Universidade Federal Fluminense
Instituto de Computação
Departamento de Ciência da Computação

João Paulo Pereira Tonelli

Simulação de uma versão do sistema de escalonamento dinâmico de tarefas do EasyGrid por meio do *framework* SimGrid

Niterói-RJ
2010

JOÃO PAULO PEREIRA TONELLI

SIMULAÇÃO DE UMA VERSÃO DO SISTEMA DE ESCALONAMENTO DINÂMICO DE TAREFAS DO EASYGRID POR MEIO DO FRAMEWORK SIMGRID

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação. Área de concentração: Sistemas Operacionais.

Orientador: Profª. Dra. MARIA CRISTINA SILVA BOERES

Niterói-RJ

2010

JOÃO PAULO PEREIRA TONELLI

SIMULAÇÃO DE UMA VERSÃO DO SISTEMA DE ESCALONAMENTO DINÂMICO DE TAREFAS DO EASYGRID POR MEIO DO FRAMEWORK SIMGRID

Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal Fluminense como parte dos requisitos para obtenção do Grau de Bacharel em Ciência da Computação. Área de concentração: Sistemas Operacionais.

Aprovada em julho de 2010.

BANCA EXAMINADORA

Profª. Dra. MARIA CRISTINA SILVA BOERES - Orientador
UFF

Profª. Dra. SIMONE DE LIMA MARTINS
UFF

Profª. Dra. ALINE DE PAULA NASCIMENTO
UFF

NITERÓI
2010

Em memória e honra do meu pai.

Agradecimentos

A Deus por me conceder saúde física e mental para a realização desta monografia, além de me educar e de me guiar em cada etapa da minha vida.

Aos meus familiares, em especial a minha mãe, por todo o apoio tanto emocional quanto financeiro a mim prestado.

Aos meus amigos que sempre me estimularam positivamente a prosseguir.

A minha orientadora Professora Cristina Boeres, por todo apoio e confiança em mim depositados.

À equipe do SimGrid por responder prontamente a todos os meus emails.

A todas as pessoas não mencionadas que contribuiram direta ou indiretamente para a realização desta monografia.

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

Com o avanço da tecnologia de redes e o barateamento de seus dispositivos, os ambientes distribuídos estão se tornando mais viáveis e atingindo dimensões cada vez maiores, chegando a romper limites continentais. Neste contexto, surgem as grades computacionais [9], que estão se popularizando por oferecerem grande poder computacional a um baixo custo. No entanto, tais ambientes geralmente são heterogêneos, dinâmicos, compartilhados e podem possuir diferentes políticas de acesso aos seus recursos, todas estas características dificultam o gerenciamento do mesmo. Neste contexto, foram propostas na literatura diversos sistemas gerenciadores de grades computacionais, cujos sistemas de escalonamento desempenham um papel crucial na utilização adequada dos seus recursos, tentando minimizar o tempo de execução de suas aplicações e maximizar a sua vazão. Dentre eles, o EasyGrid [1] se destaca por ser embutido na própria aplicação, fornecendo a ela a capacidade de auto-monitoramento, auto-ajuste e auto-recuperação. Além disso, o seu sistema de escalonamento é altamente escalável, flexível e eficiente. O objetivo principal deste projeto é implementar um simulador para uma versão do sistema de escalonamento do EasyGrid, utilizando para este fim o *framework* de simulação SimGrid [5]. Tal simulador é desenvolvido com o intuito de ser utilizado como uma ferramenta complementar para a análise e avaliação da estrutura de escalonamento simulada, sendo capaz de simular uma grande variedade de ambientes distribuídos. Há algumas vantagens em se trabalhar com ambientes simulados, entre elas destaca-se a possibilidade de se obter total controle sobre tais ambientes, o que facilita a elaboração de cenários de testes confiáveis e reprodutíveis, características estas imprescindíveis para o objetivo pretendido. Foram realizados alguns experimentos preliminares do sistema de escalonamento simulado, visando analisar a sua escalabilidade, o seu custo de execução e os benefícios obtidos com a sua utilização.

Palavras-chave: Escalonamento de tarefas. Grade computacional. Simulador.

Abstract

As computer network technologies evolves and its devices become cheaper, the distributed environments tend to be even more feasible and to reach larger dimensions that may overcome continentals boundaries. In this scenario, the computational grids [9] emerge with much popularity, providing a great computational power at a low cost. However, such environments are usually heterogeneous, dynamic, shared and may have different access policies among its resources. All these features make the management of the grid more complex. So, loads of computational grid management systems were proposed in the literature, which scheduling systems play an important role in the adequate using of its resources, trying to minimize applications makespan and to maximize the system throughput. Among them, the EasyGrid [1] turns out to be more attractive, whereas it is embedded into the user application providing it the ability of self-monitoring, self-adjusting and self-recovering. Furthermore, its scheduling system is highly scalable, flexible and efficient. The main goal of this project is to develop a simulator for the execution of a reduced version of EasyGrid's scheduling system, using for this purpose the SimGrid simulation framework [5]. Such simulator is designed to be used as a complementary tool for analyzing and evaluating the simulated scheduling structure, being able of simulate a large variety of distributed environments. There are some advantages of working with a simulated environment, through its use it is possible to have a complete control over the whole environment, which makes the development of test scenarios reliable and reproducible, these characteristics are essential to the intended goal. There were made some preliminary experiments of the simulated scheduling system in order to analyze its scalability, its execution cost and the benefits obtained from its use.

Keywords: Task Scheduling. Computational Grid. Simulator.

Capítulo 1

Introdução

Com o avanço da tecnologia de redes de computadores e o barateamento de seus dispositivos, os ambientes distribuídos estão se tornando cada vez mais viáveis e atingindo dimensões cada vez maiores, chegando a romper limites continentais. Neste contexto, surgem as grades computacionais [9], que estão se popularizando cada vez mais por oferecerem grande poder computacional a um baixo custo. Entretanto, tais ambientes geralmente são heterogêneos, dinâmicos, compartilhados e podem possuir diferentes políticas de acesso aos seus recursos, todas estas características dificultam o gerenciamento dos mesmos. Esta dificuldade aumenta significativamente com o crescimento acentuado da quantidade de recursos a serem gerenciados.

Foram propostas na literatura diversos sistemas gerenciadores de grades computacionais, como os em [11, 10, 12], que incluem a implementação de políticas de escalonamento de tarefas, que desempenham um papel crucial na utilização adequada dos recursos da grade, tentando minimizar o tempo de execução de suas aplicações e maximizar a sua vazão.

O EasyGrid [1] é um *middleware* para gerenciamento de aplicações MPI, o qual é embutido na própria aplicação. Tais aplicações podem ser do tipo *bag-of-tasks* ou possuírem relações de dependência entre suas tarefas. Entre as principais funcionalidades oferecidas pelo EasyGrid às aplicações MPI temos: o auto-monitoramento, onde a aplicação passa a monitorar a sua própria execução; o auto-ajuste, onde a aplicação se torna capaz de se ajustar às mudanças do ambiente de execução por meio do reescalonamento dinâmico de suas tarefas; e a auto-recuperação, a qual permite que a aplicação se recupere em caso de falhas.

O sistema de escalonamento do EasyGrid [2] é hierarquicamente descentralizado por aplicação, tornando-o mais escalável e possibilitando a definição de políticas distintas tanto entre diferentes aplicações, quanto entre diferentes níveis hierárquicos. Esta característica, entre outras, viabiliza a sua utilização em ambientes distribuídos de larga escala e heterogêneos como as grades computacionais.

Em um aplicação, os escalonadores de diferentes níveis hierárquicos cooperam entre si a fim de minimizar o tempo total de execução da própria aplicação MPI que está sendo monitorada. Além disso, em uma grade podem haver mais de uma aplicação EasyGrid executando, visto que os escalonadores distribuídos de cada aplicação não conflitam entre si.

Uma abordagem óbvia para se obter resultados experimentais válidos é conduzir os experimentos em plataformas de produção, ou pelo menos em um grande ambiente de testes. No entanto, tal abordagem geralmente é inviável, pois plataformas de produção podem não estar disponíveis para a realização dos experimentos pretendidos, de modo a não interromper o seu fluxo de produção. Além do mais, experimentos realizados em ambientes de execução reais podem consumir muito tempo e os resultados obtidos podem não ser reprodutíveis, e geralmente não o são, devido a natureza dinâmica dos seus recursos [5].

Como alternativa para contornar os problemas citados, podemos, ao invés de obter acesso a sistemas reais de computação, simular estes sistemas, possibilitando a criação de inúmeros cenários de experimentação, sem restrições quanto a quantidade e a disponibilidade de seus recursos, tornando, desta forma, os experimentos realizados muito mais abrangentes. Outro benefício desta abordagem é o controle total e irrestrito do ambiente simulado, o que torna os resultados obtidos completamente confiáveis e reprodutíveis.

A ferramenta utilizada para simulação neste projeto é o SimGrid [5]. Ele é um *framework* bastante conhecido pela comunidade da área afim, que fornece funcionalidades básicas para a simulação de aplicações distribuídas, tendo como principal objetivo viabilizar o estudo de políticas de escalonamento em diversos tipos de ambientes de execução.

O objetivo principal deste projeto é implementar um simulador para a execução de uma versão reduzida do sistema de escalonamento dinâmico do EasyGrid, que consiste no monitoramento de um único site (organização virtual[9]) e seus respectivos recursos. Além disso, somente aplicações do tipo *bag-of-tasks* são contempladas. Tal simulador é

desenvolvido com o intuito de ser utilizado como uma ferramenta complementar para a análise e avaliação da estrutura de escalonamento simulada, sendo capaz de simular uma grande variedade de ambientes distribuídos.

Esta monografia é organizada da seguinte forma: no capítulo 2 é descrita toda a estrutura de escalonamento dinâmico do EasyGrid, conforme proprosto em [2]; no capítulo 3 é apresentado o *framework* SimGrid, o qual é utilizado para a simulação realizada; no capítulo 4 é especificado o sistema de escalonamento simulado, assim como realizadas algumas considerações concernentes ao simulador implementado; alguns experimentos preliminares são realizados no capítulo 5; e as conclusões e os trabalhos futuros são relacionados no capítulo 6.

Capítulo 2

Estrutura de escalonamento do EasyGrid

Este capítulo descreve resumidamente a estrutura hierárquica e distribuída de escalonamento do EasyGrid, conforme proposta em [2]. Seu objetivo é proporcionar uma execução eficiente de aplicações paralelas e distribuídas em ambientes dinâmicos e heterogêneos como as grades computacionais, permitindo o emprego de diferentes heurísticas de escalonamento de forma a satisfazer tanto os requisitos do usuário, quanto os requisitos dos sites(organizações virtuais[9]) da grade computacional.

2.1 Hierarquia de escalonamento

Os escalonadores do EasyGrid estão associados aos processos gerenciadores da aplicação, que são distribuídos em três níveis diferentes de hierarquia. A figura 2.1 mostra um exemplo desta hierarquia numa grade computacional com três sites (organizações virtuais[9]). Nela podemos observar os três tipos de escalonadores dinâmicos existentes no EasyGrid, a saber:

- GDS (*Global Dynamic Scheduler*): situado no topo da hierarquia e associado ao processo gerenciador global (GM - *Global Manager*), o escalonador dinâmico global é responsável pelo reescalonamento de tarefas entre os diferentes sites.
- SDS(*Site Dynamic Scheduler*): subordinado ao GDS e associado ao processo gerenciador do site (SM - *Site Manager*), o escalonador dinâmico do site é responsável pelo reescalonamento de tarefas somente entre os recursos de seu site.

- HDS(*Host Dynamic Scheduler*): localizado no nível mais baixo da hierarquia e associado ao processo gerenciador da máquina (HM - *Host Manager*), o escalonador dinâmico da máquina é responsável pelas políticas de autorização da aplicação na máquina alvo e por disparar os processos do usuário.

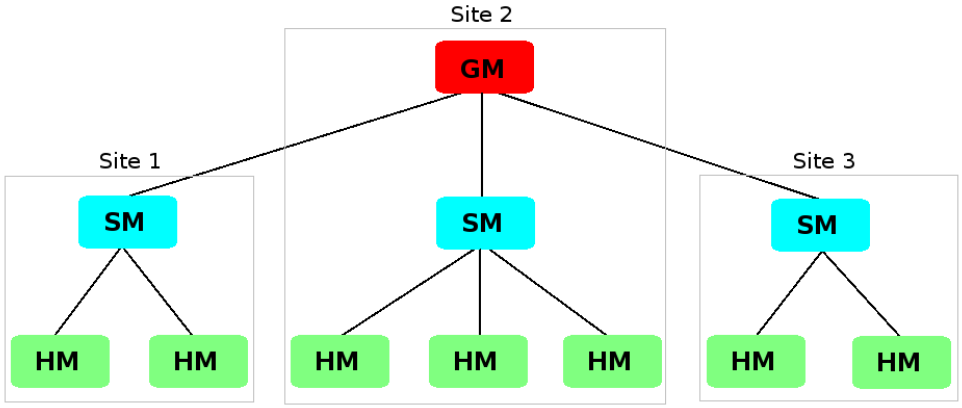

Figura 2.1: Exemplo da hierarquia de escalonadores do EasyGrid

O EasyGrid trata duas classes distintas de aplicações, a saber: aplicações do tipo *bag-of-tasks* (BoT) que são compostas somente por tarefas independentes entre si; e aplicações que são representadas por um grafo acíclico direcionado (GAD ou DAG em inglês) que define as tarefas ou processos da aplicação e suas relações de precedência.

Cada aplicação possui sua própria hierarquia de escalonadores dinâmicos, cujas funções variam de acordo com o nível em que se encontram nesta hierarquia. Como cada site, ou organização virtual[9], pode definir suas próprias políticas de acesso aos seus recursos, políticas de escalonamento de tarefas distintas podem ser empregadas entre escalonadores de mesmo nível. Por exemplo, se na figura 2.1 o site 1 permitisse que as tarefas da aplicação somente executem quando os recursos estiverem ociosos, enquanto que o site 3 permitesse a execução de até quatro tarefas concorrentemente, então as políticas de escalonamento empregadas pelos HDS do site 1 e do site 3 seriam diferentes.

Uma característica importante da estrutura hierárquica e distribuída dos escalonadores do EasyGrid é a flexibilidade obtida a partir do emprego de diferentes políticas de escalonamento entre aplicações, ou, até mesmo, entre níveis hierárquicos distintos, o que possibilita uma melhor adequação aos diferentes requisitos da aplicação ou do sistema. Outra característica importante da estrutura é a sua escalabilidade, visto que nela não

há um escalonador central para todo o sistema, mas sim vários escalonadores distribuídos por aplicação. Estas duas características principais viabilizam o seu uso em ambientes distribuídos de larga escala como as grades computacionais.

As funções de cada escalonador serão descritas a seguir, independentemente das heurísticas de escalonamento adotadas.

2.1.1 Escalonador Dinâmico da Máquina - HDS

O escalonador dinâmico de máquina é responsável por definir a ordem em que as tarefas da aplicação serão executadas, assim como determinar o instante em que uma tarefa é criada na sua máquina.

A ordem de execução das tarefas da aplicação pode ser definida por um escalonamento estático, ou seja, um escalonamento prévio, realizado antes de se submeter a aplicação à execução. Alternativamente, para aplicações cujas tarefas possuam relações de precedência, aplicações *GAD*, pode ser adotado o modelo *dataflow*, onde as tarefas são executadas à medida em que ficam prontas, isto é, à medida em que todas suas dependências forem satisfeitas e os respectivos dados estiverem disponíveis.

Existem também algumas políticas distintas para se determinar o instante de criação de uma tarefa. O EasyGrid, por exemplo, permite que sejam criadas várias tarefas no mesmo instante de tempo, possibilitando assim a execução concorrente das mesmas, o que traz benefícios dependendo de quantas tarefas executam concorrentemente. Este número é informado pelo usuário da aplicação, e o seu valor ótimo depende das características de E/S e da duração de cada tarefa [4].

Outra política interessante, implementada pelo EasyGrid, é disparar tarefas da aplicação apenas em recursos que estejam ociosos. A utilização desta política, assim como a definição de ociosidade dos recursos, são determinadas pelas suas regras de acesso na grade computacional. Por exemplo, um site pode disponibilizar apenas um recurso quando este estiver ocioso, o que pode significar, neste site, oferecer 80% ou mais de seu poder computacional.

Independentemente das políticas utilizadas pelo HDS, sempre que um processo u do usuário (tarefa da aplicação) termina, a camada de monitoramento do EasyGrid, mais especificamente o HM, fornece as seguintes informações para o HDS:

- O tempo de CPU do processo u (*tempo_cpu(u)*): é o tempo em que u efetivamente

utilizou o processador no qual foi alocado

- O tempo de parede do processo u (*tempo_parede(u)*): é o tempo medido desde o momento em que u foi disparado até o seu término, o que inclui tanto o seu tempo de processamento, quanto o seu tempo de espera.

Com essas duas informações o HDS estima o poder computacional do recurso e o tempo restante de execução, considerando as tarefas nele alocadas.

Logo, seja $R_k = \{r_1, r_2, ..., r_x\}$ o conjunto de x recursos disponíveis em um site s_k, e seja u o último processo que finalizou a execução em $r_j \in R_k$, e nt_j o número máximo de tarefas concorrentes em r_j. O fator de utilização da CPU de r_j (*percentage of CPU utilization - pcu*$_j$) é definido como:

$$pcu_j = \frac{tempo\_cpu(u)}{tempo\_parede(u)} * nt_j \tag{2.1}$$

Então, o poder computacional (*computational power - cp*$_j$) relativo oferecido pelo recurso r_j é calculado pelo HDS$_j$ da seguinte forma:

$$cp_j = \frac{pcu_j}{csi_j} \tag{2.2}$$

O csi_j (*computational slowdown index*), índice de retardo computacional do recurso j, é um valor normalizado para estimar a velocidade de processamento do recurso j, ele é obtido por meio de um programa modelador [3] executado previamente. Note que, o recurso mais rápido da grade possui o menor *csi*.

Depois de calcular o poder computacional relativo do recurso, o HDS$_j$ calcula então o tempo restante de execução (*estimated remaining time - ert*) das tarefas prontas que ainda não foram disparadas no recurso r_j, onde $Vj \subset V$ representa o conjunto destas tarefas e $\epsilon(v_i)$ representa o peso computacional da tarefa v_i, ou seja, a quantidade de operações realizadas pela tarefa v_i.

$$ert_j = (\sum_{v_i \in V_j} \epsilon(v_i)) * \frac{1}{cp_j} \tag{2.3}$$

Por fim, o HDS$_j$ disponibiliza o ert_j e o cp_j calculados ao SDS correspondente, para que possam ser usados de acordo com a política de escalonamento definida para o site. Note que estes valores são adicionados à mensagem de monitoramento que o HM envia ao SM para informar o término da tarefa de aplicação u, logo, não são criadas mensagens adicionais para tanto.

O ert_j e o cp_j calculados também podem ser enviados pelo HM ao SM por meio de mensagens de *heartbeat*, que são disparadas sempre que o intervalo de tempo entre o envio da última mensagem de monitoramento e o tempo atual ultrapassar um limite determinado, chamado de $I_{heartbeat}$ que pode ser definido pelo usuário. Situação esta que pode ser ocasionada quando não existem processos executando no recurso r_j, ou quando o processo que está executando em r_j é muito demorado. Nestes casos, o cálculo do percentual de utilização da CPU (pcu_j) se torna mais complicado, devido a inexistência de um último processo u recente do qual se possa obter o tempo de CPU e o tempo de parede. Portanto, o HDS_j se baseia em funções do sistema operacional para realizar tal cálculo.

O Algoritmo 1 resume as ações executadas por um HDS, enquanto a execução da aplicação não for finalizada.

Algoritmo 1 : HDS_j

```
1   dispara tarefas segundo a política local;
2   Se (recebe mensagem fim(u) do HM) então
3       calcula pcu_j baseado em cpu_time(u) e wall_clock_time(u);
4       calcula cp_j e ert_j;
5       adiciona cp_j e ert_j à mensagem recebida e repassa para o SM;
    fim Se
6   Se (I_heartbeat expirou) então
7       calcula pcu_j;
8       calcula cp_j e ert_j;
9       adiciona cp_j e ert_j à mensagem de heartbeat e envia ao SM;
    fim Se
10  Se (recebe pedido tarefas_ask de SDS) então
11      selecionar_tarefas_HDS(tarefas_ask);
    fim Se
```

O HDS também participa do escalonamento dinâmico do site, sendo ativado sempre que o SDS lhe enviar um pedido para ceder tarefas, caso onde ele deve selecionar quais tarefas serão cedidas, se possível(procedimento *selecionar_tarefas_HDS()*). Além disso, o SDS pode enviar para o HDS novas tarefas a serem executadas.

2.1.2 Escalonador Dinâmico do Site - SDS

O escalonador dinâmico do site, SDS, é responsável por escalonar tarefas apenas entre as máquinas de seu site, com o objetivo de minimizar o tempo total de execução da aplicação. Considerando $S = s_1, s_2, ..., s_q$ o conjunto de q sites disponíveis na grade

computacional, cada site s_k pode possuir diferentes políticas de escalonamento e executá-las em momentos distintos. Como definido anteriormente, o conjunto de recursos pertencentes ao site s_k é denotado por R_k.

Para aplicações GAD, ou seja, aplicações paralelas compostas de tarefas com relações de precedência entre si, há dois conjuntos distintos de tarefas a serem considerados pelo SDS, que são: o conjunto de tarefas prontas composto por tarefas que estão prontas para execução; e o conjunto de tarefas pendentes composto por tarefas cujas relações de dependência ainda não foram satisfeitas.

O SDS trata separadamente os conjuntos de tarefas prontas e pendentes, disparando eventos de escalonamento específicos sempre que for detectada a necessidade de reescalonamento de cada conjunto. Portanto, é possível que os eventos de escalonamento de tarefa prontas e pendentes ocorram com diferentes frequências.

Sempre que o SM do site s_k recebe os valores de ert_i e cp_i do HM de um recurso $r_i \in R_k$, ele os disponibiliza para o SDS_k que, por sua vez, calcula a média do tempo restante de execução de tarefas prontas($\overline{ert_k}$) e a soma do poder computacional relativo(tcp_k) do site s_k, da seguinte forma:

$$\overline{ert_k} = \frac{\sum r_j \in R_k ert_j}{|R_k|} \tag{2.4}$$

$$tcp_k = \sum r_j \in R_k cp_j \tag{2.5}$$

O $\overline{ert_k}$ e o tcp_k calculados são adicionados nas mensagens de monitoramento entre o SM e o GM para serem utilizados no escalonamento dinâmico global realizado pelo GDS.

Periodicamente, o SDS valida o escalonamento atual de suas tarefas a fim de verificar a necessidade de realização de um reescalonamento dinâmico das mesmas. Para o conjunto de tarefas prontas o intervalo mínimo de validação do escalonamento é denotado por I_{sched}, enquanto que para o conjunto de tarefas pendentes a frequência de validação é determinada pela topologia do grafo acíclico direcionado, GAD, que representa a aplicação, onde a informação utilizada é relacionada ao nível topológico l das tarefas em execução em s_k.

O Algoritmo 2, apresentado a seguir, resume as principais ações do escalonador dinâmico do site s_k.

Algoritmo 2 : SDS_k

```
1   Se (recebe mensagem de monitoramento ou heartbeat de algum HDS) então
2       atualiza tcp_k e ert_k;
3       adiciona o tcp_k e ert_k à mensagem recebida e repassa para o GM;
    fim Se
4   Se (schedAtivo = 0) então
5       Se (I_sched ultrapassado) então
6           (aloc, R_ask, R_sub, tarefas_ask) ← escalonar_prontas_local();
7           Se(tarefas_ask ≠ ∅)então
8               schedAtivo ← 1;
9               contPedido ← |R_ask|; contResposta ← 0;
10              envia tarefas_ask_j, ∀r_j ∈ R_ask;
            fim Se
        fim Se
11      Se (Nível l ultrapassado) então
12          (aloc, R_ask, R_sub, tarefas_ask) ← escalonar_pendentes_local();
13          Se(tarefas_ask ≠ ∅)então
14              schedAtivo ← 1;
15              contPedido ← |R_ask|; contResposta ← 0;
16              envia tarefas_ask_j, ∀r_j ∈ R_ask;
            fim Se
        fim Se
    fim Se
17  Se (schedAtivo = 1) então
18      verificar_recebimento_tarefas_SDS(aloc, R_sub);
    fim Se
```

Um evento de escalonamento local só é ativado($schedAtivo = 1$, Algoritmo 2, linhas 8 e 14) quando o SDS_k verifica a necessidade de reescalonamento de tarefas no site s_k, ou seja, quando ele verifica que o tempo total de execução de s_k pode ser reduzido (linhas 6 e 12). Neste caso, o SDS_k define a melhor alocação de tarefas nos seus recursos que é denotado por *aloc*. Além disso, ele também identifica o conjunto de recursos que devem ceder tarefas, representado por $R_{ask} \subset R_k$, o conjunto de recursos que devem receber tarefas, $R_{sub} \subset R_k$, e o conjunto de tarefas que serão pedidas no reescalonamento, denotado por $tarefas_{ask}$.

O evento de escalonamento local consiste basicamente em enviar um pedido de tarefas, $tarefas_{ask}$, para cada $r_j \in R_{ask}$ (Algoritmo 2, linhas 10 e 16). Este pedido pode identificar tarefas específicas a serem cedidas por r_j, ou apenas determinar uma quantidade destas tarefas. Normalmente, só se torna necessário especificar as tarefas a serem cedidas quando há relações de precedência entre as tarefas da aplicação, caso este em que a correta escolha das mesmas podem significar uma redução no tempo total de execução da aplicação.

Cada escalonador dinâmico da máquina alocado em $r_j \in R_{ask}$, ao receber o pedido

de tarefas, $tarefas_{ask}$, do SDS_k, irá analisar a possibilidade de atendê-lo, selecionando as tarefas para o reescalonamento (Algoritmo 1, linha 11). Em seguida, cada HDS_j envia o pacote de tarefas cedido, que pode ser vazio, para o SDS_k, o qual verifica o recebimento de todos os pacotes de tarefas (Algoritmo 2, linha 18), e então o evento de escalonamento termina ($SchedAtivo = 0$).

Note que o SDS participa do escalonamento dinâmico global em duas ocasiões: quando recebe tarefas do GDS para serem redistribuídas no seu site, ou quando é selecionado para ceder tarefas ao GDS para que estas possam ser reescalonadas em outros sites.

2.1.3 Escalonador Dinâmico Global - GDS

O escalonador dinâmico global, GDS, é capaz de reescalonar tarefas entre os diferentes sites da grade computacional. O GDS possui comportamento distinto em relação ao tipo de aplicação tratada: BoT e GAD. Para aplicações BoT o GDS pode apenas ativar eventos de escalonamento de tarefas, enquanto que para aplicações GAD ele pode atuar como um mediador entre os pedidos de tarefas realizados pelos sites da grade.

No caso de aplicações BoT, apesar do GDS desconhecer o poder computacional de cada recurso da grade, ele recebe os valores de $\overline{ert}$ e tcp enviados pelos sites por meio de mensagens de monitoramento. Como neste caso todas as tarefas da aplicação estão prontas para executar, o GDS usa as informações obtidas de cada site para decidir disparar eventos de escalonamento global que visam manter o sistema inteiro balanceado, supondo que cada site s_k já está internamente balanceado.

Para aplicações GAD, devido as relações de precedência existentes entre as tarefas, o uso dos valores de $\overline{ert}$ e tcp não se aplicam para o cálculo de um escalonamento eficiente. Como apenas o SDS_k de cada site s_k conhece o poder computacional dos seus recursos e a alocação atualizada das suas tarefas, somente eles podem realizar com maior precisão o escalonamento de suas tarefas. Sendo assim, toda vez que o SDS verificar a necessidade de um evento de escalonamento local, pedidos de tarefas pertencentes a outros sites podem ser feitos ao GDS, neste caso o GDS atua como um mediador.

Ainda para o caso de aplicações GAD, o GDS mantém informações atualizadas periodicamente pelos sites, como o tempo de execução parcial de cada site s_k e o valor estimado do tempo de execução final, ft, das tarefas que ainda não executaram. Desta

forma, sempre que um SDS precisar calcular o tempo de execução total do seu site, ele pode pedir o GDS o tempo de fim de uma tarefa u que não pertença ao seu site, mas que seja predecessora imediata de suas tarefas, sem que este pedido tenha que ser transmitido até o site em que a tarefa u está alocada.

2.2 Resumo

Este capítulo apresentou, de uma forma geral, a estrutura hierárquica de escalonamento dinâmico do *middleware* EasyGrid [2]. Políticas distintas podem ser utilizadas nos seus diferentes níveis de hierarquia, possibilitando desta forma uma melhor adaptação à execução em grades computacionais. Além disso, como cada aplicação possui a sua própria hierarquia de escalonadores, o esforço do escalonamento é dividido entre as diversas aplicações que compartilham os recursos da grade, tornando o sistema mais escalável.

Capítulo 3

O framework SimGrid

Este capítulo descreve sucintamente a estrutura do *framework* SimGrid, apresentando seus principais módulos e seus objetivos. O módulo MSG é escolhido para ser utilizado na simulação realizada neste trabalho, e uma descrição mais detalhada dele é apresentada.

3.1 Visão geral do SimGrid

Criado em 1999, e em desenvolvimento desde então, o SimGrid[5] é um *framework* bem conhecido na comunidade da área afim, que fornece funcionalidades básicas para a simulação de aplicações distribuídas, tendo como principal objetivo possibilitar o estudo de políticas de escalonamento em diversos tipos de ambientes, que podem variar desde uma pequena rede de computadores até uma grade computacional.

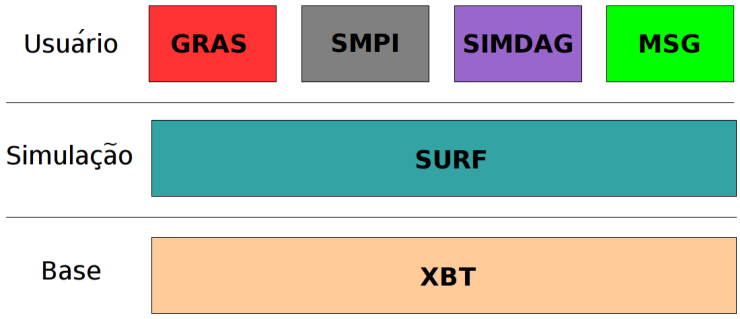

Figura 3.1: As camadas dos SimGrid e seus módulos

Os módulos do SimGrid se situam em três camadas distintas que são a camada

de base, a camada de simulação e a camada de usuário, conforme vistos na figura 3.1, e descritas a seguir.

A camada de base é composta somente pelo módulo XBT, o qual foi desenvolvido de forma a promover a portabilidade e fornecer funcionalidades básicas aos programadores como suporte a depuração da aplicação, tratamento de exceções, estruturas de dados dinâmicas, entre outras facilidades. Note que apesar da arquitetura em camadas, as funcionalidades do XBT são disponibilizadas a todos os outros módulos, podendo ser utilizadas inclusive nos módulos da camada de usuário.

A camada de simulação também é composta por um único módulo, o SURF, que é responsável pela simulação do ambiente de execução, o que inclui a utilização de diversos modelos matemáticos tanto para a simulação dos recursos computacionais, quanto para a simulação dos recursos de comunicação. Por exemplo, o SURF utiliza modelos matemáticos para simular o não determinismo do protocolo TCP [6], o compartilhamento dos enlaces de comunicação [7], entre outros.

Na camada de usuário situam-se os módulos principais, os quais são diretamente utilizados pelos usuários finais do SimGrid. Cada um dos quatro módulos da camada de usuário, a saber: GRAS, SMPI, SimDAG e MSG, fornece funcionalidades distintas. Portanto é necessário conhecer cada módulo, a fim de se escolher o mais adequado para o objetivo desejado.

O GRAS é um módulo complexo e poderoso que permite que a aplicação nele desenvolvida seja executada tanto na plataforma simulada do SURF, quanto em uma plataforma real sem a alteração do código-fonte da aplicação. No entanto, como o objetivo deste projeto é analisar a execução dos escalonadores do EasyGrid[2] apenas em ambientes simulados, a complexidade introduzida pelo GRAS não agrega valor ao objetivo pretendido. Deste modo, optou-se por não utilizá-lo neste projeto.

O módulo SMPI permite que aplicações MPI executem na plataforma simulada do SURF sem alterações no código-fonte, ou seja, permite a emulação de uma aplicação MPI. Como os escalonadores do EasyGrid[2] são implementados em MPI, este módulo seria o ideal a ser utilizado. Todavia, o SMPI ainda se encontra em fase de desenvolvimento, não estando disponível na versão 3.3.4 do SimGrid, que é a última versão disponível no momento da realização deste projeto.

O módulo SimDAG é utilizado para a simulação de aplicações que podem ser

representadas como um grafo acíclico direcionado (DAG), onde cada nó corresponde a uma tarefa da aplicação, e cada aresta corresponde a uma relação de dependência entre as tarefas por ela conectadas. Este módulo é geralmente utilizado para estudos de políticas de escalonamento estático de aplicações DAG, visto que ele não possibilita o monitoramento dinâmico, em tempo de execução, das tarefas da aplicação. Além disso, o SimDAG também não possibilita o desenvolvimento de políticas de escalonamento distribuídas. Logo, a utilização do SimDAG para o objetivo proposto neste projeto não se justifica.

O módulo MSG possui o mesmo objetivo do módulo GRAS, que é o de fornecer primitivas para o desenvolvimento de aplicações distribuídas. No entanto, a aplicação desenvolvida no MSG executa apenas sobre a plataforma simulada do SURF, tornando este módulo bem mais simples em relação ao GRAS. Dentre todos os módulos disponíveis da camada de usuário, o MSG se mostrou o mais adequado para a simulação dos escalonadores do EasyGrid[2].

3.2 Modelo arquitetural

Independentemente do módulo a ser utilizado, o ambiente de execução a ser simulado é descrito por um arquivo XML, o qual especifica as característas de sua arquitetura, descrevendo tanto os seus recursos computacionais, quanto os seus recursos de comunicação. A figura 3.2 apresenta um pequeno exemplo deste arquivo com os principais atributos a serem especificados.

Primeiramente são especificados os recursos computacionais que compõem o ambiente, neste exemplo especificamos três *hosts*: o *host*0 e o *host*2 com capacidade computacional de 500000 $flops$ cada um, e o *host*1 com capacidade computacional de 1000000 $flops$ (linhas 5, 6 e 7).

Em seguida são especificados os enlaces do ambiente de execução, neste exemplo temos: o $linkA$ com 100000 bits de largura de banda e 0.001 segundos de latência, o $linkB$ com 50000 bits de largura de banda e 0.005 segundos de latência, e o $linkC$ com 400000 bits de largura de banda e 0.0001 segundos de latência (linhas 9, 10 e 11).

Por último são definidas as rotas existentes entre os *hosts* do ambiente. Observe que as rotas podem ser compostas por mais de um enlace (linha 14), e que elas podem ser assimétricas, por exemplo, a rota do *host*1 para o *host*2 é diferente da rota do *host*2

```
1  <?xml version='1.0'?>
2  <!DOCTYPE platform SYSTEM "simgrid.dtd">
3  <platform version="2">
4
5  <host id="host0" power="500000"/>
6  <host id="host1" power="1000000"/>
7  <host id="host2" power="500000"/>
8
9  <link id="linkA" bandwidth="100000" latency="0.001"/>
10 <link id="linkB" bandwidth="50000" latency="0.005"/>
11 <link id="linkC" bandwidth="400000" latency="0.0001"/>
12
13 <route src="host0" dst="host1"> <link:ctn id="linkA"/> </route>
14 <route src="host0" dst="host2"> <link:ctn id="linkA"/> <link:ctn id="linkB"/> </route>
15 <route src="host1" dst="host0"> <link:ctn id="linkA"/> </route>
16 <route src="host1" dst="host2"> <link:ctn id="linkB"/> </route>
17 <route src="host2" dst="host0"> <link:ctn id="linkB"/> <link:ctn id="linkA"/> </route>
18 <route src="host2" dst="host1"> <link:ctn id="linkC"/> </route>
19
20 </platform>
```

Figura 3.2: Exemplo de arquivo XML para o ambiente de execução simulado

para o *host*1 (linha 16 e 18).

Além dos atributos mostrados neste exemplo, vários outros podem ser especificados. A definição de tipo de documento, DTD, completa do arquivo XML usado pelo SimGrid para descrever o ambiente de execução simulado pode ser encontrado no site oficial do projeto [8].

3.3 O módulo MSG

O módulo MSG fornece primitivas básicas para a simulação de aplicações distribuídas, onde estas aplicações executam apenas no ambiente de simulação do SimGrid, mais especificamente sobre a plataforma simulada do SURF. Portanto, este módulo se destina principalmente ao estudo e análise de algoritmos paralelos e distribuídos.

De uma forma geral, a simulação é realizada através da execução de vários processos do MSG que são distribuídos entre os recursos computacionais, isto é, os *hosts* do ambiente de execução. Estes processos podem executar de forma concorrente, ou seja, mais de um processo pode ser disparado para execução em um único *host*. Além disso, o MSG também fornece primitivas para o gerenciamento de processos, possibilitando, por exemplo, a criação, migração e deleção dinâmica destes.

No entanto, esses processos do MSG atuam somente como agentes de controle da simulação, tendo como função, basicamente, interagir com os outros processos do MSG a

fim de coordenadamente criar e disparar tarefas para execução. Assim, cada tarefa criada possui um peso de computação associado que representa a sua quantidade de operações aritméticas de ponto flutuante. Desta forma, toda a simulação da execução do algoritmo estudado é modelada por meio de tais tarefas.

Os processos distribuídos do MSG interagem entre si somente por meio de envio e recebimento de tarefas. Todavia, cada tarefa também pode possuir, além do peso computacional, dados associados, assim como o seu tamanho em bytes, simulando desta forma toda a troca de mensagens realizada pelo algoritmo estudado.

3.3.1 Modelo de aplicação

Antes de iniciar a simulação, é necessário efetuar uma distribuição inicial dos processos do MSG, ou seja, determinar para cada processo qual o *host* em que ele residirá. Além disso, também é necessário definir cada processo, informando, basicamente, qual rotina o implementa e quais são os seus parâmetros de entrada. Todas estas característas são especificadas em um arquivo XML usado pelo MSG, conhecido como arquivo de implantação da aplicação. Na figura 3.3 é exibido um pequeno exemplo deste arquivo XML.

```
1  <?xml version='1.0'?>
2  <!DOCTYPE platform SYSTEM "simgrid.dtd">
3  <platform version="2">
4  <process host="host2" function="SM">
5  »       <argument value="arq.tsk"/>
6  »       <argument value="1"/>
7  </process>
8  <process host="host2" function="HM">
9  »       <argument value="2"/>
10 </process>
11 <process host="host1" function="HM">
12 »       <argument value="4"/>
13 </process>
14 <process host="host0" function="HM">
15 »       <argument value="1"/>
16 </process>
17 </platform>
```

Figura 3.3: Exemplo do arquivo XML de implantação da aplicação

Neste exemplo são especificados quatro processos do MSG, onde o primeiro executará o código da função implementada para o SM com dois parâmetros de entrada (linhas 5 e 6) no *host*2(linha 4), enquanto os outros três processos executarão o código da função implementada para o HM com um parâmetro de valor 2 no *host*2(linhas 8-10), 4 no *host*1(linhas 11-13) e 1 no *host*0(linhas 14-16) respectivamente.

No código do simulador implementado, as funções SM e HM serão devidamente registradas, isto é, serão atribuídos o nome SM e HM ao código-fonte de suas respectinas rotinas. Observe que o registro das funções é feito por meio de primitivas fornecidas pelo MSG, e que os argumentos passados no arquivo XML devem coincidir com definição de parâmetros nas rotinas implementadas.

Note que apesar da distribuição dos processos da aplicação ser realizada previamente à execução da simulação, ela não é estática, ou seja, esta distribuição pode ser alterada em tempo de execução através da criação, migração e deleção dinâmica de processos. É importante lembrar que estes processos atuam apenas como agentes de controle da simulação, e portanto, o que será simulado, de fato, é o tráfego de tarefas entre os processos e a execução delas.

3.4 Resumo

Este capítulo apresentou sucintamente o *framework* SimGrid, discutindo as funcionalidades oferecidas por cada um de seus principais módulos, resultando na escolha do módulo MSG para ser utilizado na simulação pretendida. Também foi mostrado como descrever o ambiente de simulação a ser utilizado, e também como implantar a aplicação a ser simulada utilizando o módulo MSG.

Capítulo 4

Simulação da estrutura de escalonamento do EasyGrid

Este capítulo descreve a simulação de parte da estrutura de escalonamento do EasyGrid, por meio do módulo MSG do SimGrid. As principais heurísticas do EasyGrid que foram implementadas no simulador são mostradas, assim como as considerações concernentes à implementação do simulador proposto.

4.1 Escopo da simulação

Neste projeto, a estrutura de escalonamento do EasyGrid que foi simulada consiste em uma parte de sua hierarquia de escalonadores, a saber: os escalonadores dinâmicos do nível do site(SDS) e do nível dos *hosts*(HDS). Portanto, o gerenciador global(GM), assim como o escalonador dinâmico global(GDS), não são contemplados no trabalho aqui realizado. Logo, o ambiente de execução simulado consiste apenas de uma única organização virtual(site), que possui diversos recursos computacionais(*hosts*) associados, onde estes recursos são gerenciados apenas pelo gerenciador do site(SM).

Somente aplicações do tipo *bag-of-tasks*(BoT), ou seja, aplicações que consistem de tarefas independentes entre si, são analisadas no escopo deste projeto, isto é, somente as heurísticas de escalonamento do EasyGrid para aplicações BoT são analisadas neste projeto.

4.2 Escalonador dinâmico da máquina - HDS

Como visto no capítulo 2, cada HDS dispara suas tarefas para execução segundo a política local da máquina. As duas principais políticas locais implementadas pelo EasyGrid são a execução concorrente de tarefas e a política HDS ociosa[2]. Neste projeto, foi simulado apenas a política local de execução concorrente de tarefas, onde cada $host_j$ possui um limite máximo de tarefas da aplicação que podem ser executadas concorrentemente, o qual é denotado por nt_j.

Quando uma tarefa da aplicação u termina sua execução no $host_j$, o seu tempo de parede é medido através de primitivas do módulo MSG e o seu tempo de CPU é calculado por meio da equação a seguir, onde $\epsilon(u)$ é o peso de computação da tarefa u e λ_j a capacidade computacional do $host_j$.

$$tempo\_cpu(u) = \frac{\epsilon(u)}{\lambda_j} \tag{4.1}$$

Com estas duas informações, o HDS calcula o percentual de utilização da CPU para o $hostj$ (pcu_j) por meio da Equação 2.1 vista anteriormente, isto é, $pcu_j = \frac{tempo\_cpu(u)}{tempo\_parede(u)} * nt_j$. Em seguida é calculado o poder computacional relativo do $host_j$ (cp_j) através da seguinte equação:

$$cp_j = pcu_j * \lambda_j \tag{4.2}$$

Note que, diferentemente do EasyGrid, o modelo arquitetural do SimGrid não utiliza o índice de retardo computacional(csi), e sim, a capacidade computacional de cada host(λ) que representa quantas operações aritméticas de ponto flutuante por segundo($flops$) o $host$ é capaz de realizar.

O HDS então calcula seu tempo estimado de fim(ert) por meio da equação 2.3, e adiciona o seu cp e o ert à mensagem de monitoramento do HM, simulada por uma tarefa MSG, enviada ao SM que indica o término de execução da tarefa u.

4.2.1 HeartBeat

Enquanto nenhuma tarefa da aplicação termina sua execução, mensagens de $heartbeat$ são geradas periodicamente em um intervalo $I_{heartbeat}$ definido pelo usuário. Estas mensagens são necessárias quando não há tarefas em execução no $host$ em questão, ou quando

as tarefas que estão executando são muito demoradas. Neste caso, o cálculo do percentual de utilização da cpu(*pcu*) se torna mais complicado, pois não há uma última tarefa u recente na qual se possa obter o seu tempo de cpu e o seu tempo de parede.

No EasyGrid, o *pcu* é obtido por meio de chamadas ao sistema operacional que informam a carga de trabalho atualmente experimentada pelo *host*. No entanto, tal método não pode ser utilizado na simulação implementada, visto que toda a simulação realizada executa em um único *host* real, e a carga de trabalho deste não reflete na carga de um determinado *host* do ambiente de execução simulado. Tal informação para o *host* simulado teria que ser obtida por meio de primitivas do próprio SimGrid, porém esta funcionalidade não está implementada na última versão disponível no momento da realização deste projeto, a saber, a versão 3.3.4.

Então, para um dado $host_j$, é realizado uma estimativa do pcu_j que é obtido por meio da equação abaixo, onde π_j é a quantidade de tarefas da aplicação em execução no momento, nt_j é o número máximo de tarefas concorrentes e ϕ_j é a quantidade de carga de trabalho externa que executa no $host_j$.

$$pcu_j = \frac{\pi_j}{nt_j + \phi_j} \tag{4.3}$$

Observe que, para efeito da estimativa do pcu_j, as cargas externas são simuladas como tarefas especiais que também são monitoradas pela camada de monitoramento do EasyGrid implementada, para que desta forma ϕ_j possa ser obtido. A simulação de cargas de trabalho externas é discutida em mais detalhes no próximo capítulo.

Após se obter o pcu_j, se calcula o cp_j e o ert_j por meio das Equações 4.2 e 2.3, e então, estes são adicionados na mensagem de monitoramento do HM. Neste caso, uma mensagem de *heartbeat* é enviada ao SM, visto que ou não existe nenhuma tarefa da aplicação para ser executada, ou tarefas de aplicações muito demoradas estão sendo executadas.

O Algoritmo 3, apresentado a seguir, resume as principais ações para o tratamento de mensagens de *heartbeat*.

Algoritmo 3 : *tratar_HeartBeat()*

1 Se ($I_{heartbeat}$ ultrapassado) então
2 $pcu_j \leftarrow \pi_j/(nt_j + \phi_j)$; /*Equação 4.3*/
3 $cp_j \leftarrow pcu_j * \lambda_j$; /*Equação 4.2*/
4 $ert_j \leftarrow (\sum_{v_i \in V_j} \epsilon(v_i)) * 1/cp_j$; /*Equação 2.3*/
5 $tarefa_{heartbeat} \leftarrow criar\_tarefa\_MSG(0; tam(cp_j) + tam(ert_j) + 1)$;
6 adiciona cp_j, ert_j e $TAG_{heartbeat}$ à $tarefa_{heartbeat}$;
7 envia $tarefa_{heartbeat}$ ao SM;
fim Se

Sempre que uma tarefa da aplicação termina sua execução, a contagem de tempo para o *heartbeat* é reiniciada. Portanto, $I_{heartbeat}$ só é ultrapassado (linha 1) quando se passarem no mínimo $I_{heartbeat}$ segundos sem que nenhuma tarefa da aplicação tenha terminado.

A fim de se simular o envio de mensagens de *heartbeat*, uma tarefa MSG é criada com peso de computação nulo e com peso de comunicação igual à soma dos tamanhos em bytes do cp_j, ert_j e $TAG_{heartbeat}$ respectivamente ($tarefa_{heartbeat}$, linha 5), que, em seguida, são adicionados a ela. Depois de devidamente criada e configurada, a tarefa MSG é finalmente enviada ao SM (linha 7).

Note que, o intervalo de heartbeat($I_{heartbeat}$) desempenha um papel importante, mantendo as estimativas de desempenho dos recursos da grade atualizadas, principalmente quando a aplicação é constituída de tarefas com elevado peso computacional. Entretanto, a sua configuração deve ser realizada com cautela, pois intervalos muito pequenos podem gerar uma sobrecarga de mensagens e uma maior intrusão por parte do sistema de escalonamento, enquanto que intervalos grandes podem não manter as estimativas suficientemente atualizadas.

4.2.2 Interação com o escalonador dinâmico do site

O HDS participa do escalonamento dinâmico do site de duas formas distintas: cedendo tarefas ao SDS quando requisitado; e recebendo do SDS novas tarefas para serem executadas.

Ao receber um pedido de tarefas do SDS, o HDS deve escolher quais tarefas ceder (selecionar_tarefas_HDS, Algoritmo 1, linha 11). O algoritmo abaixo descreve a heurística implementada para a seleção destas tarefas, a qual é utilizada pelo EasyGrid em aplicações BoT.

Algoritmo 4 : *selecionar_tarefas_HDS(perc$_{ask}$)*

```
1   LP ← tarefas prontas em ordem decrescente de peso(ϵ);
2   pacote_HDS ← ∅;
3   pesoPedido ← (∑_{v∈LP} ϵ(v)) * perc_ask;
4   minPeso ← min{ϵ(v)}/v ∈ LP;
5   Enquanto (pesoPedido > minPeso) faça
6       pesoPedido ← pesoPedido - minPeso;
7       remove v de LP;
8       pacote_HDS ← pacote_HDS + v;
9       minPeso ← minϵ(v)/v ∈ LP;
    fim Enquanto
10  Se pacote_HDS ≠ ∅ então
11      tarefa_ACK ← criar_tarefa_MSG(0; tam(pacote_HDS) + 1);
12      adiciona pacote_HDS e TAG_ACK à tarefa_ACK;
13      envia tarefa_ACK ao SDS;
    senão
14      tarefa_NACK ← criar_tarefa_MSG(0; 1);
15      adiciona TAG_NACK à tarefa_NACK;
16      envia tarefa_NACK ao SDS;
    fim Se
```

Como será discutido na seção a seguir, o SDS envia para o HDS um percentual de tarefas a serem cedidas($perc_{ask}$). Desta forma, o peso de tarefas cedidas é proporcional ao peso total das tarefas da aplicação no HDS(linha 3). Basicamente, a heurística implementada visa ceder a maior quantidade de tarefas possíveis de forma a atender o peso pedido, ou seja, as menores tarefas do HDS serão cedidas. A vantagem desta abordagem é que o SDS possivelmente receberá várias tarefas pequenas, facilitando, deste modo, a redistribuição das mesmas entre os outros recursos do site.

Note que, apenas as tarefas prontas, isto é, as tarefas que ainda não foram disparadas para a execução, são consideradas pela heurística implementada. Além disso, caso o HDS possa ceder tarefas, uma única mensagem contendo um *ack* e o pacote com todas as tarefas cedidas($pacote_{HDS}$) é enviada ao SDS. Esta mensagem é simulada como uma tarefa MSG com peso de computação nulo e com peso de comunicação igual à soma do tamanho em bytes do $pacote_{HDS}$ e do TAG_{ACK} ($tarefa_{ACK}$, linha 11). Caso não seja possível ceder tarefas, o HDS envia uma mensagem de *nack* para o SDS, também simulada por uma tarefa MSG ($tarefa_{NACK}$, linha 14).

Devido a reescalonamento dinâmico de tarefas, o HDS pode receber, no decorrer da sua execução, tarefas adicionais a serem executadas. Neste caso, o HDS as inclui em sua lista de tarefas prontas, mantendo-a sempre ordenada de forma decrescente de peso. Com isso, as tarefas de maior peso de computação são sempre selecionadas para a execução,

enquanto que as tarefas menores são as últimas a serem executadas. Tal abordagem é empregada a fim de facilitar a redistribuição de tarefas entre os recursos do site, ou seja, ela torna os eventos de escalonamento mais eficientes, principalmente no final da execução da aplicação.

4.3 Escalonador dinâmico do site - SDS

4.3.1 Escalonamento Estático

Inicialmente, o escalonador dinâmico do site envia, para cada um de seus HDS subordinados, um pacote de tarefas a serem executadas naquele *host*. Esta distribuição inicial das tarefas da aplicação, isto é, o escalonamento estático da aplicação, é realizado por uma heurística gulosa, que é simulada conforme implementado no EasyGrid para aplicações BoT[2], a qual é descrita a seguir.

Algoritmo 5 : *distribuir_tarefas(tarefas, recursos, wload)*

```
1   listaTarefas ← tarefas em ordem decrescente de peso(ϵ);
2   Enquanto (listaTarefas ≠ ∅) faça
3       listaRecursos ← recursos em ordem decrescente de wload;
4       r ← first(listaRecursos); v ← first(listaTarefas);
5       aloca v em r, adicionando v ao pacote_r;
6       wload(r) ← wload(r) − ϵ(v);
7       listaTarefas ← listaTarefas - v;
    fim Enquanto
8   Para todo r_k ∈ recursos faça
9       tarefa_pacoteTarefas ← criar_tarefa_MSG(0; tam(pacote_rk) + 1);
10      adiciona pacote_rk e TAG_pacoteTarefas à tarefa_pacoteTarefas;
11      envia tarefa_pacoteTarefas para r_k;
    fim Para
```

Este algoritmo possui os seguintes parâmetros de entrada: $tarefas$ é o conjunto de tarefas a serem distribuídas; $recursos$ é o conjunto de recursos($hosts$) do site; e $wload$ é a carga de trabalho que cada recurso deseja receber, que é proporcional ao seu desempenho computacional estimado.

Observe que cada pacote de tarefas enviado é simulado por uma tarefa MSG com peso de computação nulo e peso de comunicação igual ao tamanho em bytes do pacote acrescido de 1 byte para o $TAG_{pacoteTarefas}$ (linha 9).

Na fase de distribuição inicial das tarefas da aplicação, a carga de trabalho que o $host_i$ deseja receber ($wload_i$) é calculado pela equação seguinte, onde λ_i é a capacidade

computacional do $host_i$ e $\epsilon(t)$ o peso computacional da tarefa t.

$$wload_i = \frac{\lambda_i}{\sum_{j \in recursos} \lambda_j} * \sum_{t \in tarefas} \epsilon(t) \tag{4.4}$$

De uma forma geral, o algoritmo visa alocar primeiramente as tarefas mais longas nos recursos com maior *wload*, tentando evitar, assim, que na fase final da distribuição existam tarefas maiores do que a carga de trabalho que um recurso ainda pode receber. Esta situação pode ocorrer devido ao fato das tarefas serem indivisíveis.

Note que, este algoritmo é utilizado tanto para a distribuição inicial das tarefas da aplicação, quanto para a redistribuição das mesmas na ocorrência de um evento de escalonamento dinâmico do site, como será visto na seção a seguir.

4.3.2 Escalonamento Dinâmico

A aplicação do usuário inicia a sua execução segundo o escalonamento estático realizado, que utiliza estimativas de desempenho dos recursos do ambiente obtidas em tempo de compilação. Todavia, o reescalonamento dinâmico das tarefas da aplicação pode ser necessário devido a, entre outros fatores, natureza dinâmica e de compartilhamento das grades computacionais, que exige, desta forma, que as estimativas utilizadas sejam atualizadas no decorrer da execução da aplicação.

Conforme discutido na seção 2.1.2, o SDS verifica, de tempos em tempos, a necessidade de disparar um evento de escalonamento de tarefas do site, que, para aplicações BoT, consiste basicamente em um balanceamento das cargas de trabalho de cada um de seus *hosts*. No EasyGrid, este intervalo de verificação, denotado por I_{sched}, é fixo e a sua configuração deve ser feita com diligência, visto que intervalos muito pequenos podem acarretar em uma sobrecarga no sistema, enquanto que intervalos muito grandes podem fazer com que os escalonadores demorem bastante a reagir às mudanças do ambiente de execução [2].

A fim de determinar se um evento de escalonamento de tarefas prontas deve ser disparado no site s_k, o SDS_k verifica se o índice de desbalanceamento do site (sii_k) é maior do que um limite θ pré-definido, o qual representa o índice de desbalanceamento máximo tolerado entre os recursos do site. Para tanto, o SDS_k calcula o tempo de execução restante do site s_k ($sert_k^*$, *site estimated remaining time*) que representa o tempo médio de execução das tarefas prontas que ainda não foram diparadas para execução em s_k no

momento. A equação a seguir é utilizada para o cálculo do $sert_k^*$, onde R_k é o conjunto de recursos do site s_k.

$$sert_k^* = \frac{\sum_{r_j \in R_k} ert_j * cp_j}{\sum_{r_j \in R_k} cp_j} \tag{4.5}$$

Após o cálculo do $sert_k^*$, o SDS_k verifica qual o recurso mais atrasado, isto é, o recurso com maior ert, e, em seguida, verifica o quanto este está distante do tempo alvo do site. Assim, o índice de desbalanceamento do site (sii_k) é calculado, conforme a equação abaixo.

$$sii_k = \frac{max_{r_j \in R_k}\{ert_j\}}{sert_k^*} \tag{4.6}$$

Caso seja necessário ($sii_k > \theta$), é disparado um evento de escalonamento de tarefas no site, que consiste, basicamente, em redistribuir as tarefas entre os seus recursos de forma que todos eles aproximem o seu tempo estimado de fim do tempo de execução alvo do site. Para isso, os recursos sobrecarregados devem ceder tarefas para os recursos subcarregados, onde um recurso r_j é considerado sobrecarregado quando $ert_j > sert_k^*$ e subcarregado quando $ert_j < sert_k^*$.

No EasyGrid, apenas o recurso mais sobrecarregado (r_{max}), isto é, aquele com maior ert, cede suas tarefas excedentes. Estas tarefas são distribuídas entre todos os recursos subcarregados do site, representado pelo conjunto $R_{sub} \subset R_k$. Tal estratégia foi adotada pois minimiza o custo do algoritmo e adapta a execução da aplicação às mudanças da grade de forma gradual [2].

Quando um evento de escalonamento é disparado, é enviado uma mensagem ao recurso r_{max} com o percentual de tarefas ($perc_{ask}$) a serem por ele cedidas, estas tarefas são selecionadas pelo HDS segundo o algoritmo 4 descrito anteriormente. O $perc_{ask}$ é calculado pela equação abaixo.

$$perc_{ask} = 1 - \frac{1}{sii_k} \tag{4.7}$$

Como a quantidade de tarefas prontas em um recurso qualquer pode variar no decorrer do evento de escalonamento, visto que as tarefas da aplicação continuam sendo executadas, usar um valor percentual ao invés de um valor absoluto promove maior flexibilidade, garantindo que nunca será requisitada uma carga de trabalho superior ao que

o recurso r_{max} pode ceder. Além disso, o seu uso evita que um recurso fique oscilando repetidamente entre os estados de subcarregado e de sobrecarregado [2].

O algoritmo *escalonar_prontas_local*() disparado pelo SDS (Algoritmo 2, linha 6) pode ser visualizado abaixo, como implementado na simulação.

Algoritmo 6 : *escalonar_prontas_local()*

1 $R_{ask} \leftarrow \emptyset$; $R_{sub} \leftarrow \emptyset$;
2 $tarefas_{ask} \leftarrow 0$;
3 $sert^*_k \leftarrow (\sum_{r_j \in R_k} ert_j * cp_j) / \sum_{r_j \in R_k} cp_j$; /*Equação 4.5*/
4 $sii_k \leftarrow max_{r_j \in R_k}\{ert_j\}/sert^*_k$; /*Equação 4.6*/
5 $perc_{ask} \leftarrow 1 - 1/sii_k$; /*Equação 4.7*/
6 Se ($perc_{ask} > \theta_{\%}$) então
7 determina r_{max} e R_{sub};
8 $R_{ask} \leftarrow r_{max}$;
9 $tarefas_{ask} \leftarrow perc_{ask}$;
Fim Se
10 retorna R_{ask}, R_{sub} e $tarefas_{ask}$ para o SDS;

Observe que, para a simulação implementada, um evento de escalonamento é disparado quando $perc_{ask} > \theta_{\%}$ ao invés de quando $sii_k > \theta$ como proposto no EasyGrid. Tal alteração foi realizada meramente pela conveniência de se trabalhar apenas com valores percentuais, isto é, valores pertencentes ao intervalo contínuo de 0 até 1, já que o sii_k utilizado anteriormente é sempre maior que 1. Portanto, o $\theta_{\%}$ utilizado na simulação representa o valor percentual máximo de desbalanceamento permitido no site.

O HDS_{max} alocado no recurso r_{max}, ao receber uma mensagem com a percentagem pedida de tarefas pelo SDS, verifica se é possível atendê-lo, enviando um *ack* juntamente com o pacote de tarefas cedido caso positivo, ou um *nack* caso negativo (algoritmo 4).

Assim que o SDS recebe um pacote com as tarefas cedidas por r_{max} ($pacote_{HDS}$), ele deve redistribuí-las entre os recursos do seu site s_k de acordo com a quantidade total de tarefas recebidas. Cada recurso $r_j \in R_{sub}$ deverá receber uma carga de trabalho ($wload_{r_j}$) que equivale a uma parte do peso total destas tarefas (W_{HDS}), conforme as equações a seguir.

$$W_{HDS} = \sum_{v \in pacote_{HDS}} \epsilon(v) \tag{4.8}$$

$$perc_{r_j} = \frac{(sert^*_k - ert_j) * cp_j}{\sum_{r_i \in R_{sub}}((sert^*_k - ert_i) * cp_i)} \tag{4.9}$$

$$wload_{r_j} = perc_{r_j} * W_{HDS} \tag{4.10}$$

Após calculado a carga de trabalho que cada recurso subcarregado deve receber, o SDS simulado utiliza a mesma heurística apresentada no algoritmo 5 para redistribuir as tarefas cedidas entre estes recursos, onde os parâmetros de entrada são: $distribuir\_tarefas(pacote_{HDS}, R_{sub}, wload_r)$.

4.4 Considerações referentes a simulação

4.4.1 Simulação da comunicação

Como visto nos algoritmos apresentados e descritos no capítulo anterior, todas as trocas de mensagem entre os processos gerenciadores da aplicação são simuladas por meio do envio e recebimento de tarefas MSG, cada uma simulando uma mensagem e contendo, desta forma, um peso de comunicação associado, que representa o tamanho da mensagem em bytes. No entanto, como o objetivo de tais tarefas é apenas simular a transferência de dados, seus pesos de computação são nulos.

As tarefas da aplicação também são simuladas por meio de tarefas MSG, possuindo tanto um peso de computação associado, o qual é dado em quantidade de operações de ponto flutuante, quanto um peso de comunicação, que representa os dados, em bytes, necessários para a execução da tarefa. Assim, toda vez que há um evento de escalonamento, os pesos de comunicação das tarefas da aplicação que participam do evento, isto é, o peso de comunicação do $pacote_{HDS}$, são sempre levados em consideração pelo simulador (confira $tarefa_{ACK}$, Algoritmo 4, linha 11), influenciando na latência de comunicação de forma semelhante ao ocorrido em ambientes reais de execução.

4.4.2 Simulação dos processos gerenciadores

Os processos gerenciadores do EasyGrid simulados não ficam em execução contínua durante todo o tempo, ao invés disso, eles executam uma iteração a cada período fixo de tempo, denotado por $I_{monitor}$. Tal abordagem visa reduzir a sua interferência no tempo total de execução da aplicação monitorada.

Como discutido no Capítulo 3, toda a simulação tem que ser modelada por meio

de tarefas MSG. Portanto, é necessário criar tais tarefas para simular a execução dos processos gerenciadores implementados, ou seja, dos HMs e do SM. Assim, para cada iteração de um processo gerenciador é criada uma tarefa MSG para simulá-la, sendo ela executada no final da iteração. No entanto, determinar o peso de computação destas tarefas se torna complicado, pois nem todas as iterações possuem o mesmo peso. Mesmo duas iterações de um dado processo gerenciador podem ter tempos de execução bem diferentes entre si, por exemplo, em uma iteração pode ser apenas verificado que não tem nenhuma mensagem para receber, enquanto que na outra iteração pode ser tratado todo um evento de escalonamento.

Com o intuito de se obter uma estimativa para o peso de computação da tarefa MSG v que simula uma iteração i_n do gerenciador que executa no $host_i$, é medido, por meio de primitivas do sistema operacional, o tempo de execução real da iteração i_n ($t_{i_n}^{real}$), isto é, o tempo em que ela levou para executar na máquina real onde o simulador executa. Então, o peso computacional da tarefa MSG v ($\epsilon_{msg}(v)$) em quantidade de operações de ponto flutuante é dado pela equação abaixo, onde λ_i é a capacidade computacional do $host_i$ em $flops$.

$$\epsilon_{msg}(v) = t_{i_n}^{real} * \lambda_i \tag{4.11}$$

Note que, tal estimativa é dependente da máquina real sendo utilizada para a simulação, cujo poder computacional oferecido pode afetar o tempo de iteração simulado. Portanto, o ideal é que se utilize sempre a mesma máquina, com uma carga de trabalho fixa, para realizar a simulação pretendida.

4.5 Resumo

Este capítulo descreveu a simulação de parte da estrutura de escalonamento do EasyGrid, mostrando as principais heurísticas utilizadas para aplicações BoT, entre elas a heurística para o escalonamento estático de tarefas, para a seleção de tarefas a serem cedidas pelo HDS, e para o escalonamento dinâmico das tarefas do site. Além disso, foram feitas considerações concernentes à implementação do simulador, como a simulação das iterações dos processos gerenciadores do EasyGrid.

Capítulo 5

Análise experimental

Neste capítulo são especificados os ambientes de execução e as aplicações simuladas, além dos parâmetros de execução utilizados pelo sistema de escalonamento. Os experimentos realizados possuem três finalidades distintas: analisar a escalabilidade do processo gerenciador do site (SM); avaliar o custo adicional introduzido pelo escalonamento dinâmico de tarefas; e analisar os possíveis benefícios da utilização do escalonamento dinâmico de tarefas.

5.1 Ambientes de execução simulados

Nos experimentos realizados foram utilizados dois tipos de ambientes de execução: ambientes homogêneos e ambientes heterogêneos. Em ambos, cada *host* é conectado diretamente a todos os demais e todos os enlaces possuem a mesma largura de banda, que é de 100Mbps, e a mesma latência, que é de 1 ms.

Nos ambientes homogêneos, todos os *hosts* possuem a mesma capacidade computacional, que é de 1Gflops. Já nos ambientes heterogêneos simulados, os *hosts* possuem capacidades computacionais diferentes, onde sua primeira metade é de 1Gflops enquanto que sua segunda metade é de 500Mflops.

Os ambientes de execução simulados ainda podem ser classificados como sendo dedicados ou compartilhados. No primeiro caso, todo o ambiente de execução é dedicado à aplicação do usuário, enquanto que no outro caso, podem haver aplicações externas executando enquanto a aplicação do usuário é monitorada.

5.1.1 Simulação de aplicações externas

Nos ambientes de execução compartilhados, as aplicações externas são simuladas por meio da introdução de cargas de trabalho *CPU-bound* nos *hosts* pretendidos. Com intuito de se obter um maior controle sobre o ambiente simulado, tais cargas externas são modeladas como tarefas MSG criadas pelo próprio SM, que as distribui entre os *hosts* alvos.

Note que, essas tarefas MSG de carga externa podem ser criadas a qualquer momento pelo SM e o peso de comunicação de cada uma delas é nulo, pois a idéia é simular somente a sua execução no *host* pretendido.

Conforme discutido no capítulo anterior, tal abordagem favorece o cálculo do *pcu* na ocorrência de eventos de *hearbeat*, visto que as cargas externas são tratadas como tarefas especiais que são monitoradas pela camada de monitoramento simulada, que, desta forma, passa a ter total ciência da execução das mesmas.

5.2 Aplicações simuladas

Como especificado anteriormente, este projeto contempla apenas aplicações do tipo *bag-of-tasks*, e, nos experimentos realizados, todas as tarefas das aplicações simuladas possuem um mesmo peso de computação, de 5Gflops. Além disso, cada tarefa da aplicação possui um peso de comunicação de 1MB, representando a quantidade de dados necessários para sua execução, isto é, seus dados de entrada.

5.3 Parâmetros utilizados

A Tabela 5.1 mostrada a seguir lista os parâmetros de execução utilizados pelo sistema de escalonamento simulado, onde nt_j denota o número de tarefas concorrentes a serem disparadas pelo HM; $I_{monitor}$ representa o intervalo de execução dos processos gerenciadores; I_{sched} denota o intervalo mínimo de verificação do escalonamento; $I_{heartbeat}$ representa o intervalo para eventos de *heartbeat*; e $\theta_{\%}$ denota o índice máximo do percentual de desbalanceamento tolerado.

A influência de cada parâmetro de execução no desempenho do sistema de escalonamento foi discutida nos capítulos anteriores. A configuração de parâmetros apresen-

nt_j	$I_{monitor}$	I_{sched}	$I_{heartbeat}$	$\theta_{\%}$
2	1s	2s	5s	20%

Tabela 5.1: Parâmetros de execução utilizados

tada foi utilizada em todos os experimentos, e foi escolhida arbitrariamente, levando-se em conta as considerações previamente realizadas. Observe que, a melhor configuração possível para estes parâmetros depende de características específicas tanto da aplicação quanto do ambiente de execução simulados.

5.4 Experimentos

Todos os experimentos foram realizados utilizando um sistema computacional com a seguinte configuração: processador AMD Athlon 64 de 1.6Ghz, 1.8GB de memória RAM, e sistema operacional Fedora Linux 12 com kernel 2.6.31.5-127 de 32 bits. A versão do SimGrid utilizada foi a mais recente disponível no momento de realização do projeto, 3.3.4.

Todos os resultados apresentados foram obtidos a partir da média de pelo menos cinco execuções. Note que, apesar do ambiente de execução simulado ser totalmente controlado, podem haver pequenas variações no tempo final de simulação, visto que o SimGrid modela o não determinismo do protocolo TCP [6, 7] e que, como visto no capítulo anterior, as iterações simuladas dos gerenciadores são baseadas no tempo de execução real do simulador, o qual pode variar de uma execução para outra. No entanto, conforme será mostrado, o desvio padrão obtido é relativamente pequeno.

Em todos os experimentos realizados, os processos gerenciadores SM e HM executaram no mesmo *host* do ambiente de execução simulado, ou seja, o *host* em que o SM residiu também executou tarefas da aplicação.

5.4.1 Escalabilidade do sistema de escalonamento simulado

Como neste projeto foi implementada apenas uma parte da hierarquia de escalonadores do EasyGrid, o SM passa então a gerenciar todos os recursos do ambiente, que constituem um único site, se tornando, desta forma, um potencial gargalho para o desempenho do sistema de escalonamento.

A fim de se verificar a escalabilidade do sistema de escalonamento simulado, foram realizados experimentos com o escalonamento dinâmico de tarefas ativado, considerando ambientes homogêneos e dedicados de diferentes tamanhos, e aplicações com 500 e 5000 tarefas. As Tabelas 5.2 e 5.3 mostram os resultados obtidos, onde: $\#Hosts$ representa a quantidade total de *hosts* no ambiente; $\overline{t_{simul}}$ denota a média do tempo total de simulação; $\delta_{t_{simul}}$ representa o desvio padrão amostral de t_{simul}; $\overline{t_{app}}$ denota a média do tempo de fim da última tarefa da aplicação; e $\delta_{t_{app}}$ representa o desvio padrão amostral de t_{app}.

$\#Hosts$	$\overline{t_{simul}}$	$\delta_{t_{simul}}$	$\overline{t_{app}}$	$\delta_{t_{app}}$
50	59,240898 s	0,009417 s	56,208582 s	0,009359 s
100	30,072965 s	0,022516 s	27,088634 s	0,022625 s
400	22,243691 s	0,059415 s	11,241162 s	0,063170 s

Tabela 5.2: Ambiente homogêneo e dedicado com 500 tarefas

$\#Hosts$	$\overline{t_{simul}}$	$\delta_{t_{simul}}$	$\overline{t_{app}}$	$\delta_{t_{app}}$
50	527,181000 s	2,113536 s	524,584996 s	2,280615 s
100	268,578064 s	0,360518 s	265,352490 s	0,010693 s
400	173,106070 s	2,658534 s	67,405276 s	0,063756 s

Tabela 5.3: Ambiente homogêneo e dedicado com 5000 tarefas

Note que, o desvio padrão amostral do tempo de simulação ($\delta_{t_{simul}}$) foi relativamente pequeno, não ultrapassando 0,4% de $\overline{t_{simul}}$, exceto para o caso de 5000 tarefas e 400 *hosts* onde seu valor foi cerca de 1,5% de $\overline{t_{simul}}$. O desvio padrão amostral de t_{app} ($\delta_{t_{app}}$) se comportou de forma bem similar, exceto para o caso de 5000 tarefas e 400 *host* onde foi significativamente menor, devido à grande diferença obtida entre $\overline{t_{simul}}$ e $\overline{t_{app}}$, entre outros fatores.

A métrica utilizada para se analisar os resultados é a diferença entre a média do tempo total de simulação ($\overline{t_{simul}}$) e a média do tempo de fim da última tarefa da aplicação ($\overline{t_{app}}$). Ela mede, basicamente, o tempo de finalização do sistema de escalonamento simulado, que reflete justamente na questão da escalabilidade do mesmo, visto que nesta fase o SM envia uma mensagem de fim para todos os HMs, e, em seguida, todos eles enviam, quase que simultaneamente, um mensagem de fim_ack ao SM.

Observe que, independentemente da quantidade de tarefas simulada, para 50 e 100

hosts a diferença entre $\overline{t_{simul}}$ e $\overline{t_{app}}$ é bem pequena, cerca de 3 segundos apenas. Já para 400 *hosts*, o $\overline{t_{simul}}$ é quase o dobro do $\overline{t_{app}}$, mostrando desta forma uma grande sobrecarga no SM.

Portanto, é possível notar que o emprego de um único gerenciador de site (SM) pode não ser suficiente para o gerenciamento eficiente de todos os recursos da organização virtual, e que dependendo de sua dimensão, pode ser interessante especificar dois ou mais SMs para o mesmo site.

5.4.2 Análise do escalonamento dinâmico de tarefas

Com o intuito de se avaliar o custo do escalonamento dinâmico de tarefas, foram realizados experimentos com e sem o uso do escalonamento dinâmico, considerando ambientes heterogêneos e dedicados de 50 e 100 *hosts*, e aplicações com 500 e 5000 tarefas. Com o escalonamento dinâmico desativado, as tarefas são distribuídas conforme a política de escalonamento estático implementada (Algoritmo 5), e, após esta distribuição inicial, sua alocação não sofre mais mudanças.

Os resultados obtidos são exibidos nas Tabelas 5.4 e 5.5, onde: $\#Tarefas$ é a quantidade de tarefas da aplicação; $\#Hosts$ é a quantidade total de hosts do ambiente; $\overline{t_{simul}}$ é o tempo médio total de simulação; $\delta_{t_{simul}}$ é o desvio padrão amostral de t_{simul}; $\overline{\#Total\_Eventos}$ é a quantidade média de eventos de escalonamento disparados; e $\overline{\#Eventos\_Aceitos}$ é a quantidade média de eventos de escalonamento realizados.

$\#Tarefas$	$\#Hosts$	$\overline{t_{simul}}$	$\delta_{t_{simul}}$	$\overline{\#Total\_Eventos}$	$\overline{\#Eventos\_Aceitos}$
500	50	78,419101 s	3,224371 s	15,8	10,6
	100	44,800593 s	0,011101 s	7	1
5000	50	731,341112 s	3,369626 s	124	116,8
	100	384,273771 s	4,113272 s	73	67,2

Tabela 5.4: Ambiente heterogêneo e dedicado com escalonamento dinâmico ativado

Note que, mesmo o ambiente de execução sendo dedicado, eventos de escalonamento dinâmico de tarefas são realizados. Isto ocorre pois podem haver atrasos nas comunicações realizadas entre os processos gerenciadores da aplicação, além de atrasos no monitoramento das tarefas executadas. Tais atrasos são possíveis devido ao fato de que, conforme discutido anteriormente, os gerenciadores não executam continuamente,

$\#Tarefas$	$\#Hosts$	$\overline{t_{simul}}$	$\delta_{t_{simul}}$	$\overline{\#Total\_Eventos}$	$\overline{\#Eventos\_Aceitos}$
500	50	76,106316 s	0,173506 s	0	0
	100	38,117954 s	0,130823 s	0	0
5000	50	704,169281 s	0,531242 s	0	0
	100	349,547319 s	0,193903 s	0	0

Tabela 5.5: Ambiente heterogêneo e dedicado com escalonamento dinâmico desativado

mas sim, uma iteração a cada $I_{monitor}$ segundos.

Observe também que, o desvio padrão amostral para o t_{simul} é consideravelmente maior quando o escalomento dinâmico de tarefas está ativado, pois nestes casos a quantidade de eventos de escalonamento disparados, assim como os *hosts* neles envolvidos podem variar de execução para execução. No entanto, tal valor de desvio padrão é aceitável, atingindo o máximo de 4,11% de $\overline{t_{simul}}$ no caso de 500 tarefas e 50 *hosts*, e o mínimo de 0,02% de $\overline{t_{simul}}$ no caso de 500 tarefas e 100 *hosts*.

A diferença entre as médias do tempo total de simulação com e sem o uso do escalonamento dinâmico de tarefas é usada como métrica avaliadora, já que, como os ambientes são dedicados, seu valor representa justamente o custo adicional (*overhead*) do uso do escalonamento dinâmico.

É possível observar que, essa diferença aumenta de acordo com o número de tarefas da aplicação e que, portanto, o custo do escalonamento dinâmico de tarefas é dependente do número de tarefas da aplicação. É natural que tal fenômeno ocorra, visto que com aplicações maiores podem existir mais tarefas a serem tratadas em cada evento de escalonamento, o que implica em maior custo final. A idéia é que este custo adicional resulte em um ganho significativo no tempo de execução final (*makespan*) da aplicação, o que não acontece neste caso devido ao fato de o ambiente ser dedicado, e, portanto, não existir a necessidade de se realizar o escalonamento dinâmico de tarefas.

Para os experimentos realizados, cujos resultados são exibidos nas Tabelas 5.4 e 5.5, podemos verificar que o maior custo foi obtido ao se utilizar 100 *hosts* e 500 tarefas, representando um aumento de 17,53% do tempo total de simulação com o uso do escalonamento dinâmico de tarefas, enquanto que o menor custo se deu com 500 tarefas e 50 *hosts*, representando somente 3,04% de aumento. O custo médio obtido nos experimentos equivale a um aumento médio de 8,59% no tempo total de simulação.

5.4.3 Benefícios do escalonamento dinâmico de tarefas

Na tentativa de se analisar os benefícios do escalonamento dinâmico de tarefas, experimentos similares aos anteriores foram realizados, porém neste caso são considerados ambientes compartilhados. Para simular tais ambientes, uma carga externa foi introduzida nos recursos mais lentos, que equivalem à metade do total de recursos simulados. Além disso, esta carga é disparada por volta da metade do tempo total de simulação da aplicação sem o uso do escalonamento dinâmico, o qual foi medido nos experimentos anteriores. Por exemplo, para a simulação da aplicação de 500 tarefas em 100 *hosts*, a carga externa é introduzida nos *hosts* de 50 até 99, no tempo de aproximadamente 19 segundos que é a metade do tempo total de simulação obtido para este caso (vide a Tabela 5.5). A carga externa introduzida é do tipo *CPU-bound* e persiste até o final da simulação.

Os resultados obtidos são mostrados nas Tabelas 5.6 e 5.7, onde: $\#Tarefas$ é a quantidade de tarefas da aplicação; $\#Hosts$ é a quantidade total de hosts do ambiente; $\overline{t_{simul}}$ é o tempo médio total de simulação; $\overline{\#Total\_Eventos}$ é a quantidade média de eventos de escalonamento disparados; e $\overline{\#Eventos\_Aceitos}$ é a quantidade média de eventos de escalonamento realizados.

$\#Tarefas$	$\#Hosts$	$\overline{t_{simul}}$	$\overline{\#Total\_Eventos}$	$\overline{\#Eventos\_Aceitos}$
500	50	97,141766 s	17	10
	100	55,530759 s	6	1
5000	50	794,827179 s	142	128,6
	100	447,085447 s	84,4	74,8

Tabela 5.6: Ambiente heterogêneo e compartilhado com escalonamento dinâmico ativado

$\#Tarefas$	$\#Hosts$	$\overline{t_{simul}}$	$\overline{\#Total\_Eventos}$	$\overline{\#Eventos\_Aceitos}$
500	50	96,965941 s	0	0
	100	45,387222 s	0	0
5000	50	875,554608 s	0	0
	100	436,333836 s	0	0

Tabela 5.7: Ambiente heterogêneo e compartilhado com escalonamento dinâmico desativado

A mesma métrica dos experimentos anteriores, isto é, a diferença entre os tempos

médios de simulação com e sem o uso do escalonamento dinâmico, é utilizada para a avaliação dos resultados. Neste caso, ela representa o ganho, ou perda, no tempo total de simulação (*makespan*) da aplicação.

Note que, para o caso da aplicação com 5000 tarefas em um ambiente com 50 *hosts*, o emprego do escalonamento dinâmico de tarefas implicou em uma redução de 9,22% no tempo total simulado. No entanto, para os outros casos, houve um aumento deste tempo, chegando a atingir 22,35% do tempo total simulado para 500 tarefas e 100 *hosts*. Uma das possíveis causas para tanto foi a configuração inadequada dos parâmetros de simulação utilizados, tais como o $I_{monitor}$ e o I_{sched}, que podem implicar em uma grande sobrecarga no sistema de escalonamento simulado. Outro possível fator contribuinte para tal fenômeno foi o reduzido *makespan* das aplicações analisadas, o qual pode não justificar o uso de escalonamento dinâmico de suas tarefas.

Portanto, podemos concluir apenas que a utilização do escalonamento dinâmico de tarefas com os parâmetros previamente especificados, para os casos específicos analisados nesta subseção e para a atual versão do simulador implementado, não resultou em ganhos significativos em relação ao tempo total de simulação da aplicação, com exceção do caso de 5000 tarefas e 50 *hosts* onde foi obtido um ganho de 9,22% do tempo médio de simulação.

5.5 Resumo

Neste capítulo foram especificados os ambientes de execução e as aplicações simuladas, assim como os parâmetros de execução utilizados pelos processos gerenciadores. Primeiramente foram realizados experimentos a fim de se analisar a escalabilidade do SM, concluindo-se que um único SM pode não ser suficiente para gerenciar de forma eficiente todos recursos do site. Depois, foram feitos experimentos com o intuito de se avaliar o custo adicional introduzido pelo escalonamento dinâmico de tarefas, que para os casos testados representou, em média, um aumento de 8,59% do tempo total de simulação. Por último, foram realizados experimentos para se analisar os possíveis benefícios da utilização do escalonamento dinâmico de tarefas, chegando-se a conclusão de que, para a maioria dos casos analisados e para os parâmetros de execução utilizados, não foi obtida uma redução significativa do tempo total de simulação da aplicação.

Capítulo 6

Conclusão e Trabalhos Futuros

Ambientes de execução distribuídos, como as grades computacionais, vêm se tornando cada vez mais populares, devido a sua capacidade de oferecer um grande poder computacional a um baixo custo. No entanto, tais ambientes geralmente são heterogêneos, compartilhados e podem possuir diferentes políticas de acesso entre os seus recursos. Com isso, torna-se necessário o uso de um sistema gerenciador de aplicações paralelas e distribuídas a fim de proporcionar uma execução segura e eficiente das mesmas.

Um ponto fundamental para uma execução eficiente é que o sistema gerenciador possua uma estrutura de escalonamento capaz de perceber as variações do ambiente e, assim, adaptar a execução da aplicação de acordo com o poder computacional disponível, em outras palavras, o escalonamento dinâmico das tarefas da aplicação desempenha um papel importante na eficiência da aplicação gerenciada. Neste contexto, vem sido desenvolvido na Universidade Federal Fluminense o *middleware* EasyGrid [1], que possui uma estrutura de escalonamento dinâmico escalável, flexível e eficiente [2].

Neste projeto foi implementado um simulador, por meio do *framework* de simulação SimGrid[5], para simular uma versão de uma hierarquia do sistema de escalonamento do EasyGrid que consiste nos processos gerenciadores de máquina (HM) e no processo gerenciador do site (SM), contemplando somente aplicações do tipo *bag-of-tasks*. O objetivo principal foi o de prover uma ferramenta complementar para a análise e avaliação de tal estrutura de escalonamento, sendo capaz de simular uma grande variedade de ambientes distribuídos.

Como trabalhos futuros pretende-se simular experimentos já realizados em ambientes de execução reais, a fim de se analisar a fidelidade do modelo de simulação utilizado.

Além disso, pretende-se também simular toda a estrutura de escalonamento dinâmico do EasyGrid, conforme proposta em [2], o que inclui a simulação da política HDS/ociosa, a simulação do processo gerenciador global (GM) e a contemplação de aplicações com relações de precedência entre suas tarefas, isto é, aplicações DAG. Com todas estas funcionalidades implementadas, pretende-se utilizar o simulador previamente a execução da aplicação monitorada, a fim de ajustar seus parâmetros de execução, melhorando assim o desempenho do *middleware*. Além disso, a própria hierarquia de escalonadores poderia ser ajustada, especificando, por exemplo, o emprego de mais de um SM para sites com grande quantidade de recursos, o que poderia contornar o problema de escalabilidade do SM. Também se pretende analisar o emprego de outras heurísticas na estrutura de escalonamento do EasyGrid, além de simular outras políticas de acesso aos recursos do ambiente de execução. E finalmente, pretende-se realizar extensos experimentos para diversos cenários de execução distintos, com o intuito de complementar as análises já realizadas em ambientes de execução reais.

Referências Bibliográficas

[1] BOERES, Maria Cristina Silva; REBELLO, Vinod Eugene. EasyGrid: Towards a framework for the automatic grid enabling of legacy MPI applications. *Concurrency and Computation: Practice and Experience*, Malden(USA), 25 abr. 2004. v.16, p.425-432.

[2] NASCIMENTO, Aline de Paula. *Escalonamento Dinâmico para Aplicações Autonômicas MPI em Grades Computacionais*. 208f. Tese (Doutorado em Ciência da Computação)-Instituto de Computação, Universidade Federal Fluminense, Niterói. 2008.

[3] MENDES, Helder de Amorim. *HlogP: um modelo de escalonamento para a execução de aplicações MPI em grades computacionais*. Niterói, 2004. 139f. Dissertação (Mestrado em Ciência da Computação)-Instituto de Computação, Universidade Federal Fluminense, Niterói. 2004.

[4] NASCIMENTO, Aline de Paula; SENA, Alexandre da Costa; SILVA, Jaques A. da; VIANNA, Daniela Q. C.; BOERES, Maria Cristina Silva; REBELLO, Vinod Eugene. Managing the execution of large scale MPI applications on computational grids. In: SYMPOSIUM ON COMPUTER ARCHITETURE AND HIGH PERFORMANCE COMPUTING (SBAC-PAD), 17., 2005, Rio de Janeiro. *Anais...* Rio de Janeiro: IEEE Computer Society, 2005. 8f. p.69-76.

[5] CASANOVA, Henri; LEGRAND, Arnaud; QUINSON, Martin. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In: IEEE INTERNATION CONFERENCE ON COMPUTER MODELING AND SIMULATION, 10., 2008, Cambridge/UK. *Anais...* Cambridge/UK: IEEE Computer Society, 2008. 6f. p.126-131.

[6] CASANOVA, Henri; FUJIWARA, Kayo. Speed and Accuracy of Network Simulation in the SimGrid Framework. In: INTERNATIONAL WORKSHOP ON NETWORK SIMULATION TOOLS, 2007, Pisa/Italy. *Anais...* Pisa/Italy:[s.n.], 2007.

[7] MASSOULIÉ, Laurent; ROBERTS, James. Bandwidth Sharing: Objectives and Algorithms. In: Annual Joint Conference of the IEEE Computer and Communications Societies, 18., 1999, New York/USA. *Anais...* New York/USA: [s.n.], 1999. 9f. v.3, p.1395-1403.

[8] UNIVERSITY OF HAWAII AT MANOA (GRENOBLE/FRANCE) AND UNIVERSITY OF NANCY (NANCY/FRANCE). *SimGrid*. Website oficial do projeto SimGrid. Disponível em: <http://simgrid.gforge.inria.fr/doc/index.html>. Acesso em: junho de 2010.

[9] FOSTER, Ian; KESSELMAN, Carl; TUECKE, Steven. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputing Applications*, United States, Ago. 2001. Volume 15, Número 3, p.200-222.

[10] FREY,James; TANNENBAUM, Todd; LIVNY, Miron, FOSTER, Ian, TUECKE, Steven. Condor-G: A computational management agent for multi-institutional grids. *Journal of Cluster Computing*, [s.l.], Jul. 2002. Volume 5, Número 3, p.237-246.

[11] CAO, Junwei; STEPHEN, Jarvis; SAINI, Subhash; NUDD, Graham. GridFlow: workflow management for grid computing. In: INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID (CCGrid 2003), 3., 2003, Tokio/Japan. *Anais...* [s.n]: IEEE Computer Society, 2003. 8f. p.198-205.

[12] BERMAN, Francine; CHIEN, Andrew; COOPER, Keith; DONGARRA, Jack; FOSTER, Ian; GANNON, Dennis; JOHNSSON, Lennart; KENNEDY, Ken; KESSELMAN, Carl; MELLOR-CRUMMEY, John; REED, Dan; TORCSON, Linda; WOLSKI, Rich. The GrADS Project: Software support for high-level Grid application development. *The International Journal of High Performance Computing Applications*, [s.n.], 2001. Volume 4, Número 15, p.327-344.