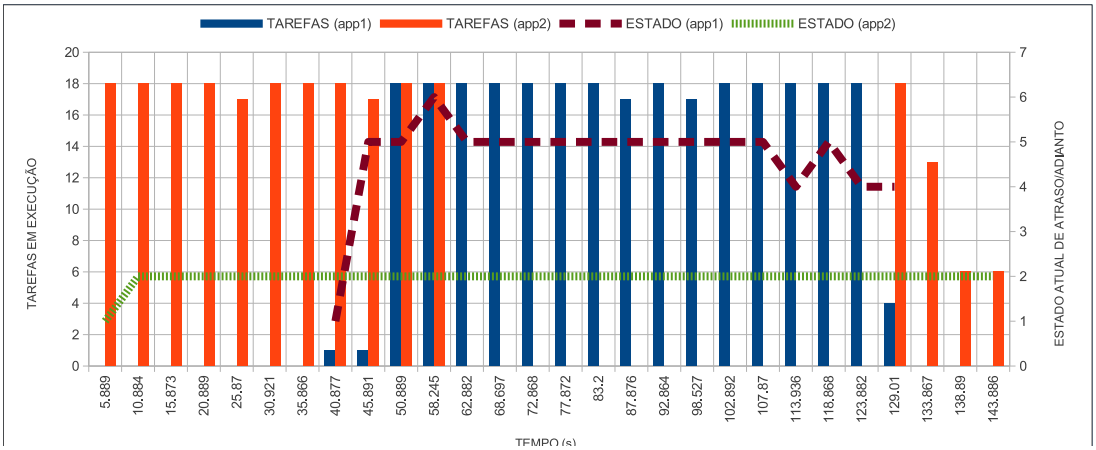
O escalonamento realizado neste cenário obteve um *makespan* de 147,75 segundos para o *workload*, sendo que este tempo se refere ao instante de término de app_2. Com isso, para este cenário, o *makespan* apresentou uma pequena redução de 1,47% em relação ao experimento base.



(a) Quantidade de tarefas e comportamento das duas aplicações: *host* Dorado



(b) Quantidade de tarefas e comportamento das duas aplicações: *host* Hanamura



(c) Quantidade de tarefas e comportamento das duas aplicações: *host* Hollywood

Figura 4.3: Duas aplicações app_1 e app_2 com 512 tarefas cada, compartilhando três *hosts*, onde $TA(app_1) = 100$ segundos (curto) e $TA(app_2) = 400$ segundos (longo), iniciando a execução em momentos distintos

Por meio da análise deste cenário, pode-se concluir que a prioridade pode ser estabelecida pela adoção do **Tempo Alvo** também em situações nas quais as aplicações iniciam

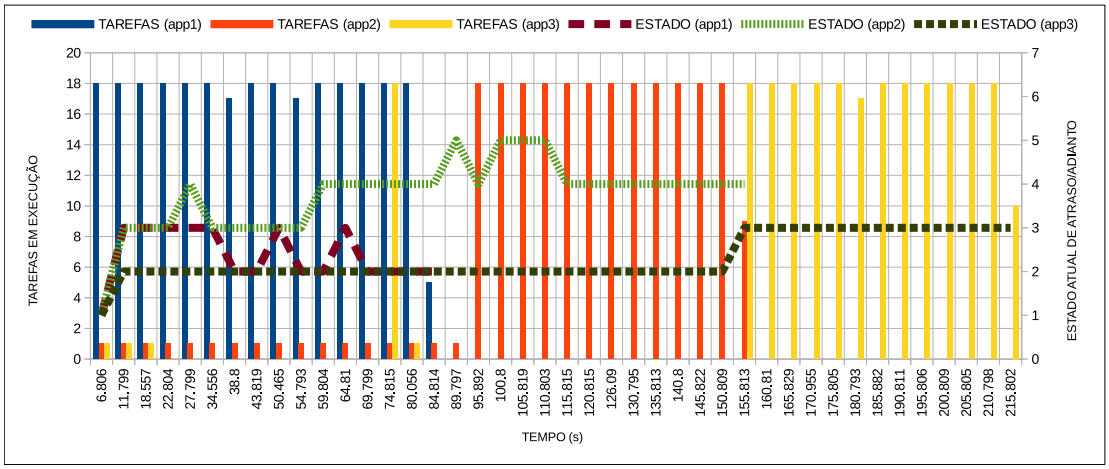
suas execuções em instantes distintos de tempo. Naturalmente, conclui-se também que o instante de início de cada aplicação também influencia no resultado do escalonamento. Portanto, é possível afirmar que a estratégia de escalonamento proposta nesta tese é aplicável a situações nas quais o instante de início das aplicações é desconhecido.

4.3.2.4 Cenário 4: três aplicações com Tempos Alvos diferentes iniciando no mesmo momento

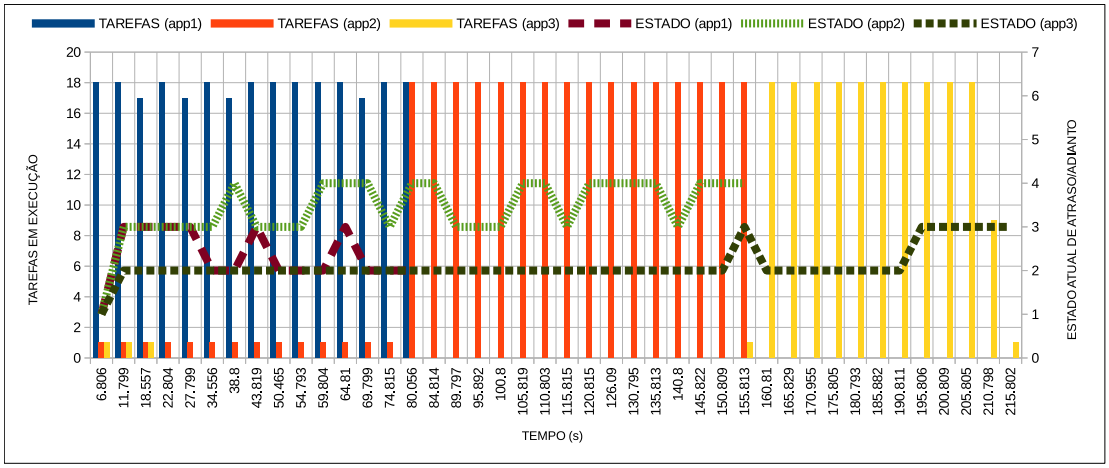
Este cenário foi projetado com o intuito de validar o uso do **Tempo Alvo** para estabelecer a prioridade de uso dos recursos compartilhados de processamento, em situações nas quais três aplicações executam simultaneamente. Para isso, os seguintes valores de **Tempo Alvo** foram adotados: $TA(app_1) = 100$ segundos; $TA(app_2) = 200$ segundos 100; e $TA(app_3) = 300$ segundos. Ao adotar tais configurações, esperava-se que o resultado do escalonamento fosse similar ao de um realizado por sistemas escalonadores tradicionais, ou seja, que fosse simulada uma execução sequencial das aplicações. No entanto, a estratégia proposta considera que o AMS de cada aplicação pode aproveitar ciclos ociosos de processamento para executar tarefas, ou ainda mudar o seu comportamento de acordo com os estados de adianto/atraso. Nos gráficos da Figura 4.4 são apresentadas as variações de comportamento das aplicações durante suas execuções, em cada um dos *hosts* utilizados.

Com os resultados obtidos pela execução de três aplicações simultaneamente, foi possível observar as adoções de diferentes níveis de prioridade entre as aplicações ao explorar o conjunto de *hosts* disponíveis. O **Tempo Alvo** de 100 segundos fez com que app_1 tivesse maior prioridade em relação às demais. Também foi constatado que a prioridade de app_1 fez com que seus estados variassem entre `NO TEMPO` (4) e `ADIANTADO` (3) em todos os *hosts*. A razão pela qual o seu AMS explorou o limite máximo de 18 tarefas, mesmo quando o estado `ADIANTADO` foi detectado, está relacionada à baixa prioridade das demais aplicações, e na consequente atenuação de demanda por recursos por parte delas. A aplicação app_1 continuou executando com maior prioridade até as duas últimas medições feitas pelo seu AMS, pois nesse ponto da execução, os AMSs de app_2 e app_3 detectaram a redução do consumo de recursos de processamento, o que os levou a intensificar a criação de novas tarefas para execução.

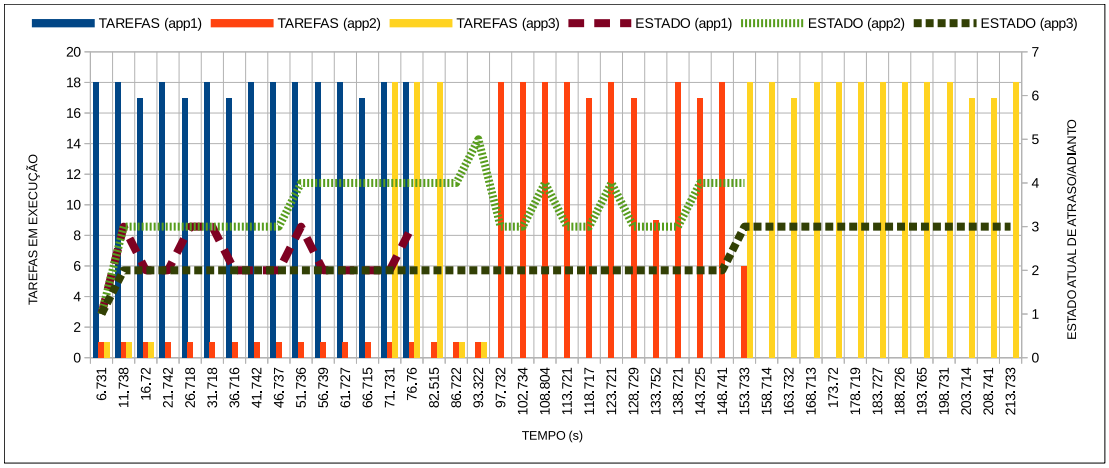
Os instantes finais da execução de app_1 determinaram o momento em que os AMSs de app_2 e app_3 precisaram decidir qual aquela que teria maior prioridade sem qualquer comunicação entre eles. É possível observar que isso ocorreu de maneira distinta em cada um dos três *hosts*. Nos *hosts* Dorado e Hollywood, o AMS de app_3 foi o primeiro



(a) Quantidade de tarefas e comportamento das três aplicações: *host* Dorado



(b) Quantidade de tarefas e comportamento das três aplicações: *host* Hanamura



(c) Quantidade de tarefas e comportamento das três aplicações: *host* Hollywood

Figura 4.4: Três aplicações com 512 tarefas cada compartilhando os três *hosts*, com Tempo Alvo distintos, iniciando a execução simultaneamente

a detectar que poderia explorar o limite de 18 tarefas simultâneas, mesmo possuindo prioridade menor do que app_2. Nas duas medições seguintes, no entanto, o AMS de app_2 acusou o estado `ATRASADO COM CHANCE` (5), o que o fez forçar a execução do limite máximo

de tarefas simultâneas. Tal comportamento se seguiu também pelo fato de app_3 estar no estado `MUITO ADIANTADO` (2), fazendo o seu AMS ceder recursos ao detectar que outra aplicação continuava executando com alta carga de processamento, uma vez que possuía maior prioridade.

A Figura 4.5 foi elaborada com o objetivo de facilitar a visualização do comportamento dos AMSs de cada aplicação, em relação à quantidade total de tarefas executadas. Para isso, foi adotada uma única escala de tempo referente aos dados obtidos pelas medições do AMS de app_2. Por meio da análise da Figura 4.5, fica evidente que as aplicações respeitaram as prioridades atribuídas a elas por intermédio dos valores de **Tempo Alvo**, mesmo sem haver comunicação entre si. É possível observar que o período de término de execução da app_1 coincide com aquele no qual as aplicações app_2 e app_3 decidem sobre qual possui a maior prioridade sobre o uso dos recursos.

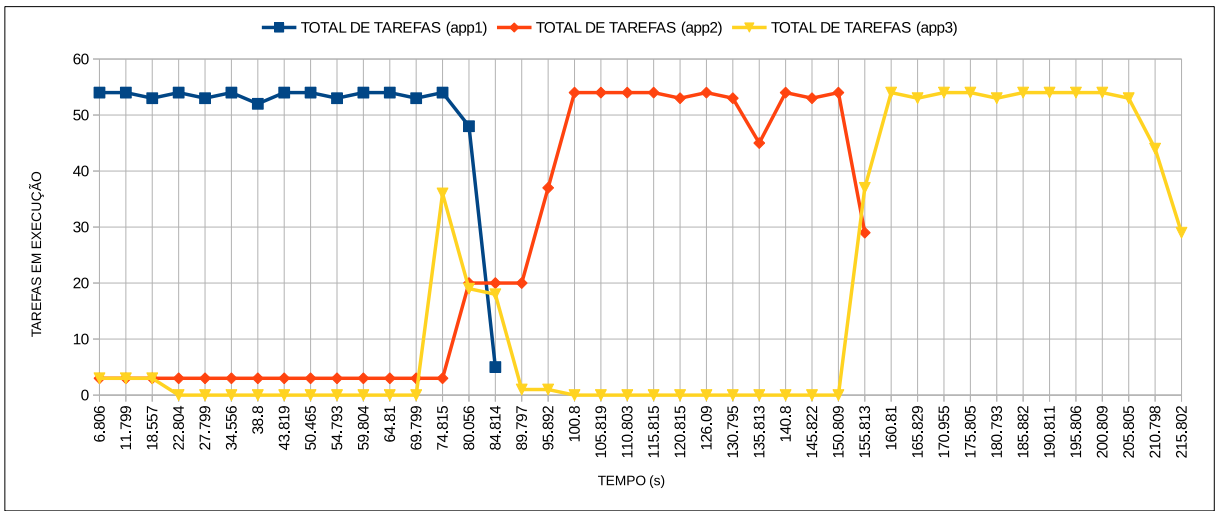


Figura 4.5: Visão geral do total de tarefas executadas por cada uma das três aplicações

O escalonamento deste cenário resultou em um *makespan* igual a 217,805 segundos para o *workload* com 3 aplicações, equivalente ao instante de término de app_3. Tal resultado representa uma redução de 3,17% em relação ao *makespan* de 224,946 segundos do experimento base.

4.3.2.5 Conclusões sobre o uso do Tempo Alvo como critério de prioridade

Os quatro cenários de experimentos anteriormente apresentados tiveram o objetivo de validar o uso do **Tempo Alvo** como critério de prioridade de acesso aos recursos compartilhados. Os três primeiros tiveram ainda o objetivo de mostrar três escalonamentos diferentes de um mesmo *workload* utilizando a mesma estratégia, porém, com instantes

de submissão e valores de **Tempo Alvo** diferentes. Por meio da análise dos resultados obtidos, foi possível concluir que diferentes configurações de **Tempo Alvo** para as aplicações não apenas influenciam nos seus comportamentos, como também provocam resultados diferentes para o escalonamento. Consequentemente, foi visto que aplicações que compartilham recursos e que possuem altas prioridades tendem a reduzir o *makespan* do *workload*. Ainda, observou-se que diferentes valores de **Tempo Alvo** podem reduzir o *makespan*, pois, de qualquer forma, o AMS tenta utilizar ao máximo os recursos disponibilizados.

Em geral, os resultados demostraram que o uso do **Tempo Alvo** representa um modo de controlar as prioridades da execução das aplicações em um ambiente compartilhado. Diferentes combinações de **Tempo Alvo** permitem alcançar um escalonamentos distintos, com maior ou menor concorrência entre as aplicações do *workload*.

4.3.3 Avaliação do impacto da estratégia proposta sobre o *makespan* do *workload*

Os experimentos realizados nesta subseção tiveram a finalidade de avaliar o impacto da estratégia de escalonamento proposta sobre o *makespan* do *workload* em situações nas quais há o compartilhamento de recursos. Isso foi feito ao confrontar os resultados obtidos considerando a política proposta com aqueles obtidos por meio do escalonamento realizado no experimento base. No entanto, os experimentos não estão restritos a apenas avaliar o *makespan*, mas também a analisar o comportamento da estratégia com diferentes quantidades de aplicações. Estes experimentos foram divididos em três cenários distintos.

Os resultados dos experimentos apresentados na Subseção 4.3.2 forneceram subsídios interessantes a respeito da configuração de **Tempo Alvo** das aplicações. Foi observado que diferentes valores de $TA()$ fazem com que a execução do conjunto de aplicações ocorra de maneira similar ao que acontece em um escalonamento tradicionalmente adotado em ambientes de HPC, porém, introduzindo graus de compartilhamento de recursos, que variam conforme as prioridades de cada aplicação. Por exemplo, a redução do *makespan* alcançada no experimento da Subseção 4.3.2.4, se deu em virtude da escolha dos valores de $TA(app_i)$ para cada uma das três aplicações.

Os resultados relativos aos tempos de execução das aplicações e aos *makespans* dos *workloads* dos experimentos dos três cenários a seguir foram obtidos por meio da média aritmética de 30 execuções de cada um dos experimentos, com intervalo de confiança de 95%.

4.3.3.1 Cenário 1: avaliação do impacto da estratégia proposta sobre o *makespan* de um *workload* com três aplicações

Neste primeiro cenário, foram realizados três experimentos, cada um composto pela execução de um *workload* com três aplicações. Para relembrar, cada aplicação (app_i) é do tipo *bag of tasks*, possuem 512 tarefas e têm suas execuções iniciadas no mesmo instante. Os valores de $TA(app_i)$ de cada aplicação, em segundos, foram escolhidos de maneira a serem maiores do que o tempo de execução de app_i no experimento base, que foi igual a 74,982 segundos. Os valores de **Tempo Alvo** são apresentados na Tabela 4.3.

Tabela 4.3: Valores de Tempo Alvo adotados para cada aplicação em cada um dos experimentos do Cenário 1

	TA(app_1)	TA(app_2)	TA(app_3)
Experimento 1	80	160	240
Experimento 2	100	200	300
Experimento 3	150	300	450

Os valores de $TA(app_1) = 80$ segundos, $TA(app_2) = 160$ segundos e $TA(app_3) = 240$ segundos adotados para as aplicações no Experimento 1 não permitiram a redução do *makespan* do *workload*, pois forçou os AMSs das aplicações a criar e executar quantidades elevadas de tarefas simultaneamente, aumentando a concorrência pelo uso dos recursos compartilhados. Os comportamentos dos AMSs das aplicações durante suas execuções causaram um aumento de 2,83% do *makespan*, em comparação ao experimento base. Os comportamentos que cada uma das três aplicações apresentou na execução cujos tempos de término mais se aproximaram da média das 30 execuções do Experimento 1 podem ser vistos no gráfico da Figura 4.6.

É importante destacar que houve concorrência elevada entre as aplicações durante 41,3% do tempo total de execução (entre os instantes 45,589 e 180,592 segundos), causada pela alta prioridade das aplicações. Somado a isso, há a relação entre os valores de **Tempo Alvo** das aplicações, com apenas 80 segundos de diferença entre eles. Portanto, concluiu-se que valores curtos de **Tempo Alvo** causaram o aumento de 2,83% do *makespan* do *workload*, uma vez que os mesmos causaram uma concorrência elevada pelos recursos compartilhados de processamento.

A configuração $TA(app_1) = 100$ segundos, $TA(app_2) = 200$ segundos e $TA(app_3) = 300$ segundos adotada no experimento 2 proporcionou uma redução de 3,17% do *makespan* do *workload*, quando comparado ao experimento base. Foi possível observar, por meio da análise da Figura 4.7, que a configuração adotada permitiu a cada AMS criar novas

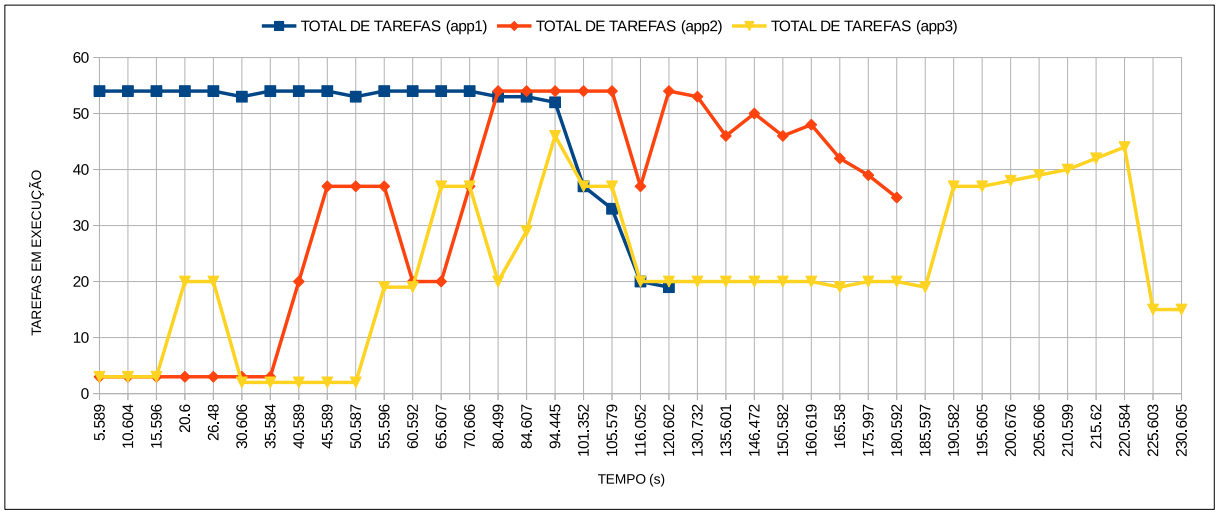


Figura 4.6: Cenário 1 - Experimento 1: quantidade de tarefas em execução para cada uma das três aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

tarefas, de modo a tirar proveito do compartilhamento de recursos, mas sem prejudicar as execuções das demais aplicações.

É importante ressaltar que a redução de 3,17% do *makespan*, embora pequena, foi obtida ao compartilhar os recursos, o que reduz a subutilização da infraestrutura. Portanto, qualquer redução do *makespan* alcançada por meio da estratégia de escalonamento proposta é considerada significativa.

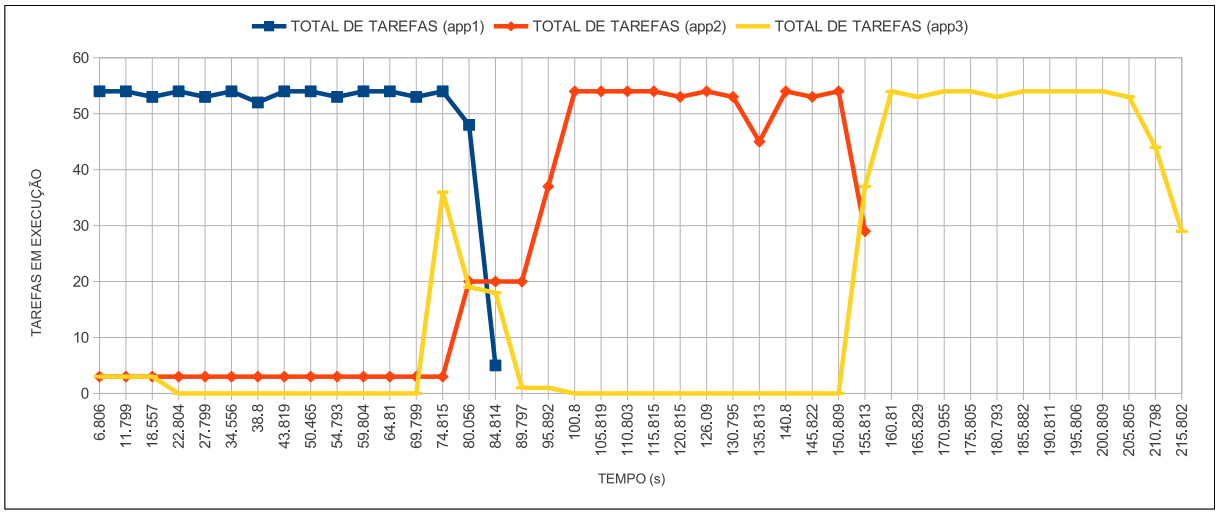


Figura 4.7: Cenário 1 - Experimento 2: quantidade de tarefas em execução para cada uma das três aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

Ainda sobre o resultado do Experimento 2 apresentado na Figura 4.7, foi observado que houve baixa utilização dos recursos no período compreendido entre 84,814 e 100,8 segundos. A explicação para o ocorrido está no tempo necessário para os AMSs das aplicações app_2 e app_3 decidirem qual delas possui maior prioridade e assumir o controle

efetivo dos recursos. O intervalo de tempo necessário para o AMS de app_3 detectar que poderia utilizar os recursos foi significativamente menor (em torno de 5 segundos), uma vez que ela era a única a ainda utilizar os recursos.

Para o Experimento 3, os valores de **Tempo Alvo** foram definidos de maneira que o de app_1 fosse aproximadamente o dobro do tempo necessário para executar uma instância da aplicação no ambiente não compartilhado (150 segundos). Os demais valores de **Tempo Alvo** foram definidos como múltiplos de 150 segundos. As configurações dos valores de **Tempo Alvo** adotadas para o experimento 3 acarretaram na redução de 1,16%, em relação ao *makespan* de 3 aplicações obtido pela execução do experimento base. Os comportamentos das aplicações mediante as configurações adotadas para o experimento 3 são apresentadas no gráfico da Figura 4.8.

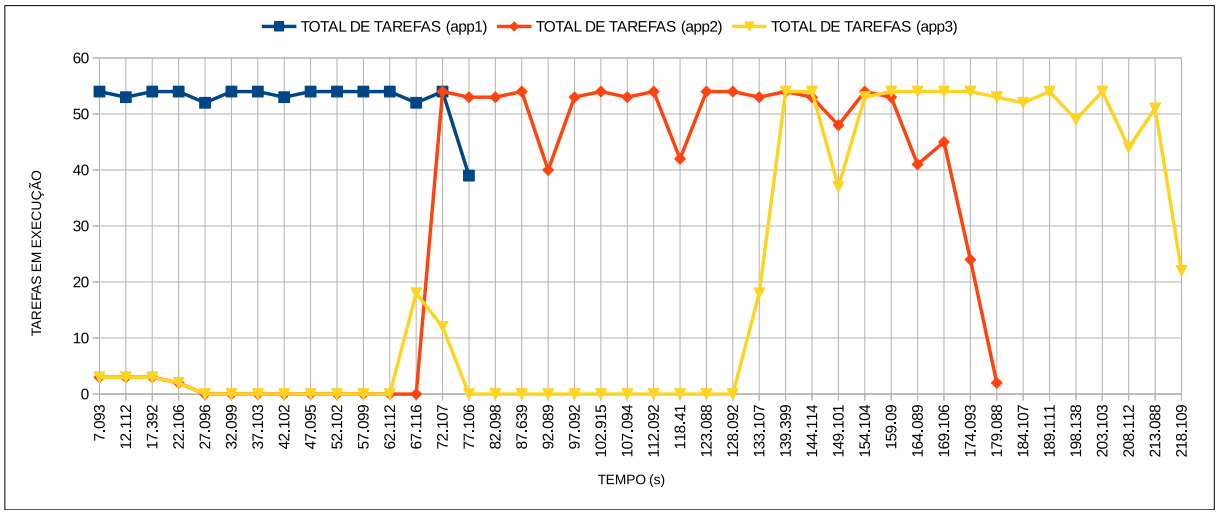


Figura 4.8: Cenário 1 - Experimento 3: quantidade de tarefas em execução para cada uma das três aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

A redução do *makespan* neste experimento é justificada pelo comportamento de app_3 no primeiro terço do período de medições. É possível observar que o seu AMS criou poucas ou nenhuma tarefa neste período. Isso ocorreu devido aos valores de **Tempo Alvo** de app_2 e app_3, que foram maiores do que aqueles do experimento 2. Assim, ambos os AMSs detectaram o estado `MUITO ADIANTADO` (2), no qual nenhuma nova tarefa é criada, além de detectarem carga de processamento de outra aplicação. A execução mais tardia das tarefas de app_2 também fez com o seu instante de término fosse maior do que aquele obtido no experimento 2. Ao mesmo tempo, o AMS de app_3 detectou a necessidade de executar mais tarefas a partir do instante 139,399 segundos. Esse comportamento foi responsável por aumentar a concorrência pelos recursos entre os instantes 139,399 e 169,089 segundos. O compartilhamento dos recursos de modo mais intenso no último terço das medições fez

com que app_3, última do *workload* a ter sua execução concluída, finalizasse no instante 222,336 segundos.

Tabela 4.4: Resultados dos tempos de execução das três aplicações e do *makespan* do conjunto de aplicações para cada um dos três experimentos

	TE app_1 (s)	TE app_2 (s)	TE app_3 (s)	*Makespan* (s)	*Makespan* (%)
Base	74,982	149,964	224,946	224,946	N/A
Exp. 1	130,128	182,137	231,310	231,310	+2,83
Exp. 2	81,719	154,725	217,805	217,805	-3,17
Exp. 3	80,295	180,359	222,336	222,336	-1,16

Os resultados do *makespan* e dos tempos de execução das aplicações dos três experimentos deste cenário são apresentados na Tabela 4.4. Ao considerar o *makespan* obtido no experimento base para três aplicações, foi possível observar que as diferentes relações entre os valores de **Tempo Alvo** influenciaram na forma como as aplicações executaram e sobre o resultado do *makespan* do *workload* em cada experimento. Observa-se que é possível ajustar os valores de **Tempo Alvo** para alcançar a redução do *makespan*, caso este seja o objetivo do usuário.

Em resumo, os valores de **Tempo Alvo** especificados no Experimento 2 proporcionaram a maior redução do *makespan* do *workload*. Porém, no caso do Experimento 1, mais tarefas das três aplicações foram executadas concorrentemente, o que acarretou no aumento do *makespan* daquele experimento. Em contrapartida, os valores de **Tempo Alvo** do Experimento 3 fizeram com que os AMSs de app_2 e de app_3 não criassem tarefas, pois ambas se encontravam no estado `MUITO ADIANTADO` (2) no início de suas execuções e detectaram o uso intensivo dos recursos de processamento por outra aplicação (app_1). Tal comportamento proporcionou a redução do *makespan* do *workload* do Experimento 3, porém, inferior ao obtido no Experimento 2.

4.3.3.2 Cenário 2: avaliação do impacto da estratégia proposta sobre o *makespan* de um *workload* com quatro aplicações

O cenário 2 foi projetado com o propósito de avaliar o impacto da estratégia de escalonamento proposta sobre o *makespan* de um *workload* composto por quatro aplicações que tiveram suas execuções iniciadas no mesmo instante. Neste cenário, também foram realizados três conjuntos de experimentos. Os valores de **Tempo Alvo** para cada aplicação são apresentados na Tabela 4.5.

Tabela 4.5: Valores de Tempo Alvo adotados para cada uma das quatro aplicações em cada um dos experimentos do Cenário 2

	TA(app_1)	TA(app_2)	TA(app_3)	TA(app_4)
Experimento 1	80	160	240	320
Experimento 2	100	200	300	400
Experimento 3	150	300	450	600

Os valores de **Tempo Alvo** adotados no Experimento 1 foram responsáveis pelo aumento do grau de concorrência entre as aplicações no decorrer dos seus períodos de execução. As aplicações app_1, app_2 e app_3 compartilharam intensamente os recursos durante mais da metade dos seus respectivos tempos de execução, o que causou o aumento dos seus tempos de término. O AMS de app_4, que possuía menor prioridade, intensificou a execução de tarefas apenas a partir do instante 168,681 segundos, quando app_1 e app_2 já estavam próximas de concluir e com menos tarefas em execução. Isso fez com que ela compartilhasse recursos apenas com app_3, de modo a causar um impacto reduzido sobre a sua execução. O comportamento das quatro aplicações impactou no aumento dos seus tempos de execução individuais (Tabela 4.6). Apesar de tal aumento, o comportamento gerenciado pela estratégia proposta proporcionou uma redução de 7,07% do *makespan* do *workload*, cujo valor foi de 278,717 segundos.

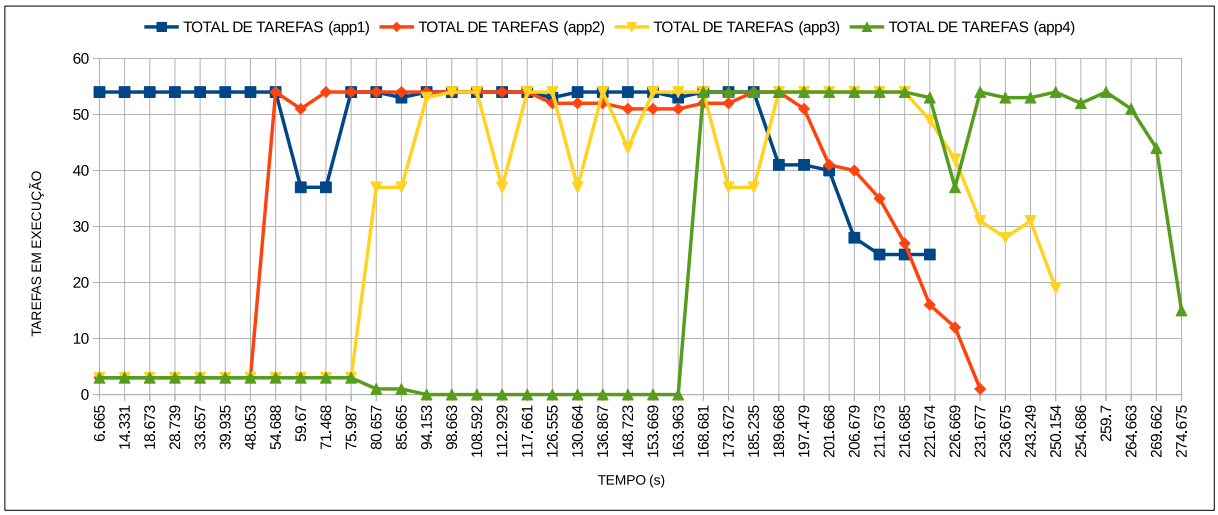


Figura 4.9: Cenário 2 - Experimento 1: quantidade de tarefas em execução para cada uma das quatro aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

Outro ponto que contribuiu para a redução do *makespan* foi a diferença relativamente pequena entre os tempos de término da primeira e da última aplicações. Isso ocorreu devido aos valores de $TA()$ definidos para as três aplicações e aos estados especificados pelos seus AMSs. Embora os valores reduzidos de **Tempo Alvo** tenham causado maior

concorrência entre as aplicações, eles também permitiram que os ciclos ociosos deixados por uma aplicação fossem aproveitados pelas demais, o que permitiu antecipar a execução de tarefas e levou à redução do *makespan*. Essa característica é uma das maiores contribuições da estratégia de escalonamento proposta.

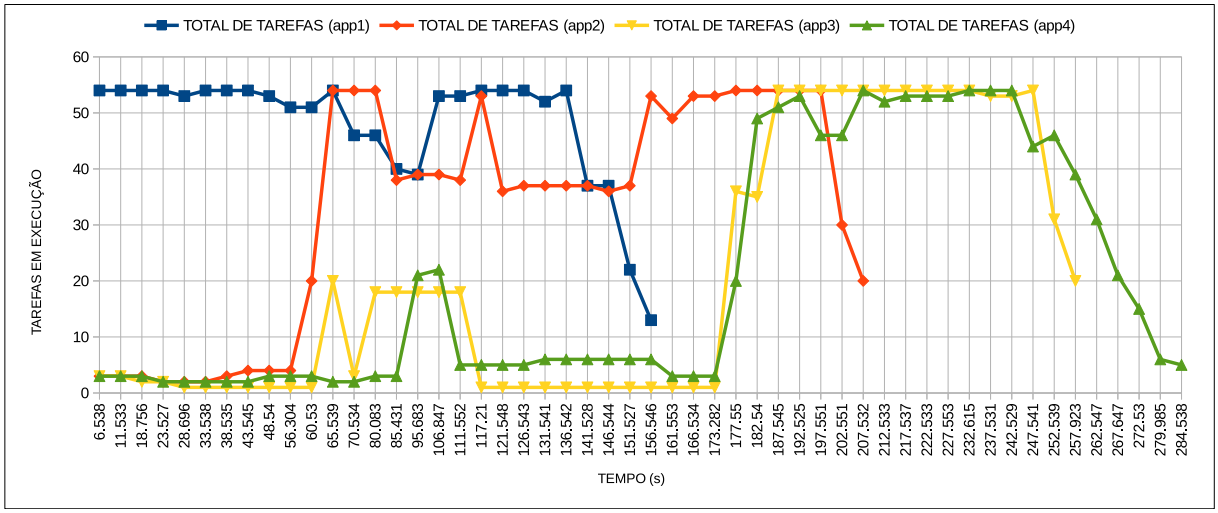


Figura 4.10: Cenário 2 - Experimento 2: quantidade de tarefas em execução para cada uma das quatro aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

Em relação ao Experimento 2, em que os valores de $TA()$ não são tão curtos nem tão longos em relação aos tempos de execução base de cada aplicação, os comportamentos apresentados pelas aplicações durante a sua realização (Figura 4.10) mostram que as aplicações concorreram aos pares pelos recursos compartilhados. A primeira metade do período de execução do conjunto foi marcado pela concorrência mais intensa entre app_1 e app_2, devido aos seus valores de **Tempo Alvo**. Esses valores fizeram com que ambos os AMSs intensificassem a criação de tarefas a partir da metade da execução da aplicação 1. Ao mesmo tempo, os valores mais altos de **Tempo Alvo** levaram os AMSs de app_3 e de app_4 a manter uma baixa vazão de execução de tarefas enquanto app_1 e app_2 compartilhavam os recursos. Contudo, é possível observar um período, entre os instantes 65,539 e 111,52 segundos, no qual os AMSs de app_3 e de app_4 detectaram e se aproveitaram da baixa utilização em um dos 3 *hosts*. A partir do instante 177,55 segundos, o AMS de app_3 detectou o estado `ATRASADO COM CHANCE` (5), que o fez explorar o limite máximo de criação de tarefas. Simultaneamente, o AMS de app_4 detectou a redução de utilização dos recursos e também começou a intensificar a execução de tarefas e, posteriormente, também detectou o estado `ATRASADO COM CHANCE`. Esses comportamentos fizeram com que elas compartilhassem intensamente os recursos até o término de app_3.

A execução do *workload* conforme as especificações do experimento 2 acarretou em

um *makespan* de 287,138 segundos, 4,26% inferior aos 299,928 segundos obtidos com a execução do experimento base (Tabela 4.6). Ao comparar os resultados dos experimentos 1 e 2, foi possível perceber que, para diferenças maiores entre os valores de **Tempo Alvo** das aplicações (no Experimento 1 a diferença entre os $TA()$s é de 80 segundos, enquanto no Experimento 2 é de 100 segundos) houve a ocorrência de um aumento da diferença entre os tempos de término da primeira e da última aplicações a terem suas execuções concluídas. Isso reduziu a concorrência por recursos e causou um pequeno aumento do *makespan* do *workload*.

Os resultados dos Experimentos 1 e 2 reforçam a hipótese de que o aumento da diferença relativa entre os valores de **Tempo Alvo** das aplicações acarretam no aumento do *makespan* do *workload* com quatro aplicações. O Experimento 3 foi elaborado com finalidade de corroborar essa hipótese. Para isso, foi adotada uma configuração na qual a diferença relativa entre os valores de **Tempo Alvo** é igual a 150 segundos. Além disso, o **Tempo Alvo** de app_1, que possui a maior prioridade do conjunto, foi definido com 150 segundos. Os comportamentos das 4 aplicações durante suas execuções podem ser vistos no gráfico da Figura 4.11.

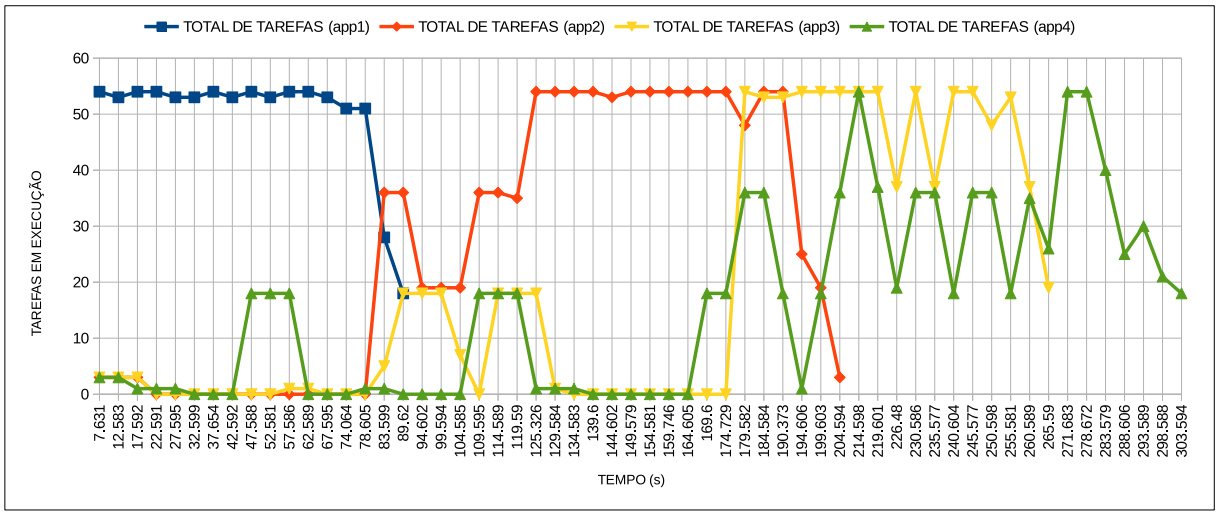


Figura 4.11: Cenário 2 - Experimento 3: quantidade de tarefas em execução para cada umas das quatro aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

Os valores de **Tempo Alvo** adotados no Experimento 3 causaram, em geral, a redução da concorrência entre as aplicações. É possível visualizar claramente períodos de tempo nos quais as aplicações utilizam os recursos de forma dedicada ou com baixo grau de competição. Os comportamentos descritos ocorreram em razão de haver tempo suficiente para que as aplicações menos urgentes esperassem, enquanto as mais urgentes executavam. Todavia, esses comportamentos não proporcionaram um resultado tão sa-

tisfatório em relação ao *makespan* do *workload*. Para este experimento, o *makespan* foi 306,288 segundos. Esse valor representa um aumento de 2,12% em relação ao experimento base. Os resultados de *makespan* do *workload* e dos tempos de execução das aplicações dos três experimentos do cenário 2 estão sintetizadas na Tabela 4.6.

Tabela 4.6: Resultados dos tempos de execução das quatro aplicações e do *makespan* do conjunto de aplicações para cada um dos três experimentos

	TE app1 (s)	TE app2 (s)	TE app3 (s)	TE app4 (s)	*Makespan* (s)	*Makespan* (%)
Base	74,982	149,964	224,946	299,928	299,928	N/A
Exp. 1	225,227	231,836	252,227	278,717	278,717	-7,07
Exp. 2	160,506	212,993	260,069	287,138	287,138	-4,26
Exp. 3	98,352	209,049	273,288	306,288	306,288	+2,12

Neste cenário, foi observado que reduzir a diferença entre os valores de **Tempo Alvo** das aplicações acarreta em reduzir o *makespan* do conjunto de aplicações. As prioridades definidas no Experimento 1 fez com que os AMSs das quatro aplicações criassem mais tarefas simultaneamente durante mais da metade do tempo de execução. Ao comparar os resultados entre os cenários 1 e 2, percebe-se que o aumento da quantidade de aplicações remete à redução das prioridades e ao aumento da concorrência entre elas, caso o objetivo seja reduzir o *makespan*. O cenário 3 foi formulado com o intuito de averiguar se tais afirmações são verdadeiras também com cinco aplicações que compartilham recursos.

4.3.3.3 Cenário 3: avaliação do impacto da estratégia proposta sobre o *makespan* de um *workload* com cinco aplicações

O cenário 3 foi elaborado com a finalidade de avaliar o impacto da estratégia de escalonamento proposta sobre o *makespan* de um *workload* composto por cinco aplicações. Foram elaborados três experimentos, nos quais as variações entre os valores de $TA()$ seguiram o mesmo padrão adotado nos cenário 1 e 2. Assim, houve variação de 80 segundos para cada aplicação no Experimento 1, 100 segundos no Experimento 2 e 150 segundos no Experimento 3. Os valores de $TA(app_i)$ utilizados para todas as aplicações em cada um dos experimentos são apresentados na Tabela 4.7.

O Experimento 1 contemplou a execução do conjunto mediante a adoção de valores de **Tempo Alvo** múltiplos de 80 segundos, sendo esse o valor definido para a aplicação 1. Ao tomar como base o ocorrido nos cenários 1 e 2, era esperada uma concorrência intensa entre as aplicações, devido à pequena diferença entre os seus valores de **Tempo Alvo**. Contudo, tornou-se necessário verificar o efeito dessa concorrência sobre o *makespan*. Como pode ser

Tabela 4.7: Valores de Tempo Alvo adotados para cada uma das cinco aplicações em cada um dos experimentos do Cenário 3

	TA(app_1)	TA(app_2)	TA(app_3)	TA(app_4)	TA(app_5)
Experimento 1	80	160	240	320	400
Experimento 2	100	200	300	400	500
Experimento 3	150	300	450	600	750

observado por meio das informações contidas na Tabela 4.8, a execução do Experimento 1 resultou em um *makespan* de 357,519 segundos, o que representa uma redução de 4,64% em relação ao experimento base com cinco aplicações. Os comportamentos de cada uma das cinco aplicações que proporcionaram essa redução do makespan são apresentados no gráfico da Figura 4.12.

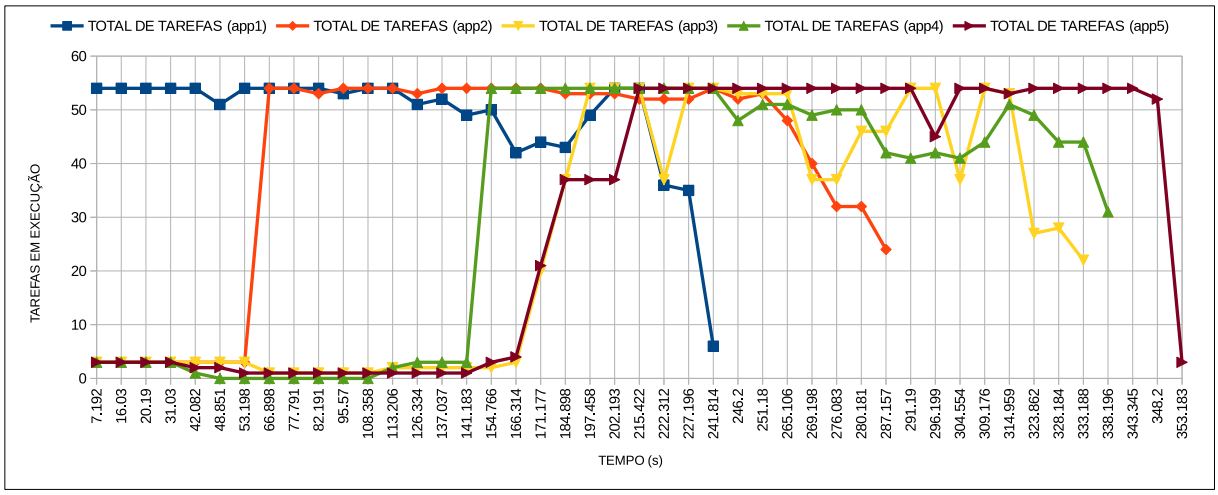


Figura 4.12: Cenário 3 - Experimento 1: quantidade de tarefas em execução para cada uma das cinco aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

Os AMSs das aplicações app_2, app_3, app_4 e app_5 detectaram a ocorrência de estados de atraso enquanto pelo menos uma outra aplicação encontrava-se em execução. Esse comportamento ocorreu em virtude dos valores de **Tempo Alvo** definidos. Assim, ao menos três das cinco aplicações concorreram pelos recursos de maneira mais intensa durante 48,6% do tempo total necessário para concluir a execução do *workload*. No entanto, a concorrência mais intensa pelos recursos, causado por valores reduzidos de **Tempo Alvo**, mostrou-se importante para obter a maior redução do *makespan* no cenários 3.

Para o Experimento 2, foram considerados valores de **Tempo Alvo** múltiplos de 100 segundos. Esses valores foram utilizados com o intuito de reduzir a prioridade das aplicações, em relação àquelas definidas no Experimento 1. Com isso, esperava-se reduzir a intensidade da concorrência entre as aplicações. Os valores de **Tempo Alvo** deste

experimento resultaram em um *makespan* de 365,316 segundos, 2,65% inferior àquele obtido no Experimento Base. Porém, houve um aumento de aproximadamente 8 segundos em relação ao *makespan* do Experimento 1. Esse aumento foi causado pelo aumento da relação entre os valores de **Tempo Alvo**. Os comportamentos de cada uma das cinco aplicações que levaram ao *makespan* citado são apresentados no gráfico da Figura 4.13.

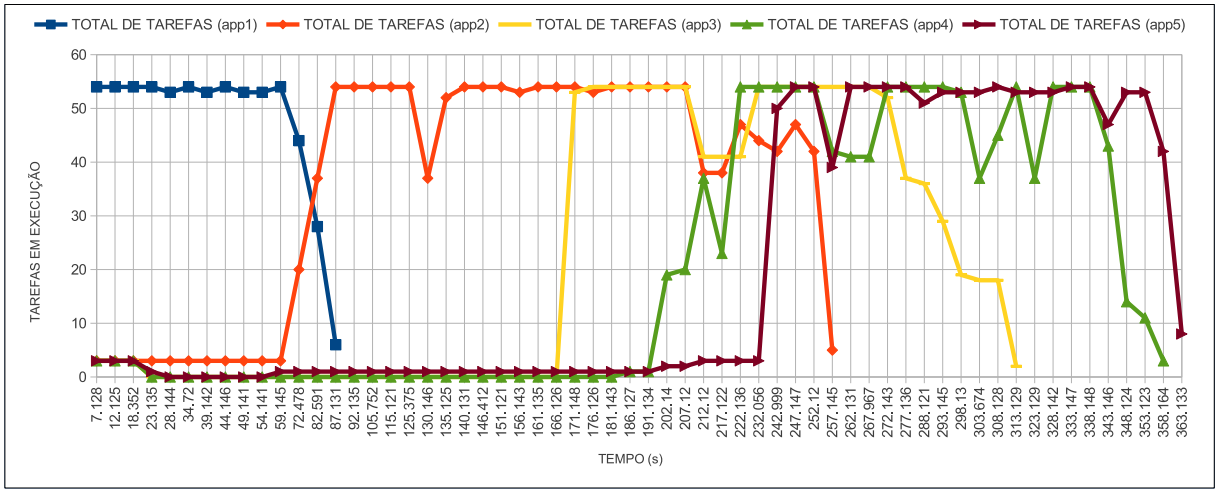


Figura 4.13: Cenário 3 - Experimento 2: quantidade de tarefas em execução para cada uma das cinco aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

A adoção de valores de **Tempo Alvo** múltiplos de 100 segundos para as aplicações causou menor concorrência pelo uso dos recursos na primeira metade da execução do experimento. Até o instante 212,12 segundos, observa-se que apenas app_2 e app_3 concorreram pelos recursos. A partir deste instante, também começou a haver o compartilhamento de recursos com app_4 e, posteriormente, com app_5. Os comportamentos das aplicações são justificados pela detecção de estados de atraso por parte dos seus AMSs. A intensa concorrência por recursos persistiu até o fim da execução de cada uma das aplicações restantes. Logo, é possível assumir que a redução das prioridades implicou na execução mais flexível das aplicações. Porém, tal comportamento não resultou em redução do *makespan*, quando comparado ao Experimento 1.

Para a realização do Experimento 3, foram adotados valores de **Tempo Alvo** múltiplos de 150 segundos para as aplicações. O objetivo foi averiguar a consequência do uso desses valores sobre o *makespan* do *workload*. Após a realização do experimento, foi obtido um *makespan* de 385,607 segundos. Tal resultado faz do Experimento 3 aquele que obteve o pior resultado do cenário 3, sendo 2,85% maior do que o obtido com o Experimento Base.

Por meio da análise dos comportamentos das aplicações no Experimento 3, apresen-

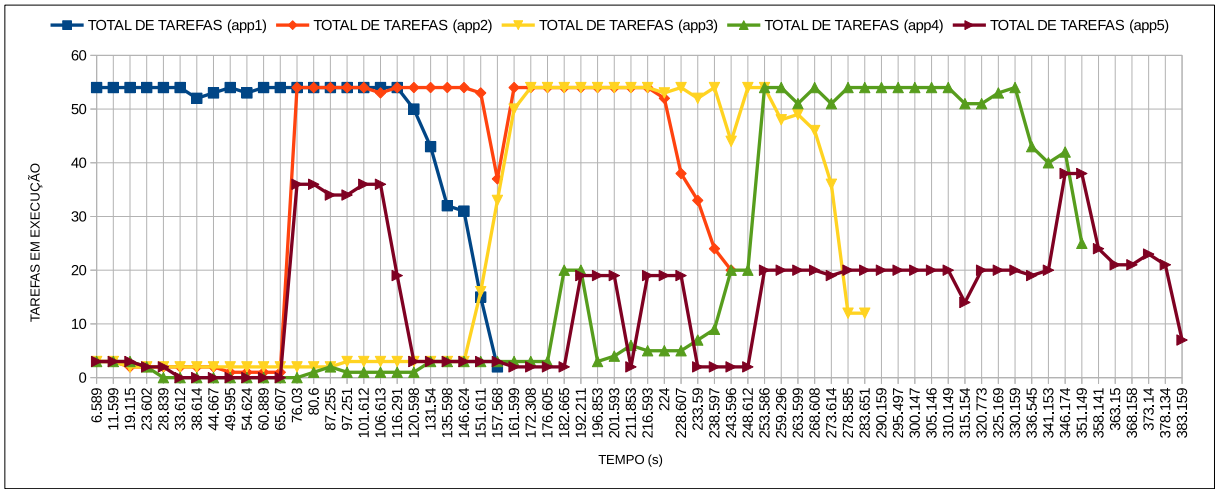


Figura 4.14: Cenário 3 - Experimento 3: quantidade de tarefas em execução para cada uma das cinco aplicações com 512 tarefas executando em 3 *hosts* com 12 núcleos cada

tados no gráfico da Figura 4.14, é possível entender a razão pela qual houve o aumento do *makespan*. O diferença de 150 segundos dos valores de **Tempo Alvo** das aplicações permitiu que o *workload* apresentasse uma execução com menor concorrência por recursos de processamento. Observa-se que os AMSs das aplicações app_1, app_2, app_3 e app_4 executaram tarefas com baixo grau de concorrência. Em contrapartida, as tarefas de app_5 foram distribuídas durante todo o período de execução do *workload*. Tendo em vista os comportamento apresentados, conclui-se que a execução mais dispersa do conjunto, em relação aos demais experimentos deste cenário, foi responsável pelo aumento do *makespan*.

Os resultados dos experimentos do cenário 3 (Tabela 4.8) atestam para uma maior redução do *makespan*, causado pela redução dos valores de **Tempo Alvo** das aplicações. Tais resultados confirmam aqueles apresentados pelos experimentos do cenário 2. Portanto, constata-se uma relação clara entre o aumento das prioridades, o aumento da concorrência entre as aplicações e a redução do *makespan* do *workload*. No entanto, os valores de **Tempo Alvo** não devem ser reduzidos abaixo do tempo de execução de uma aplicação em uma situação na qual não há concorrência (74,982 segundos para a aplicação utilizada). Isso causaria a concorrência indiscriminada por recursos e invalidaria o escalonamento baseado em prioridades.

Por fim, os resultados dos experimentos do cenário 3 mostram que é possível adotar a estratégia de escalonamento proposta para reduzir o *makespan* de um *workload* composto por cinco aplicações. No entanto, é necessário ajustar corretamente os valores de **Tempo Alvo** de cada aplicação. Ademais, a estratégia de escalonamento considera a concorrência das aplicações pelo uso dos recursos, seja em maior ou em menor grau. Portanto, obteve-

Tabela 4.8: Resultados dos tempos de execução das cinco aplicações e do *makespan* do conjunto de aplicações para cada um dos três experimentos

	TE app1 (s)	TE app2 (s)	TE app3 (s)	TE app4 (s)	TE app5 (s)	*Makespan* (s)	*Makespan* (%)
Base	74,982	149,964	224,946	299,928	374,910	374,910	N/A
Exp. 1	235,349	290,896	336,021	343,019	357,519	357,519	-4,64
Exp. 2	87,241	263,878	315,472	360,396	365,316	365,316	-2,65
Exp. 3	158,129	246,148	285,113	355,661	385,607	385,607	+2,85

se um melhor aproveitamento da infraestrutura disponível. O resumo dos resultados dos experimentos com o foco na redução do makespan e as conclusões gerais a respeito deles são apresentados na Subseção 4.3.3.4.

4.3.3.4 Conclusões sobre o impacto da estratégia de escalonamento proposta sobre o *makespan* do *workload*

Os resultados dos experimentos sintetizados na Tabela 4.9 mostram que, para cada um dos cenários, foram registradas pelo menos duas situações nas quais houve redução do *makespan*. Somado a isso, está o fato de que o aumento do *makespan* não ultrapassou 3% em nenhum dos experimentos. Ao considerar o ganho obtido com a redução ou eliminação da ociosidade dos recursos, concluiu-se que a estratégia proposta apresenta resultados satisfatórios, da perspectiva do *makespan* do *workload*.

Tabela 4.9: Síntese dos resultados dos experimentos de avaliação do impacto da estratégia de escalonamento proposta sobre o *makespan* do *workload*

		makespan (s)	*makespan* (%)
Cenário 1	Exp. 1	231,310	+2,83
	Exp. 2	217,805	-3,17
	Exp. 3	222,336	-1,16
Cenário 2	Exp. 1	278,717	-7,07
	Exp. 2	287,138	-4,26
	Exp. 3	306,288	+2,12
Cenário 3	Exp. 1	357,519	-4,64
	Exp. 2	365,316	-2,65
	Exp. 3	385,607	+2,85

O uso de valores menores de **Tempo Alvo** (prioridades altas) causou o aumento do makespan no cenário 1 (três aplicações). Como consequência, a competição elevada

resultou na conclusão tardia das aplicações. No entanto, foi registrada a redução do *makespan* para os cenários 2 e 3 com esta mesma definição de prioridades. A explicação encontra-se no maior tempo de execução do *workload*, causado pela redução da competição entre as aplicações. Esse comportamento é observado no Experimento 1 do cenário 2, no qual a estratégia tirou melhor proveito da concorrência entre as aplicações. Nele, foi registrada uma redução de 7,07% do *makespan*, em relação ao Experimento Base, a maior redução dentre todos os experimentos.

4.4 Adoção da estratégia proposta para resolver um problema comum de escalonamento

Os experimentos apresentados nesta seção foram elaborados com a finalidade de avaliar o uso da estratégia proposta para resolver um problema de escalonamento comum em ambientes de HPC e em supercomputadores: reduzir a subutilização dos recursos de processamento por meio do compartilhamento, ao mesmo tempo em que o *makespan* do *workload* também é reduzido. Além disso, os experimentos também foram utilizados para avaliar o efeito da estratégia sobre o tempo médio de *turnaround* do *workload*, com o propósito de verificar se os resultados associados a essa métrica representam benefícios ou restrições quanto ao uso da estratégia.

Para realizar os experimentos, uma típica situação de escalonamento de aplicações foi recriada. Nela, cada aplicação requer apenas um subconjunto dos recursos disponíveis. No entanto, quando a primeira aplicação tem sua execução iniciada, as outras devem aguardar enquanto os seus requisitos de hardware não são completamente satisfeitos, mesmo que parte dos recursos esteja disponível. Este conjunto de experimentos mostra o quanto eficiente é a estratégia proposta para resolver este problema e como usuários e provedores de serviço podem se beneficiar com a sua adoção.

O *workload* usado nestes experimentos é composto por duas aplicações app_1 e app_2. Cada uma delas requer dois dos três *hosts* disponíveis no ambiente para executar (24 núcleos de um total de 36). Assim, foi elaborado um cenário base no qual as aplicações são executadas uma após a outra e sem o uso do EasyGrid AMS, como seria feito por um sistema escalonador convencional. Em seguida, foram elaborados dois cenários diferentes, nos quais foram atribuídos diferentes valores de **Tempo Alvo** para as aplicações. Os resultados foram analisados e o *makespan* e o tempo médio de *turnaround* dos *workloads* foram comparados. É importante lembrar que o tempo de espera na fila das aplicações que

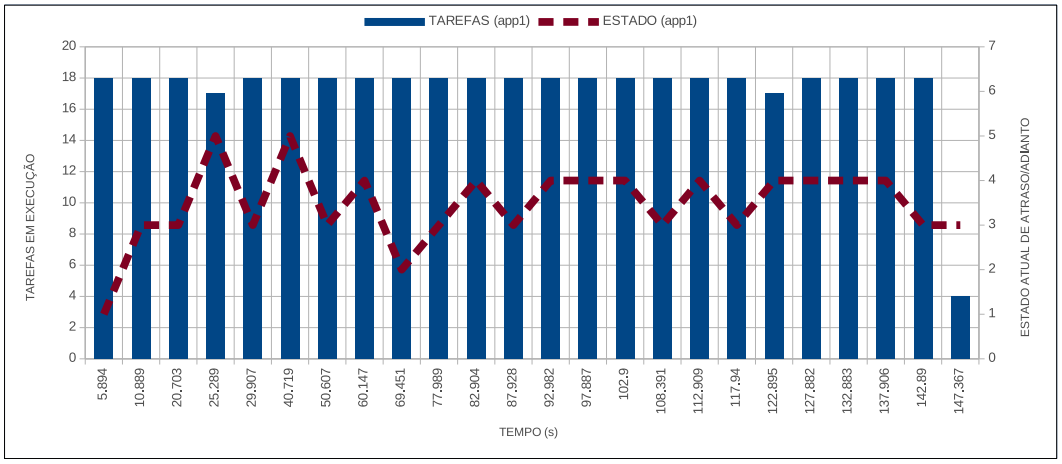
utilizam a estratégia proposta é igual a zero. Isso se deve ao fato de que não é necessário que elas aguardem a disponibilidade de todos os recursos necessários para iniciar suas execuções de forma exclusiva.

Com o objetivo de obter um valor base para avaliar a estratégia, foram realizadas 30 execuções de um *workload* composto por apenas uma aplicação sem o uso do EasyGrid AMS. As execuções resultaram em um tempo médio de execução igual a 112,87 segundos, com um intervalo de confiança de 95%. A partir do resultado obtido, foi considerado um *makespan* teórico base de 225,74 segundos para um *workload* com duas aplicações. Além disso, foi considerado um tempo médio de *turnaround* igual a 169,31 segundos para o mesmo *workload*, uma vez que app_2 obteve um tempo de espera em fila igual a 112,87 segundos antes de ter a sua execução iniciada.

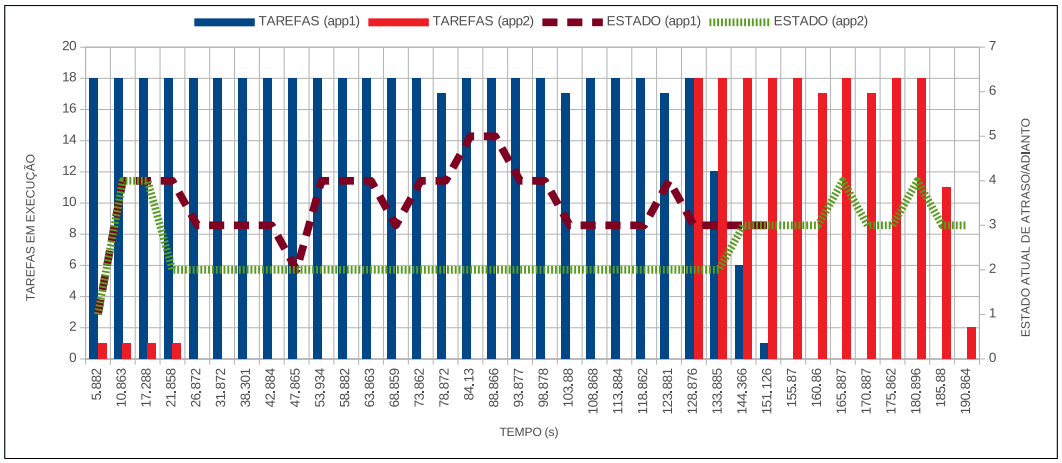
O *workload* do primeiro cenário foi composto por duas aplicações. Cada uma delas requer dois *hosts* (24 núcleos) para executar. A estratégia de escalonamento proposta foi utilizada com o objetivo de permitir que elas pudessem compartilhar um dos *hosts* e, com isso, executar simultaneamente. Os valores de **Tempo Alvo** das aplicações foram $TA(app_1) = 150$ segundos e $TA(app_2) = 240$ segundos. A intenção de usar esses valores de **Tempo Alvo** foi atribuir um curto para app_1 e um longo para app_2. A Figura 4.15 mostra o comportamento de cada aplicação durante os períodos em que elas estiveram em execução nos *hosts* que elas utilizaram.

A aplicação app_1 iniciou sua execução nos *hosts* Dorado e Hanamura ao mesmo tempo em que app_2 iniciou sua execução nos *hosts* Hanamura e Hollywood. O AMS de app_1 detectou baixa utilização de CPU em ambos os *hosts* começou a criar novas tarefas até o limite máximo de 18. Como o AMS de app_1 foi capaz de executar o máximo de tarefas desde o início de sua execução, o estado `ATRASADO COM CHANCE` (5) foi detectado apenas em três medições. Além disso, o $TA(app_1)$ estabeleceu maior prioridade para app_1 utilizar o *host* compartilhado.

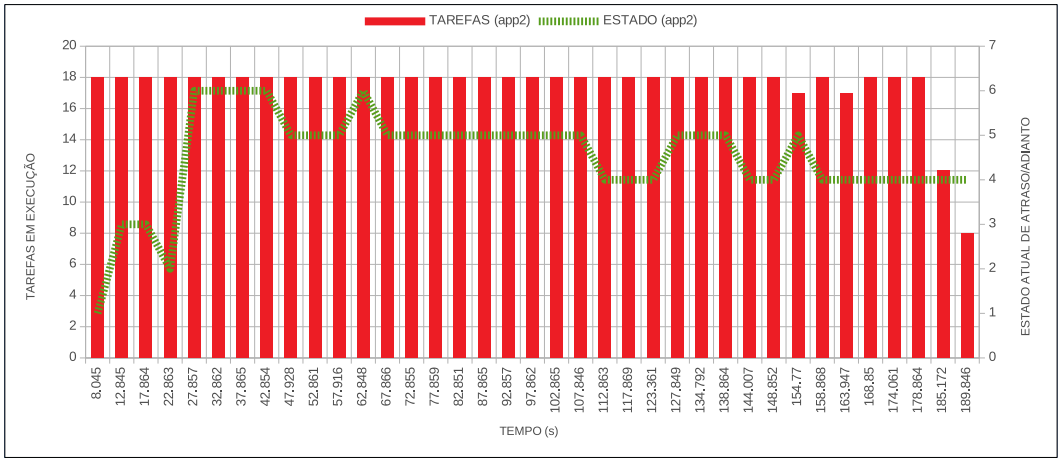
Ao mesmo tempo, o AMS de app_2 detectou o uso elevado de CPU no *host* Hanamura, o que fez com que apenas uma tarefa fosse criada a cada medição em que ela permaneceu no estado `INDEFINIDO` (1). Em seguida, o estado foi trocado brevemente para `NO TEMPO` (4) antes de mudar para `MUITO ADIANTADO` (2), no qual ele permaneceu até que fosse detectado o fim da execução de app_1 e a liberação dos recursos usados por ela. O comportamento de espera adotado por app_2 no *host* Hanamura fez com que o seu AMS reescalonasse dinamicamente as suas tarefas para o *host* Hollywood. Isso fez com o que o AMS de app_2 no *host* Hollywood ficasse em estados críticos de atraso durante a maior



(a) Quantidade de tarefas em execução e variação de estados de atraso/adianto de app_1 no *host* Dorado



(b) Quantidade de tarefas em execução e variação de estados de atraso/adianto de ambas as aplicações no *host* Hanamura



(c) Quantidade de tarefas em execução e variação de estados de atraso/adianto de app_2 no *host* Hollywood

Figura 4.15: Escalonamento de duas aplicações, sendo $TA(app_1) = 150$ segundos (curto) e $TA(app_2) = 240$ segundos (longo), cada uma executando 512 tarefas e compartilhando apenas o *host* Hanamura

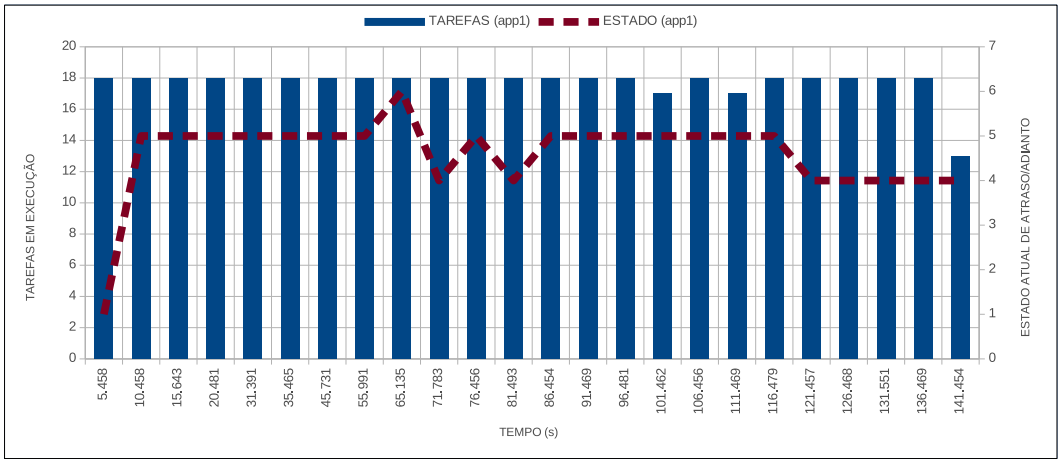
parte da execução. Contudo, este mesmo comportamento foi o responsável por garantir que não houvesse subutilização dos recursos disponíveis.

Os tempos médios de execução registrados para o primeiro cenário, após 30 execuções, foram 140,01 segundos para app_1 e 194,06 segundos para app_2. O tempo de execução de app_2 também representa o *makespan* do *workload*, enquanto o seu tempo médio de *turnaround* foi igual a 167,04 segundos. Ao comparar os resultados obtidos com a estratégia de escalonamento proposta com aqueles do cenário base, observa-se uma redução de 31,68 segundos (14%) do *makespan*. O tempo médio de *turnaround* também foi reduzido em 2,27 segundos (1,34%). É importante destacar que as reduções citadas foram alcançadas mesmo com o compartilhamento de um dos *hosts*, o que garantiu significativamente o período de ociosidade dos recursos do ambiente.

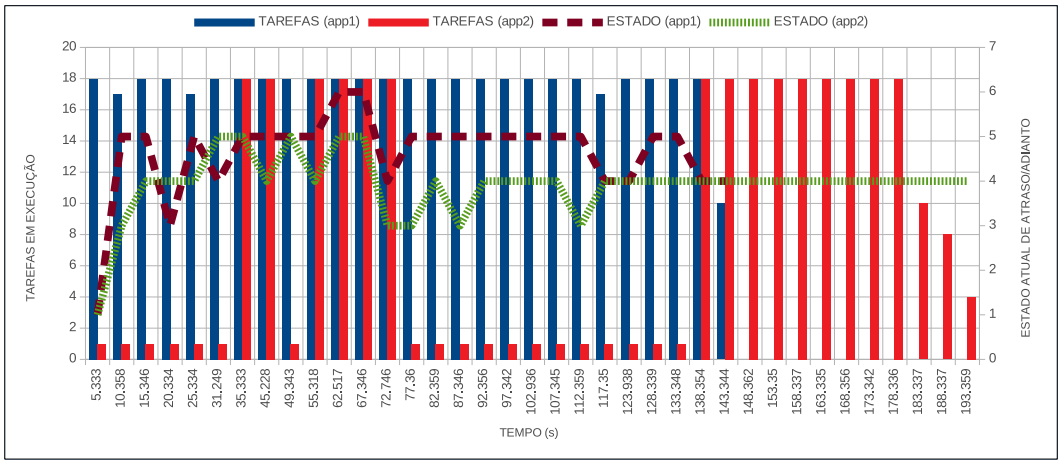
O segundo cenário foi projetado com a finalidade de aprofundar a avaliação da estratégia proposta quando apenas um subconjunto dos recursos é compartilhado. Assim, diferentes valores foram definidos, tal que $(app_1) = 180$ segundos e $TA(app_2) = 210$ segundos. A ideia foi atribuir um **Tempo Alvo** mais longo para app_1 e um mais curto para app_2, quando comparados ao primeiro cenário e analisar a influência desta mudança sobre o *makespan* e o tempo médio de *turnaround* do *workload*. A Figura 4.16 mostra os comportamentos de ambas as aplicações nos hosts que elas utilizaram para executar.

O **Tempo Alvo** mais curto de app_2 fez o seu AMS detectar o estado `ATRASADO COM CHANCE` (5) no *host* compartilhado Hanamura nos primeiros 35 segundos de execução, o que não havia ocorrido no primeiro cenário. Com isso o AMS de app_2 foi forçado a criar e executar tarefas até o limite máximo permitido, apesar do AMS de app_1 também o estar fazendo. No entanto, o **Tempo Alvo** mais longo de app_1 permitiu que o maior nível de concorrência imposto por app_2 não comprometesse a sua execução. Isso é constatado ao verificar que o AMS de app_1 detectou o estado `ATRASADO COM POUCA CHANCE` (6) em apenas três dentre todas as medições realizadas nos dois *hosts*. A partir da segunda metade da execução de app_1, o AMS de app_2 detectou o estado `NO TEMPO` (4), o que o permitiu reduzir a quantidade de tarefas criadas para apenas uma. Esse comportamento persistiu até o AMS app_2 detectar a liberação dos recursos ocupados por app_1 e executar o restante das suas tarefas.

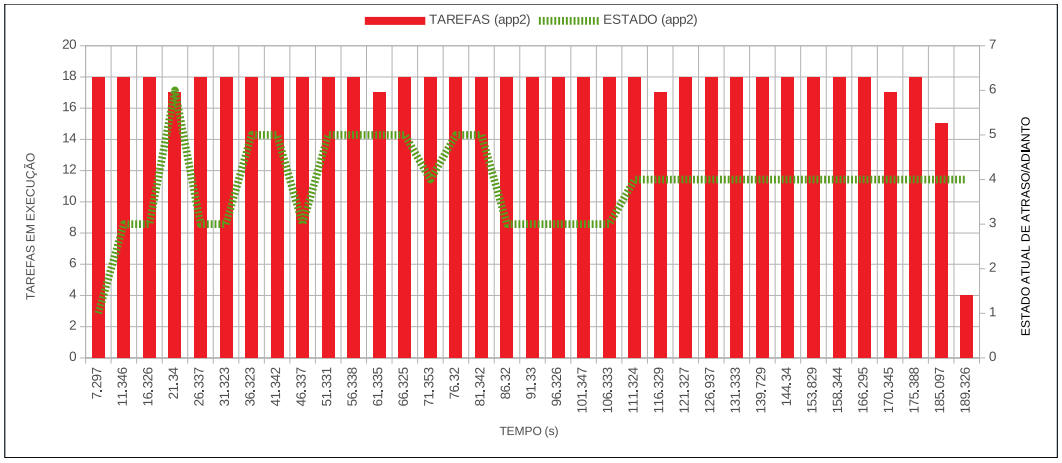
O escalonamento das aplicações do *workload* no segundo cenário resultaram em tempos de execução iguais a 143,69 segundos para app_1 e 191,39 para app_2. O tempo de execução de app_2 também representa o *makespan* do *workload* 34,35 segundos (15,2%) menor do que o obtido no cenário base. Os resultados obtidos levaram ainda a um tempo



(a) Quantidade de tarefas em execução e variação de estados de atraso/adianto de app_1 no *host* Dorado



(b) Quantidade de tarefas em execução e variação de estados de atraso/adianto de ambas as aplicações no *host* Hanamura



(c) Quantidade de tarefas em execução e variação de estados de atraso/adianto de app_2 no *host* Hollywood

Figura 4.16: Escalonamento de duas aplicações, sendo $TA(app_1) = 180$ segundos (menos curto) e $TA(app_2) = 210$ segundos (menos longo), cada uma executando 512 tarefas e compartilhando apenas o *host* Hanamura

médio de *turnaround* igual a 169,04 segundos, uma redução de 0,27 segundos em relação ao cenário base, o que mostra que a adoção da estratégia proposta não causa o aumento desta métrica. Novamente, as reduções foram alcançadas reduzindo significativamente a ociosidade do ambiente, uma vez que todos os recursos foram utilizados durante toda a execução do *workload*.

Tabela 4.10: Resumo dos resultados do escalonamento de duas aplicações compartilhando apenas 1 dos 3 *hosts*: *makespan* e tempo médio de *turnaround*

	Makespan (segundos)	Tempo médio de *Turnaround* (segundos)
Exp. Base	225,74	169,31
Cenário 1	194,06	167,04
Cenário 2	191,39	169,04

Este conjunto de experimentos mostrou que a estratégia de escalonamento proposta foi capaz de reduzir o *makespan* do *workload*. Em relação ao tempo médio de *turnaround*, foi observado que não houve aumento em nenhum dos cenários, embora as reduções tenham sido pequenas.

O *workload* foi executado em dois cenários distintos no quais os valores de **Tempo Alvo** das aplicações foram variados, o que foi feito para avaliar o comportamento da execução das aplicações em diferentes situações. Embora algumas das reduções alcançadas tenham sido pequenas, deve-se ressaltar que o *workload* foi composto por apenas duas aplicações. Logo, novos experimentos que considerem *workloads* e conjuntos de recursos maiores devem ser executados, a fim de verificar os resultados da estratégia proposta em ambientes com maior concorrência por recursos. Os resultados obtidos com este conjunto de experimentos estão sintetizados na Tabela 4.10.

Por fim, é importante destacar que os resultados alcançados com a adoção da estratégia de escalonamento proposta são relevantes para diferentes grupos de interesse. Para os usuários das aplicações, as reduções de *makespan* e do tempo médio de *turnaround* são importantes, uma vez que ambos atuam na redução do tempo de obtenção do resultado das aplicações. Já do ponto de vista dos provedores de serviço e gestores dos recursos computacionais, o aumento da taxa de utilização da infraestrutura é importante ao permitir que a capacidade de atendimento seja aumentada, o que torna possível atender a um maior número de usuários simultaneamente.

4.5 Síntese do capítulo de resultados

Os experimentos e resultados apresentados neste capítulo tiveram a finalidade de validar a estratégia de escalonamento proposta. A validação da estratégia teve início com a comprovação de que ela é capaz de realizar diferentes escalonamentos de um mesmo *workload*, apenas por meio do ajuste do parâmetro **Tempo Alvo** das suas aplicações. Tais ajustes levam os AMSs das aplicações a adotarem comportamentos específicos em situações nas quais elas compartilham recursos de processamento. Foi observado que a redução do **Tempo Alvo** implica em criar e executar uma maior quantidade de tarefas da aplicação, ou seja, aumentar a sua prioridade. As diferentes configurações de **Tempo Alvo** das aplicações permitiu manipular os seus comportamentos, de modo a alcançar os diferentes resultados de escalonamento.

A partir dos experimentos que resultaram em diferentes escalonamentos, foram identificados alguns padrões de comportamento das aplicações, que variam em função dos valores distintos de **Tempo Alvo** adotados. A partir desses padrões, foram escolhidos valores de **Tempo Alvo** para avaliar o impacto da estratégia proposta sobre o *makespan* do *workload* em diferentes situações de escalonamento. Foi observado, por exemplo, que valores diferentes de **Tempo Alvo** tendem a reduzir o *makespan*, enquanto valores próximos ou iguais causam o aumento da concorrência entre as aplicações e, como consequência, uma maior utilização dos recursos do ambiente.

O escalonamento realizado mostrou-se eficiente no que tange à redução do *makespan* do *workload*, mesmo quando as suas aplicações compartilham os recursos de processamento. Isso ocorreu em dois terços dos experimentos realizados. Nos experimentos nos quais as aplicações compartilharam todos os recursos, a maior redução alcançada foi de 7,07% em relação ao experimento base. Essa redução é considerada significativa, uma vez que ela foi comparada com o *makespan* teórico gerado por estratégias de alocação exclusiva de recursos, que são adotadas pela maioria dos sistemas de escalonamento tradicionais.

Os experimentos nos quais as aplicações compartilharam apenas parte dos recursos disponíveis apresentaram resultados expressivos. Foi possível alcançar reduções tanto do *makespan* do *workload*, como do seu tempo médio de *turnaround*. Além disso, todos os *hosts* foram utilizados durante todo o período de execução do *workload*. Esses resultados são muito significativos para os usuários que desejam obter os resultados de suas aplicações em menor tempo. Além disso, os provedores de serviços que almejam aumentar a taxa de utilização de suas infraestruturas também podem beneficia-se da estratégia, uma vez

que ela permite atender a uma maior número de usuários simultâneos com padrões de qualidade iguais ou ainda melhores.

Capítulo 5

Considerações finais

As aplicações científicas computacionais exercem um papel importante no constante desenvolvimento social e tecnológico da humanidade. Essas aplicações geralmente processam elevadas quantidades de dados e resolvem cálculos complexos, o que pode resultar em longos períodos de execução. Portanto, o uso de infraestruturas computacionais de alto desempenho, como supercomputadores e *data centers*, torna-se fundamental para satisfazer as demandas geradas por essas aplicações.

A alocação de um conjunto finito de recursos computacionais necessários para suprir as demandas de aplicações com diferentes características é uma tarefa complexa. Tal complexidade se deve à quantidade e à heterogeneidade dos recursos, à possibilidade de compartilhá-los e à suscetibilidade a falhas. Estratégias baseadas em *Job Scheduling* foram desenvolvidas e são amplamente adotadas por sistemas escalonadores tradicionais. Abordagens como essas possuem o objetivo principal de reduzir o *makespan* das aplicações que são executadas. O objetivo é alcançado por meio do uso exclusivo de um conjunto de recursos por uma aplicação enquanto ela se encontra em execução, o que acarreta na subutilização da infraestrutura. Além disso, todo o trabalho de alocação de recursos para atender às demandas das aplicações é realizado, geralmente, por um único escalonador centralizado. Isso pode causar atrasos devido a sobrecarga de processamento realizado por ele, além de tornar o sistema mais sujeito a falhas.

Todo o trabalho apresentado nesta tese teve como objetivo maior, propor um diferente paradigma de escalonamento, no qual aplicações paralelas realizassem o próprio escalonamento, de maneira a compartilhar os recursos de processamento disponíveis nos ambientes utilizados por elas. O resultado obtido foi uma estratégia de escalonamento, na qual aplicações autonômicas adotam um conjunto de políticas de comportamento e agem como uma sociedade. Assim, cada aplicação é capaz de alterar o próprio comportamento

de acordo com o tempo que ainda resta para ela executar e com as características dos recursos compartilhados. Somado a isso, está o fato de que a estratégia proposta permite alcançar diferentes resultados de escalonamento para um mesmo *workload* por meio do ajuste de apenas uma parâmetro. A criação e a validação da estratégia de escalonamento proposta por meio da análise dos resultados alcançados com a sua adoção, proporcionaram as seguintes contribuições:

- O desenvolvimento de uma estratégia de escalonamento descentralizada, na qual as aplicações autonômicas compartilham os recursos de processamento do ambiente computacional nos quais elas executam. O compartilhamento de recursos possibilita a execução concorrente das aplicações, o que resulta no melhor aproveitamento da infraestrutura disponível.

- O desenvolvimento de uma estratégia capaz de escalonar um mesmo *workload* de formas diferentes, o que pode ser alcançado por meio do simples ajuste dos valores de **Tempo Alvo** de cada aplicação.

- A adoção do EasyGrid AMS e do seu respectivo modelo de programação, com a finalidade de permitir que as próprias aplicações gerenciem os seus comportamentos. Além disso, tirou-se proveito de aspectos intrínsecos a ele, como o escalonamento dinâmico de tarefas e a tolerância a falhas.

- A especificação de um conjunto de comportamentos, aliada ao escalonamento autonômico, permite que as aplicações aumentem ou diminuam dinamicamente a quantidade de tarefas que devem ser executadas, sem depender de um escalonador central. Assim, é possível que cada uma delas tome ações independentes, mais rapidamente e com maior precisão.

- A estratégia proposta se mostrou capaz de explorar o compartilhamento dos recursos de processamento, ao mesmo tempo em que promoveu a redução do *makespan* e do tempo médio de *turnaround* do *workload*. Os resultados alcançados favorecem tanto aos usuários das aplicações, proporcionando a redução do tempo de resposta, tanto aos provedores de serviço, ao permitir o aumento da taxa de utilização dos seus recursos e da sua capacidade de atendimento.

A validação e a experimentação da estratégia proposta produziram resultados que justificam considerá-la como uma alternativa para o escalonamento de aplicações em ambientes compartilhados. A estratégia mostrou-se eficaz em reduzir a ociosidade dos recursos, ao mesmo tempo em que o tempo de resposta das aplicações também é reduzido.

Os experimentos nos quais as aplicações compartilharam todos os recursos disponíveis mostraram que foi possível reduzir o *makespan* em até 7,07%. De maneira semelhante, os resultados dos experimentos em que apenas uma fração dos recursos é compartilhada proporcionou reduções de makespan superiores a 15%. Em ambos os casos, a ociosidade dos recursos do ambiente foi praticamente eliminada.

Os resultados sugerem a real possibilidade de escalonar aplicações que compartilham recursos, mesmo quando o objetivo é o alto desempenho, alcançada por meio da redução do *makespan* e do tempo médio de *turnaround* do *workload*. Contudo, é necessário mencionar que alguns experimentos não produziram tais reduções. Esses resultados não restringem, de maneira alguma, a adoção da estratégia proposta, uma vez que sempre houve a melhoria da utilização do ambiente computacional. Verifica-se apenas a necessidade de planejar corretamente a configuração dos valores de **Tempo Alvo** das aplicações que fazem parte do *workload*, a fim de obter os melhores resultados possíveis.

Por fim, as contribuições e os resultados obtidos e apresentados nesta tese levantam questionamentos a respeito de como o escalonamento de aplicações em ambientes de ambientes computacionais compartilhados deve ser realizado em um futuro próximo. É importante deixar claro que o objetivo não é invalidar estratégias de escalonamento tradicionais consolidadas. Porém, a iminência da computação em exaescala, aliada à crescente preocupação com a redução de custos de manutenção de *data centers* e de supercomputadores, rogam por diferentes abordagens de escalonamento que proporcionem novas perspectivas, como a que foi apresentada neste trabalho.

5.1 Trabalhos futuros

Desenvolver e validar uma estratégia de escalonamento é uma tarefa complexa, ainda mais tratando-se do escalonamento de aplicações que compartilham os recursos do ambiente computacional no qual elas executam. Os resultados obtidos, apresentados e comentados nesta tese representam apenas a etapa inicial desse processo. Portanto, é necessário estabelecer novas metas que permitam dar continuidade à pesquisa realizada até o momento. Assim, foram identificadas as seguintes atividades que devem ser executadas para cumprir esse propósito:

- A validação da estratégia de escalonamento também por meio de experimentos envolvendo aplicações reais de e-Ciência. O objetivo é permitir que os usuários dessas aplicações sejam beneficiados pelo uso compartilhado dos recursos, além de propor-

cionar o melhor aproveitamento da infraestrutura computacional.

- A criação de um modelo que permita definir os valores de **Tempo Alvo** das aplicações de forma automática, com o objetivo de tirar esta responsabilidade do usuário. Para isso, planeja-se utilizar os dados obtidos com a validação da estratégia proposta para o escalonamento de aplicações reais de e-Ciência, além daqueles já obtidos durante a elaboração desta tese.

- O aprofundamento do conhecimento a respeito da adoção da estratégia proposta em situações nas quais apenas uma fração dos recursos é compartilhada. Tais situações são muito comuns em supercomputadores e *data centers* de uma forma geral. Experimentos iniciais já realizados mostraram resultados satisfatórios em relação à redução do *makespan* do e do tempo médio de *turnaround* do *workload*. O planejamento envolve a execução de novos experimentos com *workloads* que possuem um maior número de aplicações, além de aumentar também a quantidade de recursos e de variar a forma como eles são compartilhados.

- A avaliação da inclusão dos conceitos de interferência entre aplicações na estratégia de escalonamento proposta. O objetivo é verificar se é possível tirar proveito desses conceitos, quando associados às politicas de comportamento que foram desenvolvidas.

- Iniciar os estudos a respeito dos efeitos da adoção da estratégia de escalonamento proposta sobre o consumo energético dos recursos computacionais compartilhados, com o objetivo de alinhar a continuidade do trabalho com as pesquisas associadas a ambientes de computação em exaescala.

Estes tópicos servirão para direcionar a continuidade da pesquisa no futuro, com o propósito de aumentar a qualidade do escalonamento de aplicações que compartilham recursos e também de melhor a utilização da infraestrutura computacional. Além disso, será explorada a aplicabilidade do trabalho realizado, para contribuir com as pesquisas na área de escalonamento de aplicações de e-Ciência em ambientes computacionais de exaescala.

5.2 Produções associadas e outras contribuições

A pesquisa desenvolvida e apresentada nesta tese contribuiu direta e indiretamente com áreas relacionadas ao escalonamento de aplicações com o compartilhamento de recursos.

As contribuições estão presentes em trabalhos já publicados ou que foram submetidos encontram-se em análise.

O artigo intitulado "*Autonomic Job Scheduling on Distributed and Shared Computing Environments: a New Approach for an Old Problem*" foi submetido para o periódico IEEE *Transactions on Parallel and Distributed Systems* em maio de 2018. Nele, está contida a descrição da estratégia de escalonamento apresentada nesta tese, acompanhada dos experimentos realizados para validá-la. Foram apresentadas no artigo as contribuições associadas à redução da ociosidade do ambiente utilizado, à redução do *makespan* e do tempo médio de *turnaround* do *workload*.

O artigo [83] apresenta um modelo de escalonamento de *clusters* virtuais em nuvens privadas. O trabalho considera que há a possibilidade de compartilhamento de recursos pelos ambientes virtuais, o que é uma característica intrínseca às nuvens computacionais. Para isso, o modelo criado define uma matriz de afinidades, gerada a partir da relação entre as características de uso intensivo dos recursos durante a execução das aplicações. A matriz de afinidade é utilizada para identificar combinações de aplicações que sejam capazes de executar concorrentemente e, ao mesmo tempo, minimizar os impactos negativos causados em virtude dessa concorrência. Os conhecimentos obtidos durante a confecção da tese sobre concorrência de aplicações, compartilhamento de recursos, e sobre afinidade e interferência de aplicações contribuíram para a produção do artigo [83].

O artigo [56] apresenta um algoritmo de escalonamento de máquinas virtuais usadas para executar aplicações de Computação Massivamente Paralela e Distribuída (CMPD) que compartilham os *hosts* do ambiente. O algoritmo aloca as máquinas de acordo com dados históricos das execuções das respectivas aplicações e conforme uma matriz de afinidades. Além disso, o algoritmo apresenta um comportamento proativo, responsável por migrar uma ou mais máquinas virtuais para outros *hosts*, caso sejam previsto que a afinidade entre as aplicações atinja um limite considerado inaceitável. Esse limite pode ser configurado para cada par de aplicações. Os conhecimentos obtidos acerca das métricas de escalonamento, como *makespan*, tempo médio de *turnaround* e utilização dos recursos realizados durante a elaboração desta tese contribuíram para a elaboração do algoritmo de escalonamento apresentada no artigo [56]. Essas métricas foram utilizadas para realizar um novo conjunto de experimentos, o que resultou em novos critérios que são considerados para realizar ou não a migração das máquinas virtuais. Esses novos resultados estão presentes em um artigo que encontra-se em fase final de elaboração, e que será submetido para o periódico *Computers & Electrical Engineering*.

Por fim, os trabalhos apresentados nos artigos [83] e [56] foram integrados e estendidos na forma de um capítulo intitulado "*Affinity Aware Scheduler of Cluster Virtual Nodes on Clouds*" do livro *Advances in Computers and Software Engineering: Reviews*. Nele, são apresentados novos resultados obtidos por meio da adoção conjunta de ambas as abordagens, além de resultados adicionais que ainda não haviam sido publicados.

Referências

[1] Aad, G.; Abajyan, T.; Abbott, B.; Abdallah, J.; Abdel Khalek, S.; Abdelalim, A.; Abdinov, O.; Aben, R.; Abi, B.; Abolins, M., e. a. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett. B716* (2012), 1–29.

[2] Agnetis, A.; Mirchandani, P. B.; Pacciarelli, D.; Pacifici, A. Scheduling Problems with Two Competing Agents. *Operations Research 52*, 2 (2011), 229–242.

[3] Álvarez, G. P. R. *HPC Scheduling in a Brave New World.* Tese de Doutorado, Umea University, Department of Computing Science, Umea, Sweden, 2017.

[4] Anand, E.; Panneerselvam, R. Literature Review of Open Shop Scheduling Problems. *Intelligent Information Management* (jan 2015), 32–53.

[5] Arronategui, U.; Celaya, J. A Highly Scalable Decentralized Scheduler of Tasks with Deadlines. In *2011 12th IEEE/ACM International Conference on Grid Computing(GRID)* (09 2011), vol. 00, pp. 58–65.

[6] Bailey, D. H.; Barszcz, E.; Barton, J. T.; Browning, D. S.; Carter, R. L.; Dagum, L.; Fatoohi, R. A.; Frederickson, P. O.; Lasinski, T. A.; Schreiber, R. S.; Simon, H. D.; Venkatakrishnan, V.; Weeratunga, S. K. The NAS Parallel Benchmarks: Summary and Preliminary Results. In *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 1991), Supercomputing '91, ACM, pp. 158–165.

[7] Baker, M.; Sterling, T. Cluster Computing White Paper. `http://citeseerx.ist.psu.edu/viewdoc/download`, 2000. [Online; acessado em 6 de fevereiro de 2017].

[8] Barney, B. Introduction to Parallel Computing. `https://computing.llnl.gov/tutorials/parallel_comp/#Whatis`, 2017. [Online; acessado em 20 de fevereiro de 2017].

[9] Belviranli, M. E.; Bhuyan, L. N.; Gupta, R. A Dynamic Self-scheduling Scheme for Heterogeneous Multiprocessor Architectures. *ACM Trans. Archit. Code Optim. 9*, 4 (Jan. 2013), 57:1–57:20.

[10] Bender, M. A.; Chakrabarti, S.; Muthukrishnan, S. Flow and Stretch Metrics for Scheduling Continuous Job Streams. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 1998), SODA '98, Society for Industrial and Applied Mathematics, pp. 270–279.

[11] Bode, B.; Halstead, D. M.; Kendall, R.; Lei, Z.; Jackson, D. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters. In *Proceedings of the*

4th Annual Linux Showcase & Conference - Volume 4 (Berkeley, CA, USA, 2000), ALS'00, USENIX Association, pp. 27–27.

[12] Boeres, C.; Alves Fonseca, A.; de Amorim Mendes, H.; Toscano Menezes, L.; Teones Moura, N.; Alves da Silva, J.; de Azevedo Vianna, B.; Rebello, V. An EasyGrid Portal for Scheduling System-aware Applications on Computational Grids. *Concurrency and Computation: Practice and Experience* (2006), 553–566.

[13] Boeres, C.; Nascimento, A. P.; Rebello, V. E. F.; Sena, A. C. Efficient Hierarchical Self-scheduling for MPI Applications Executing in Computational Grids. In *Proceedings of the 3rd International Workshop on Middleware for Grid Computing* (New York, NY, USA, 2005), MGC '05, ACM, pp. 1–6.

[14] Bueno, H. R. Grid SA: Uma Sociedade Autônoma. Dissertação de Mestrado, Instituto de Computação - Universidade Federal Fluminense, Niterói, RJ, Brasil, 2009.

[15] Buyya, R.; Yeo, C. S.; Venugopal, S.; Broberg, J.; Brandic, I. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Gener. Comput. Syst. 25*, 6 (June 2009), 599–616.

[16] Casanova, H.; Desprez, F.; Suter, F. Minimizing Stretch and Makespan of Multiple Parallel Task Graphs via Malleable Allocations. *2013 42nd International Conference on Parallel Processing* (2010), 71–80.

[17] Casanova, H.; Legrand, A.; Zagorodnov, D.; Berman, F. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In *Heterogeneous Computing Workshop* (2000), IEEE Computer Society, pp. 349–363.

[18] Cheng, R.; Gen, M.; Tsujimura, Y. A Tutorial Survey of Job-shop Scheduling Problems Using Genetic Algorithms. *Computers & Industrial Engineering 30*, 4 (1996), 983–997.

[19] Conway, R.; Maxwell, W.; Miller, L. *Theory of Scheduling*. Addison-Wesley Publishing Company, 1967.

[20] Coulouris, G.; Dollimore, J.; Kindberg, T.; Blair, G. *Distributed Systems: Concepts and Design*, 5th ed. Addison-Wesley Publishing Company, USA, 2011.

[21] Curcin, V.; Ghanem, M. Scientific Workflow Systems - Can one size fit all? In *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International* (2008), pp. 1–9.

[22] Darlington, J.; Ghanem, M.; Guo, Y.-k.; Wing To, H. Guided Resource Organisation in Heterogeneous Parallel Computing. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.4309&rep=rep1&type=pdf`, 1996. [Online; acessado em 2 de fevereiro de 2017].

[23] Davidson, S. B.; Freire, J. Provenance and Scientific Workflows: Challenges and Opportunities. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2008), SIGMOD '08, ACM, pp. 1345–1350.

[24] de Oliveira Passos, F. G. *Provendo Autonomia às Aplicações da e-Ciência.* Tese de Doutorado, Universidade Federal Fluminense, Instituto de Computação, Niterói, RJ, 2014.

[25] de Paula Nascimento, A. *Escalonamento Dinâmico para Aplicações Autonômicas MPI em Grades Computacionais.* Tese de Doutorado, Universidade Federal Fluminense, Instituto de Computação, Niterói, RJ, 2008.

[26] Delimitrou, C.; Kozyrakis, C. Paragon: QoS-aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2013), ASPLOS '13, ACM, pp. 77–88.

[27] Department of Energy (DoE), U. S. A. The Magellan Report on Cloud Computing for Science. Tech. rep., U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), dec 2011.

[28] Dongarra, J.; Meuer, H.; Simon, H.; Strohmaier, E. High Performance Computing Today. `http://www.netlib.org/utk/people/JackDongarra/PAPERS/hpc-today.pdf`, 2000. [Online; acessado em 2 de fevereiro de 2017].

[29] Elhart, I.; Langheinrich, M.; Memarovic, N.; Heikkinen, T. Scheduling Interactive and Concurrently Running Applications in Pervasive Display Networks. In *Proceedings of The International Symposium on Pervasive Displays* (New York, NY, USA, 2014), PerDis '14, ACM, pp. 104:104–104:109.

[30] Feitelson, D.; Rudolph, L. Metrics and Benchmarking for Parallel Job Scheduling. In *Job Scheduling Strategies for Parallel Processing, IPPS/SPDP'98 Workshop, Orlando, Florida, USA, March 30, 1998, Proceedings* (1998), pp. 1–24.

[31] Feitelson, D. G.; Rudolph, L. Towards Convergence in Job Schedulers for Parallel Supercomputers. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing* (London, UK, 1996), IPPS '96, Springer-Verlag, pp. 1–26.

[32] Feitelson, D. G.; Rudolph, L.; Schwiegelshohn, U. Parallel Job Scheduling - A Status Report. In *Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing* (Berlin, Heidelberg, 2005), JSSPP'04, Springer-Verlag, pp. 1–16.

[33] Foster, I. What is the Grid? - A Three Point Checklist. *GRIDtoday 1*, 6 (July 2002).

[34] Foster, I.; Kesselman, C.; Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. High Perform. Comput. Appl. 15*, 3 (Aug. 2001), 200–222.

[35] Foster, I.; Zhao, Y.; Raicu, I.; Lu, S. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop, 2008.* (Nov. 2008), IEEE, pp. 1–10.

[36] Garey, M. R.; Johnson, D. S.; Sethi, R. The Complexity of Flowshop and Jobshop Scheduling. *Mathematical Operational Research 1*, 2 (May 1976), 117–129.

[37] Gupta, A. Efficient High Performance Computing in the Cloud: Keynote Talk. In *Proceedings of the 8th International Workshop on Virtualization Technologies in Distributed Computing* (New York, NY, USA, 2015), VTDC '15, ACM, pp. 1–1.

[38] Harrod, W. US DoE Exascale Computing Initiative 2012. `http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/aug12/2012-ECI-ASCAC-v4.pdf`, 2012. [Online; acessado em 8 de junho de 2017].

[39] HPC4E. HPC4E Project information. `https://hpc4e.eu/the-project`, 2018. [Online; acessado em 29 de janeiro de 2018].

[40] Intel, C. 50 Years of Moore's Law: Fueling Innovation We Love and Depend On. `http://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html`, 2015. [Online; acessado em 5 de novembro de 2016].

[41] Jackson, A. HPC and Big Data. `http://icms.org.uk/assets/files/downloads/BigData/JacksonHPCandBigData.pdf`, 2017. [Online; acessado em 27 de novembro de 2017].

[42] Jette, M. A.; Yoo, A. B.; Grondona, M. SLURM: Simple Linux Utility for Resource Management. In *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003* (2002), Springer-Verlag, pp. 44–60.

[43] Kish, L. B. End of Moore's law: Thermal (Noise) Death of Integration in Micro and Nano Electronics. *Physics Letters A 305*, 3–4 (2002), 144–149.

[44] Kumaresan, M.; Venkatesan, G. Enabling High Performance Computing in Cloud Computing Environments. In *2017 IEEE International Conference on Electrical, Instrumentation and Communication Engineering (ICEICE)* (April 2017), pp. 1–6.

[45] Licht, F. L. *Afinidade de Tipos de Aplicações em Nuvens Computacionais*. Tese de Doutorado, Universidade Federal do Paraná, Departamento de Informática, Curitiba, PR, 2014.

[46] Liu, J.; Saletore, V. A. Self-scheduling on Distributed-memory Machines. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 1993), Supercomputing '93, ACM, pp. 814–823.

[47] Liu, Y.; Bobroff, N.; Fong, L.; Seelam, S.; Delgado, J. New Metrics for Scheduling Jobs on Cluster of Virtual Machines. In *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum* (May 2011), pp. 1001–1008.

[48] LNCC. SDumont - Sistema de Computação Petaflópica do SINAPAD. `http://sdumont.lncc.br/`, 2018. [Online; acessado em 26 de fevereiro de 2018].

[49] Luszczek, P.; Meek, E.; Moore, S.; Terpstra, D.; Weaver, V.; Dongarra, J. Evaluation of the HPC Challenge Benchmarks in Virtualized Environments. In *VHPC 2011, 6th Workshop on Virtualization in High-Performance Cloud Computing* (Bordeaux, France, 08/2011 2011).

[50] Mastrolilli, M.; Svensson, O. Hardness of Approximating Flow and Job Shop Scheduling Problems. *J. ACM 58*, 5 (oct 2011), 20:1–20:32.

[51] Mell, P. M.; Grance, T. The NIST Definition of Cloud Computing. Tech. rep., Gaithersburg, MD, United States, 2011.

[52] Mondragon, O. H.; Bridges, P. G.; Jones, T. Quantifying Scheduling Challenges for Exascale System Software. In *Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers* (New York, NY, USA, 2015), ROSS '15, ACM, pp. 8:1–8:8.

[53] NASA. Problem Sizes and Parameters in NAS Parallel Benchmarks. `https://www.nas.nasa.gov/publications/npb_problem_sizes.html#url`, 2018. [Online; acessado em 18 de abril de 2018].

[54] Nascimento, A. P.; Boeres, C.; Rebello, V. E. F. Dynamic Self-scheduling for Parallel Applications with Task Dependencies. In *Proceedings of the 6th International Workshop on Middleware for Grid Computing* (New York, NY, USA, 2008), MGC '08, ACM, pp. 1:1–1:6.

[55] Oliveira, F. J. B. d. Escalonamento Dinâmico, Distribuído e Colaborativo de Aplicações Paralelas e Autônomas com Prazo. Dissertação de Mestrado, Instituto de Computação - Universidade Federal Fluminense, Niterói, RJ, Brasil, 2016.

[56] Oliveira, V.; Barbosa, J.; Bandini, M.; Schulze, B.; Pinto, R. Alocação de Ambientes Virtuais com base na Afinidade entre Perfis de Aplicações Massivamente Paralelas e Distribuídas. In *Anais do XXXV Simpósio Brasileiro de Redes de Computadores, SBRC 2017, Belém-PA, Brasil, Maio, 2017*, pp. 16–29.

[57] Oliveira, V. D. d. Alocação de Ambientes Virtuais Baseada em Perfis Através do Monitoramento e do Aprendizado de Padrões de Consumo em Recursos de CMPD. Dissertação de Mestrado, Instituto Militar de Engenharia, Rio de Janeiro, RJ, Brasil, 2016.

[58] Pinedo, M. L. *Scheduling: Theory, Algorithms and Systems*, 4th ed. Springer, 2010.

[59] Polo, J.; Becerra, Y.; Carrera, D.; Torres, J.; Ayguadé, E.; Steinder, M. Adaptive MapReduce Scheduling in Shared Environments. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, USA, May 26-29, 2014* (2014), pp. 61–70.

[60] Raicu, I.; Zhao, Y.; Dumitrescu, C.; Foster, I.; Wilde, M. FALKON: A Fast and Light-weight tasK executiON Framework. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2007), SC '07, ACM, pp. 43:1–43:12.

[61] Ramakrishnan, L.; Plale, B. A Multi-dimensional Classification Model for Scientific Workflow Characteristics. In *Proceedings of the 1st International Workshop on Workflow Approaches to New Data-centric Science* (New York, NY, USA, 2010), Wands '10, ACM, pp. 4:1–4:12.

[62] Rodriguez, M.; Buyya, R. A Taxonomy and Survey on Scheduling Algorithms for Scientific Workflows in Cloud Computing Environments. *Journal of Parallel and Distributed Computing* (September 2014).

[63] Roemer, T. A. A Note on the Complexity of the Concurrent Open Shop Problem. *J. Scheduling 9*, 4 (2006), 389–396.

[64] Rudolph, D. C.; Polychronopoulos, C. D. An Efficient Message-passing Scheduler Based on Guided Self Scheduling. In *Proceedings of the 3rd International Conference on Supercomputing* (New York, NY, USA, 1989), ICS '89, ACM, pp. 50–61.

[65] Saidi-Mehrabad, M.; Fattahi, P. Flexible Job-shop Scheduling with Tabu Search Algorithms. *The International Journal of Advanced Manufacturing Technology 32*, 5 (Mar 2007), 563–570.

[66] Salehie, M.; Tahvildari, L. Autonomic Computing: Emerging Trends and Open Problems. In *Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software* (New York, NY, USA, 2005), DEAS '05, ACM, pp. 1–7.

[67] Schanne, M.; Gelhausen, T.; Tichy, W. Adding Autonomic Functionality to Object-Oriented Applications. In *Proceedings of the 14th International Conference on Database and Expert Systems Applications (DEXA 2003)* (Sept. 2003), pp. 725–730.

[68] Silberschatz, A.; Galvin, P. B.; Gagne, G. *Operating System Concepts*, 8th ed. Wiley Publishing, 2008.

[69] Silva, F.; Senger, H. Improving scalability of Bag-of-Tasks applications running on master–slave platforms. *Parallel Computing 35*, 2 (2009), 57 – 71.

[70] Silva, L. M.; Buyya, R. Parallel Programming Models and Paradigms. In *High Performance Cluster Computing: Programming and applications*. Prentice Hall PTR, 1999, ch. 1, pp. 4–27.

[71] Smith, C. W.; Tran, S.; Sahni, O.; Behafarid, F.; Shephard, M. S.; Singh, R. Enabling HPC Simulation Workflows for Complex Industrial Flow Problems. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure* (New York, NY, USA, 2015), XSEDE '15, ACM, pp. 41:1–41:7.

[72] Staples, G. TORQUE Resource Manager. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2006), SC '06, ACM.

[73] Tang, P.; Yew, P.-C.; Zhu, C.-Q. Impact of Self-scheduling Order on Performance on Multiprocessor Systems. In *Proceedings of the 2Nd International Conference on Supercomputing* (New York, NY, USA, 1988), ICS '88, ACM, pp. 593–603.

[74] Tannenbaum, T.; Wright, D.; Miller, K.; Livny, M. Beowulf Cluster Computing with Linux. MIT Press, Cambridge, MA, USA, 2002, ch. Condor: A Distributed Job Scheduler, pp. 307–350.

[75] Tantitharanukul, N.; Natwichai, J.; Boonma, P. Workflow-Based Composite Job Scheduling for Decentralized Distributed Systems. In *2013 16th International Conference on Network-Based Information Systems* (Sept 2013), pp. 583–588.

[76] Toporkov, V. V.; Toporkova, A. S.; Tselishchev, A.; Yemelyanov, D.; Potekhin, P. Preference-based Fair Resource Sharing and Scheduling Optimization in Grid VOs. ICCS - International Conference on Computational Science, 2014.

[77] Turek, W. Erlang-based Desynchronized Urban Traffic Simulation for High Performance Computing Systems. *Future Generation Computer Systems 79*, Part 2 (2018), 645 – 652.

[78] Wang, R.; Avasarala, V.; Mullen, T.; Yen, J. A Market-Based Adaptation for Resolving Competing Needs for Scarce Resources. *2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (2006), 350–356.

[79] Wilson, B.; Sara, J.; Amy, C. Cloud vs. Datacenter Costs for High Performance Computing (HPC): A Real World Example. `https://www.internet2.edu/blogs/detail/14114`, 2017. [Online; acessado em 27 de novembro de 2017].

[80] WLCG, T. Worldwide LHC Computing Grid. `http://wlcg.web.cern.ch/`, 2017. [Online; acessado em 27 de novembro de 2017].

[81] Yamada, T.; Nakano, R. Job-shop Scheduling. In *Genetic Algorithms in Engineering Systems*, P. J. Fleming and A. M. S. Zalzala, Eds., vol. 55 of *Control Engineering Series*. The Institution of Electrical Engineers, 1997, Computer Science Department Technical Report 7, pp. 134–160.

[82] Yokoyama, D. Modelo para o Escalonamento de Aplicações Científicas em Ambientes de Nuvens Baseado em Afinidades. Dissertação de Mestrado, Laboratório Nacional de Computação Científica, 2015.

[83] Yokoyama, D.; Schulze, B.; Kloh, H.; Bandini, M.; Rebello, V. Affinity Aware Scheduling Model of Cluster Nodes in Private Clouds. *Journal of Network and Computer Applications 95*, Supplement C (2017), 94 – 104.

[84] Zhuravlev, S.; Blagodurov, S.; Fedorova, A. Addressing Shared Resource Contention in Multicore Processors via Scheduling. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2010), ASPLOS XV, ACM, pp. 129–142.

[85] Zhuravlev, S.; Saez, J. C.; Blagodurov, S.; Fedorova, A.; Prieto, M. Survey of Scheduling Techniques for Addressing Shared Resources in Multicore Processors. *ACM Comput. Surv. 45*, 1 (Dec. 2012), 4:1–4:28.