Biblioteca de Escalonamento de Tarefas em *Grid* Computacional - *LIBTS*

Patrícia B. Franco[1], Roberta Spolon[2], Marcos A. Cavenaghi[2], Renata S. Lobato[3]

[1]Programa de Pós-Graduação em Ciência da Computação
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), Bauru, SP - Brasil

[2]Departamento de Computação/FC
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), Bauru, SP - Brasil

[3]Departamento de Ciências de Computação e Estatística/IBILCE
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP) – São José do Rio Preto, SP – Brasil

patbfranco@yahoo.com.br, {roberta,marcos}@fc.unesp.br,
renata@ibilce.unesp.br

Abstract. *The use of computational grid simulators is particularly important for studying the algorithms of task scheduling. Through the simulators it's possible to assess and compare the performance of different algorithms in various scenarios. Despite the simulation tools provide basic features for simulation in distributed environments, they don't offer internal policies of task scheduling, so that the implementation of the algorithms must be realized by the user himself. Therefore, this study aims to present the library of task scheduling LIBTS (LIBrary Tasks Scheduling) which is developed and adapted to the SimGrid simulator to provide the users with a tool to analyze the algorithms of task scheduling in the computational grid.*

Resumo. *O uso de simuladores de grid computacional é de especial importância para o estudo de algoritmos de escalonamento de tarefas. Através dos simuladores é possível avaliar e comparar o desempenho de diferentes algoritmos em diferentes cenários. Apesar das ferramentas de simulação fornecerem funcionalidades básicas para simulação de ambientes distribuídos, não disponibilizam políticas internas de escalonamento de tarefas, a implementação dos algoritmos deve ser feita pelo próprio usuário. Esse trabalho apresenta a biblioteca de escalonamento de tarefas LIBTS (Library Tasks Scheduling) desenvolvida e adaptada ao simulador SimGrid para oferecer aos usuários uma ferramenta para o estudo de algoritmos de escalonamento de tarefas em grid computacional.*

1. Introdução

Grid computacional é uma infraestrutura de computação que conecta múltiplos recursos computacionais de diversos computadores, para permitir a execução de aplicações com alta demanda de recursos, memória e espaço em disco [Foster et al 2001].

As características próprias do ambiente de *grid* computacional, como a heterogeneidade, comportamento dinâmico, distribuição em larga escala e compartilhamento de recursos, demonstram a complexidade desse ambiente. O problema de escalonamento é uma questão importante em um ambiente de *grid* computacional, pois um algoritmo de escalonamento eficiente deve fazer a distribuição das tarefas para os recursos apropriados, melhorando assim o desempenho da aplicação [Cirne at al 2007, Cho-Chin and Chun-Wei 2008, Yu and Zhou 2007].

O uso de simuladores para ambientes de *grid* computacional é de especial relevância para o estudo de algoritmos de escalonamento de tarefas, pois, através das ferramentas de simulação, torna-se possível analisar e comparar em diferentes cenários, o desempenho de diferentes algoritmos. Diversas ferramentas de simulação foram desenvolvidas para esse propósito, como Bricks, GridSim, MicroGrid, OptorSim, SimGrid [Buyya 2002, Casanova 2001, SimGrid 2011].

As ferramentas de simulação oferecem funções básicas e abstrações para a simulação de aplicações em ambientes distribuídos heterogêneos. Apesar disto, não disponibilizam políticas internas de escalonamento de tarefas, a implementação dos algoritmos deve ser feita pelo próprio usuário. Portanto o objetivo deste trabalho é apresentar a biblioteca de escalonamento de tarefas *Library Tasks Schedules* (LIBTS) com os algoritmos *Workqueue* (WQ), *Workqueue with Replication* (WQR), *Sufferage*, *XSufferage*, *Dynamic* FPLTF. A biblioteca foi integrada ao simulador SimGrid para auxiliar no estudo dos algoritmos de escalonamento de tarefas em *grid* computacional, permitindo aos usuários dedicar-se ao estudo das políticas de escalonamento existentes através da comparação dos algoritmos implementados, da análise dos resultados, assim como a implementação de novos algoritmos.

Dessa forma, este trabalho está organizado nas seguintes seções: na seção 2 são apresentadas uma síntese sobre escalonamento de tarefas em *grid* computacional e as políticas de escalonamento de tarefas em *grid* utilizadas neste trabalho; na seção 3 são descritas a biblioteca LIBTS e suas características. Na seção 4 são apresentadas as conclusões obtidas.

2. Escalonamento de Tarefas em *Grid* Computacional

Escalonamento em *grid* computacional é o processo de tomar decisões de escalonamento envolvendo recursos sobre múltiplos domínios administrativos. A realização de um escalonamento é feita através da ativação de um conjunto de regras que ditam como e quando determinadas informações do sistema devem ser colhidas, de que maneira essas informações influenciam na distribuição de tarefas e quais serão os recursos utilizados para a execução das aplicações. Assim, os algoritmos de escalonamento são utilizados para implementar as regras de uma política de escalonamento. Enquanto as políticas de escalonamento ditam as regras gerais de como lidar com processos e administrar recursos do sistema, os algoritmos de escalonamento estão preocupados com a implementação dessas regras que podem ser feitas de diversas formas [Souza 2000, Schopf 2002, Reis 2005, Cirne et al 2007, Weifeng Sun et al, 2010].

O escalonamento de processos em *grid* computacional é uma área de pesquisa pelo desafio que a própria natureza desse sistema representa [Cirne 2002, Hong e Ni

2009, Ghanem at al 2010]. As características de *grid* que representam desafios no momento da atribuição das tarefas são:

- Grande quantidade de recursos: a grande quantidade de recursos torna-se um problema para o escalonador, que pode se tornar um gargalo do sistema, pois ele deve escolher de forma apropriada, qual recurso irá executar cada processo.
- Grande heterogeneidade de recursos: máquinas pertencentes ao *grid* podem apresentar configurações heterogêneas. Entre as configurações, as principais são: poder de processamento, interconexões e sistemas operacionais.
- Alto compartilhamento de recursos: a variação de carga nas máquinas causada pela submissão de novos processos ao sistema é proporcional ao número de usuários do *grid*, isto é, quanto mais usuários, maior será a variação de carga do sistema. Isso pode fazer com que políticas de escalonamento que não presumem tal fato atinjam um resultado negativo.
- Movimentação e consistência de dados: em *grid* deve-se evitar a submissão de aplicações que realizem muita comunicação, pois a baixa latência da rede de interconexão dos recursos pode causar prejuízos ao escalonamento.

2.1. Políticas de Escalonamento em *Grid* Computacional

Reis [Reis 2005] relata em seus estudos que uma das características das políticas de escalonamento é que elas devem ser focadas em um conjunto de aplicações específicas. Uma política deve conhecer os detalhes das aplicações as quais irá escalonar, pois a adoção de uma política genérica pode influenciar de maneira negativa nos resultados das execuções. Muitas vezes é mais vantajoso utilizar políticas de escalonamento simples em vez de uma política altamente eficaz que não atenda ao perfil do processo ao qual se deve supervisionar.

Um tipo de aplicação paralela utilizada em *grid* computacional são as aplicações BoT (*Bag-of-Task*). Essas aplicações são compostas por tarefas independentes, por isso, não é necessário qualquer tipo de comunicação entre as tarefas durante o processamento (a execução das tarefas não depende uma das outras). Desta forma, permite o uso de políticas baseadas em apenas alguns dados do sistema, raramente necessitando de informações sobre a infraestrutura do *grid*, como latência da rede e largura de banda existente entre os recursos [Silva 2003, Cirne et al, 2007, Ghanem et al 2010].

As características das aplicações BoT implicam em uma maior simplicidade para escalonar as tarefas, o que permite o uso de políticas tradicionais como *Workqueue* (WQ) e *Round-Robin* (RR) (utilizadas localmente em sistemas operacionais) a serem utilizadas para o escalonamento em *grid* computacional. Essas políticas, apesar de serem simples, representam uma grande base para o desenvolvimento de outras mais robustas e adaptadas com as características do ambiente e das aplicações do *grid* [Reis 2005].

Diversas políticas de escalonamento de tarefas, como *Workqueue with Replication* (WQR), *Sufferage*, *XSufferage*, *Dynamic FPLTF*, foram desenvolvidas para otimizar os problemas de escalonamento de tarefas.

O *Workqueue with Replication* (WQR) foi desenvolvido para solucionar o problema da obtenção de informações sobre a aplicação e a carga de utilização dos

recursos do *grid*. Em sua fase inicial o WQR é similar a um WQ, as tarefas são enviadas para execução nas máquinas que se encontram disponíveis. Quando uma máquina finaliza a execução de uma tarefa, esta recebe uma nova tarefa para processar. Os algoritmos WQR e WQ passam a diferir no momento em que uma máquina se torna disponível e não há mais nenhuma tarefa pendente para executar. Neste momento, o WQ já terminou seu trabalho e apenas aguarda a finalização de todas as tarefas. Porém, o WQR inicia sua fase de replicação para tarefas que ainda estão em execução, e assim que a tarefa original ou uma de suas réplicas finalizarem, as outras são interrompidas. Vale ressaltar que o WQR assume que as tarefas são idempotentes, isto é, não geram efeitos colaterais, como incrementação de valores, de modo a prevenir a inconsistência de dados que as réplicas poderiam gerar [Silva 2003].

A ideia básica do *Sufferage* [Maheswaran et al 1999] é determinar o quanto cada tarefa seria prejudicada se não fosse escalonada no processador que a executaria de forma mais eficiente. Portanto, o Sufferage prioriza as tarefas de acordo com o valor que mede o prejuízo de cada tarefa. O valor *sufferage* de cada tarefa é definido pela diferença entre o melhor e o segundo melhor CT (*Completion Time*), considerando todos os processadores do *grid*, onde CT = TBA + *Task Cost*. *Task Cost* = (*Task size* / *Host speed*) / (1 - *Host load*).

O *Task Size* é o tempo necessário para uma máquina com *Host Speed* = 1 completar a tarefa quando *Host Load* = 0. O *Host Load* representa a fração de CPU da máquina que não está disponível para a aplicação (fração de CPU que está sendo usada por outros usuários e aplicações). Deve-se lembrar que *Host Load* varia com o tempo, dependendo da carga que é imposta à máquina por outros usuários e aplicações. O *Host Speed* representa a velocidade relativa da máquina. TBA (*Time to Become Available*) é o tempo para o host se tornar disponível.

O *XSufferage* [Casanova et al 2000] é um algoritmo de escalonamento baseado nas informações sobre o desempenho dos recursos para associar tarefas aos processadores. Este algoritmo aborda o impacto das grandes transferências de dados em aplicações que usam grandes quantidades de dados. O *XSufferage* é uma extensão modificada do algoritmo *Sufferage*.

A principal diferença entre o *Sufferage* e *XSufferage* é o método usado para calcular o valor do *sufferage*. O algoritmo *XSufferage* leva em consideração a transferência dos dados de entrada da tarefa durante o cálculo dos tempos de execução. Isso implica no uso das informações usadas também pelo *Sufferage* mais a largura de banda disponível na rede que conecta os recursos, diferente do *Sufferage* que necessita somente das informações relacionadas à CPU e o tempo estimado de execução da tarefa. Um ponto a ser observado é que este algoritmo considera somente os recursos livres no momento em que vai escalonar uma tarefa, pois caso contrário sempre o recurso mais rápido e com melhor conexão de rede receberia todas as tarefas.

O *Dynamic* FPLTF é o resultado da modificação do FPLTF [Menascé 1995] feita por [Paranhos et al 2003]. O *Dynamic* FPLTF necessita de três tipos de informações para escalonar as tarefas devidamente:

1) *Task Size*: é o tempo necessário para uma máquina com *Host Speed* = 1 completar a tarefa quando *Host Load* = 0.

2) *Host Load*: representa a fração de CPU da máquina que não está disponível para a aplicação (fração de CPU que está sendo usada por outros usuários e aplicações). Deve-se lembrar que *Host Load* varia com o tempo, dependendo da carga que é imposta à máquina por outros usuários e aplicações.

3*) Host Speed*: representa a velocidade relativa da máquina. Uma máquina que tem *Host Speed* = 2 executa uma tarefa duas vezes mais rapidamente que uma máquina com *Host Speed* = 1

No início do algoritmo, o tempo para se tornar disponível TBA (*Time to Become Available*) de cada host é iniciado com 0 e as tarefas ordenadas por tamanho em ordem decrescente. Desta maneira, a maior tarefa é a primeira a ser alocada. Cada tarefa é alocada para o host que provê o menor tempo de execução CT (*Completion Time*) para ela, onde: CT = TBA + *Task Cost*. *Task Cost* = (*Task size* / *Host speed*) / (1 - *Host load*).

Quando uma tarefa é alocada para uma máquina, o valor do TBA correspondente a este host é incrementado com *Task Cost*. As tarefas são alocadas até que todas as máquinas do *grid* estejam em uso. Após isso, a execução da aplicação é iniciada. Quando uma tarefa é completada, todas as tarefas que não estão rodando são escalonadas novamente até que todas as máquinas fiquem em uso. Isso continua até que todas as tarefas sejam completadas.

3. LIBTS

A LIBTS foi desenvolvida para disponibilizar a implementação dos algoritmos de escalonamento de tarefas: WQ, WQR, *Sufferage*, *XSufferage* e *Dynamic* FPLTF, no SimGrid. O SimGrid é uma ferramenta de simulação de aplicações em ambientes distribuídos heterogêneos utilizada para o estudo dos algoritmos de escalonamento de tarefas em *grid* computacional, que não disponibiliza políticas internas de escalonamento de tarefas, além disso, a implementação dos algoritmos deve ser feita pelo próprio usuário.

A biblioteca LIBTS utiliza as funções do módulo MSG disponibilizadas pelo SimGrid e executa com aplicações do tipo *Master-Slave*. Em uma aplicação *Master-Slave* um computador é definido como *Master* (mestre) e os outros como *Slaves* (escravos). A função do *Master* é controlar o envio das tarefas para os *Slaves.*

O escalonamento de tarefas tem sido uma questão importante para melhorar a execução paralela em sistemas distribuídos. O modelo *Master-Slave* tem sido aplicado com sucesso em muitas aplicações paralelizadas em diferentes domínios de aplicações. Alguns pesquisadores e desenvolvedores de aplicações, como [Yuangiang et al 2008] têm se dedicado a explorar os métodos para melhorar o desempenho de uma aplicação *Master-Slave* em sistemas heterogêneos.

Na Figura 1 é apresentada uma visão geral do SimGrid e como a LIBTS foi adicionada para interagir com os módulos do SimGrid.

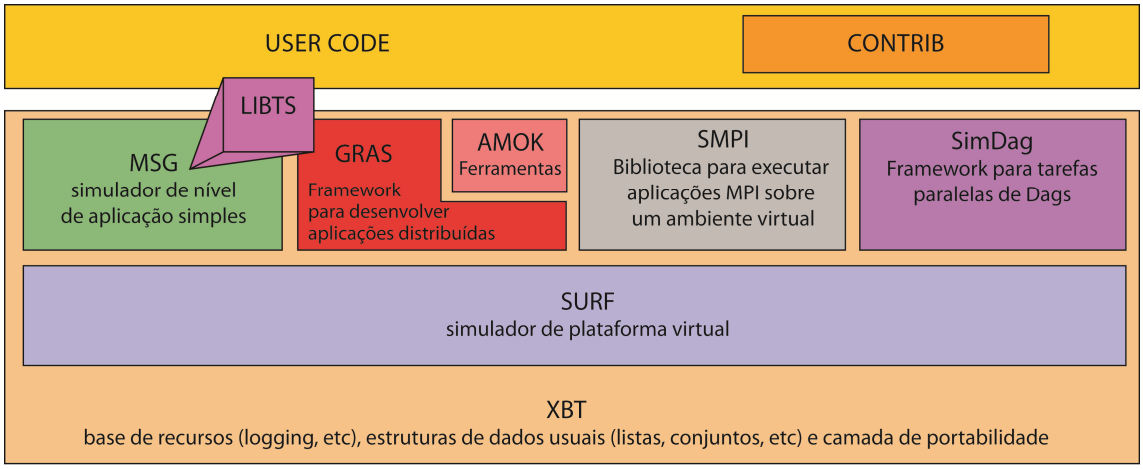

Figura 1. SimGrid com a Biblioteca LIBTS

A biblioteca LIBTS foi desenvolvida em linguagem C, que é a linguagem utilizada pelo SimGrid. Para o desenvolvimento da LIBTS, foi utilizado um notebook com 4GB de memória RAM, processador Intel Core 2 Duo 2.10GHz com Linux Ubuntu v.10.04 e a versão 3.5 do SimGrid. Para os testes das comparações entre os algoritmos implementados na biblioteca, foi utilizado o mesmo notebook do desenvolvimento da LIBTS e também um desktop com 8GB de memória RAM, processador Intel Core I7 2.93GHz com Linux Ubuntu v.10.10 e posteriormente com Linux Ubuntu v.11.04.

3.1. Arquitetura da LIBTS

A biblioteca é composta pelo módulo principal "Escalonamento", que contém os algoritmos: WQ, WQR, *Sufferage*, *XSufferage*, *Dynamic* FPLTF, como também os algoritmos básicos FIFO, LIFO, RR, SJF. Para os algoritmos funcionarem no SimGrid, foram feitas adaptações seguindo os padrões dos componentes do módulo MSG Nativo do Simgrid. Uma das alterações efetuadas foi a utilização da estrutura dos dados de acordo com os *MSG datatypes* do simulador. A arquitetura da biblioteca LIBTS é apresentada na Figura 3 (b).

O usuário que desejar desenvolver seu próprio exemplo pode utilizar a LIBTS, desde que crie a aplicação seguindo os padrões dos componentes do módulo MSG Nativo. Após a criação do exemplo, o usuário deve acrescentar em seu código a linha que chama a função "*escalonamento.c*" passando os parâmetros necessários. Exemplo:

escalonamento (todo, number_of_tasks, slaves, slaves_count);

Para descrever a plataforma e a descrição da aplicação, pode-se utilizar arquivos XML (eXtensible Markup Language) que devem conter informações como: o nome do nó, as funções que os nós devem exercer (*server* ou *slave*), o poder computacional do nó (em MFLOPS – *Mega Floating point Operations Per Second*), a largura de banda (em *Bytes*), a latência (em segundos) e a tabela de roteamento entre os nós [SimGrid 2011].

O exemplo de aplicação *Master-Slave* disponibilizado pelo SimGrid contém todas as informações (declaração das bibliotecas do SimGrid, dados das aplicações e plataforma, criação e organização das tarefas, entre outras) dentro do código-fonte "Masterslave_bypass.c", desta forma não é utilizado arquivos XML para descrever as aplicações e a plataforma, como também não tem opções de escalonamento das tarefas.

A aplicação *Master-Slave* contém as informações necessárias para o funcionamento da aplicação, com algumas diferenças, como por exemplo: para as aplicações e a plataforma são utilizados arquivos XML e o escalonamento de tarefas é efetuado pela LIBTS, onde o usuário pode escolher qual algoritmo será utilizado.

A Figura 2 apresenta um trecho do código-fonte do "Masterslave.c" com a declaração das bibliotecas do SimGrid, da LIBTS e das aplicações e plataforma.

```
#include <stdio.h>
#include "escalonamento.h"

XBT_LOG_NEW_DEFAULT_CATEGORY(msg_test,"Messages  specific  for  this  msg
example");
int flag_task[TCOUNT];
int atoi_task=9;
int s_flag;
int *p_flag_task=flag_task;

#include "masterslave.h"
{
   |

}
  /*  Simulation setting */
MSG_set_channel_number(MAX_CHANNEL);
MSG_create_environment ("plataform_test_2.xml");

/*   Application deployment */
MSG_function_register("master", master);
MSG_function_register("slave", slave);
MSG_launch_application("application_test_2.xml");
{
   |

}
INFO1("Got %d slave(s) :", slaves_count);

  for (i = 0; i < slaves_count; i++)
    INFO1("\t %s", slaves[i]->name);

  INFO1("Tem %d tarefas para processar:", number_of_tasks);

/* LIBTS */
  escalonamento (todo, number_of_tasks, slaves, slaves_count);

  INFO0("Goodbye now!");
  free(slaves);
  free(todo);
  return 0;
```

Figura 2. Trecho do código-fonte "Masterslave.c"

Na Figura 3 é apresentada a arquitetura de uma aplicação no SimGrid.

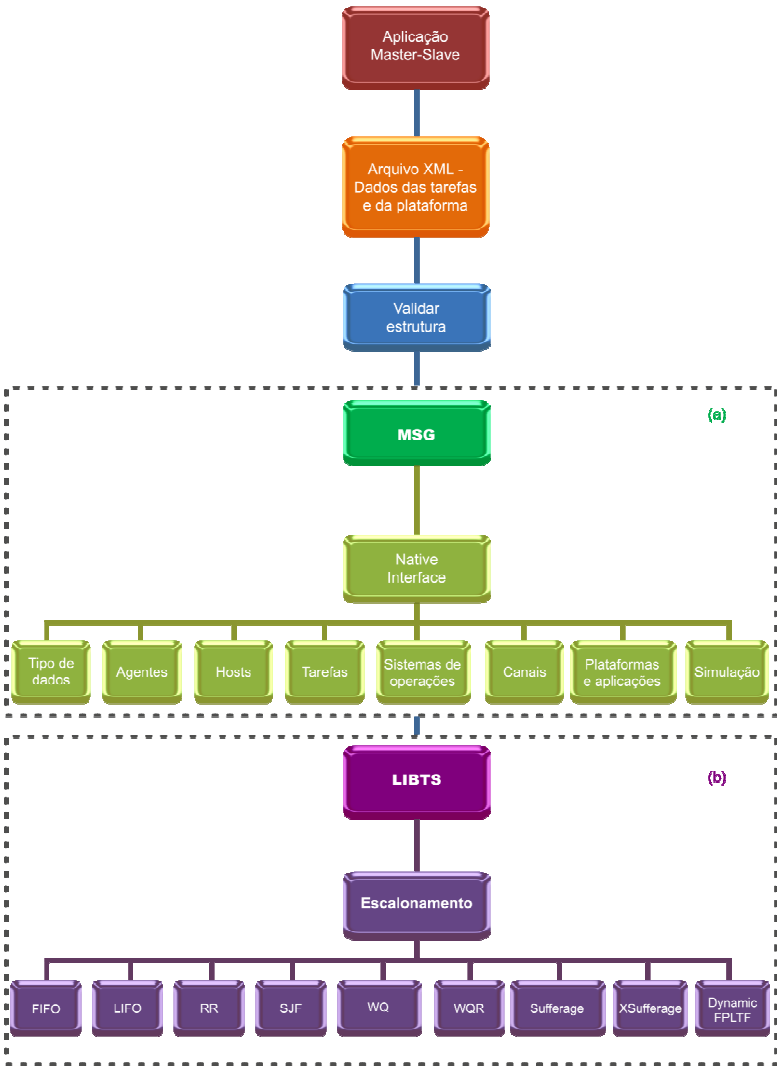

Figura 3. Arquitetura de uma Aplicação no SimGrid. a) Arquitetura do módulo MSG do SimGrid. (b) Arquitetura da LIBTS

Foram realizados experimentos em diferentes cenários com plataformas de 5 e 90 *hosts*, que são disponibilizados nos exemplos do módulo MSG do SimGrid. É complexo gerar manualmente um arquivo de plataforma, pois os arquivos de plataformas do SimGrid devem conter informações, como todos os *hosts* e *links* de roteamentos entre eles, utilizando todos os algoritmos de geração clássica do simulador [SimGrid 2011].

Um ponto a ser destacado nos experimentos realizados é que os algoritmos baseados em informações sobre o ambiente e as aplicações (*Sufferage*, *XSufferage* e *Dynamic* FPLTF) foram alimentados com as informações necessárias (por exemplo tamanho das tarefas, velocidade e carga das máquinas. Devido a complexidade de um ambiente de *grid*, nem sempre é possível obter essa situação em um ambiente real.

4. Conclusão

Este trabalho apresentou uma síntese sobre escalonamento de tarefas em *grid* computacional. O SimGrid é um ferramenta de simulação que oferece funcionalidades básicas para simulação em ambientes distribuídos, entretanto, não disponibiliza políticas internas de escalonamento de tarefas, a implementação dos algoritmos deve ser feita pelo próprio usuário. Toda vez que um usuário quer estudar sobre escalonamento de tarefas em *grid* computacional, como também comparar as políticas de escalonamento de tarefas existentes com novas políticas, necessitam implementá-las. Desta maneira, o usuário perde muito tempo, pois além de precisar fazer um estudo aprofundado das políticas de escalonamento de tarefas e do funcionamento do simulador, necessita fazer as implementações das políticas de escalonamento integrando ao simulador.

A biblioteca LIBTS foi desenvolvida e integrada ao simulador SimGrid para auxiliar no estudo dos algoritmos de escalonamento de tarefas em *grid* computacional, permitindo aos usuários dedicar-se ao estudo das políticas de escalonamento existentes através da comparação dos algoritmos implementados, da análise dos resultados, assim como a implementação de novos algoritmos.

Vários cenários foram utilizados para testar a LIBTS. O cenário de validação foi criado e executado na LIBTS e seus resultados foram confirmados pelo teste de mesa, validando assim o funcionamento da LIBTS. Outros testes de comparação dos algoritmos foram realizados, o desempenho dos tempos de simulação das aplicações foram apresentados em tabelas e gráficos. As análises estatísticas foram efetuadas utilizando a distribuição *t* de *Student* para estimar os valores médios e nível de confiança dos tempos de simulação.

Referências

Buyya, R., “Economic-based Distributed Resource Management and Scheduling for Grid Computing”. (2002) Thesis of Doctor. URL: http://www.buyya.com/thesis/

Casanova, H., Simgrid: “A Toolkit for the Simulation of Application Scheduling”. May (2001), 430-437. Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01).

Casanova, H. and et al., “Heuristics for Scheduling Parameter Sweep Applications in Grid environments”. (2000), In 9th Heterogeneous Computing Systems Workshop (HCW 2000).

Cho-Chin Lin, and Chun-Wei Shih., “An efficient scheduling algorithm for grid computing with periodical resource reallocation”.Computer and Information Technology. (2008), 295-300. CIT 2008. 8th IEEE International Conference. DOI: 10.1109/CIT.2008.4594690.

Cirne, W. Grids Computacionais: Arquiteturas, Tecnologias e Aplicações. Terceiro Workshop em Sistemas Computacionais de Alto Desempenho, 2002.

Cirne, W., et. al., “On the efficacy, efficiency and emergent behavior of task replication in large distributed systems”. Journal of Parallel Computing, v. 33 n. 3 (2007).

Foster, I., and C. Kesselman, and S. Tuecke, “The anatomy of the Grid: Enabling scalable virtual organizations”. Intl. J. Supercomputer Applications, (2001). URL: http://www.globus.org/alliance/publications/papers/anatomy.pdf

Ghanem, A.M.A.; Saleh, A.I.; Ali, H.A. High performance adaptive framework for scheduling Grid Workflow applications. Computer Engineering and Systems (ICCES), p. 52-57, 2010.

Hong Jiang; Tianwei Ni; PB-FCFS-a task scheduling algorithm based on FCFS and backfilling strategy for grid computing. Pervasive Computing (JCPC), p. 507-510, 2009.

Maheswaran, M., and et. al., “Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems”. (1999), URL: http://cisr.nps.edu/downloads/MSHN/dynamic_jpdc_special.pdf

Menascé, D., and et al., “Static and Dynamic Processor Scheduling Disciplines in Heterogeneous Parallel Architectures”. Journal of Parallel and Distributed Computing. (1995), 1-18.

Paranhos, D., and W. Cirne, and F.V. Brasileiro, “Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids”. (2003). URL: http://walfredo.dsc.ufcg.edu.br/resume.html#publications

Reis, V.Q. Escalonamento em grids computacionais: estudo de caso. (Mestrado). ICMC-USP, São Carlos-SP, 2005.

SimGrid. “SimGrid Project. Toolkit for simulation of distributed applications in heterogeneous distributed environments”. (2011). URL: http://simgrid.gforge.inria.fr/

Schopf, J.M. A General Architecture for Scheduling on the Grid. Special Issue on Grid Computing, J. Parallel and Distributed Computing, April 2002.

Silva, D.P. Usando Replicações para Escalonar Tarefas Bag-of-Tasks em Grids Computacionais. (Mestrado), Universidade Federal de Campina Grande (UFCG), 2003.

Souza, P.S.L. AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos. (Doutorado). ICMC-USP, São Carlos, Brasil. 2000.

Weifemg Sun; et al. A Priority-Based Task Scheduling Algorithm in Grid. Parallel Architectures, Algorithms and Programming (PAAP), p.311-315, Dec. 2010.

Yu Liang and Zhou Jiliu, “The Improvement of a Task Scheduling Algorithm in Grid Computing”. Data, Privacy, and E-Commerce, (2007), 292–297. ISDPE 2007. The First International Symposium on Digital Object Identifier: 10.1109/ISDPE.2007.17.

Yuanqiang Huang; et al. EOMT: A Master-Slave Task Scheduling Strategy for Grid Environment. High Performance Computing and Communications (HPCC'08), pp.226-233, Sept. 2008.