Aspectos de Utilização do SimGrid para Aplicações de PAD

Francisco José da Silva Borges de Santana[1]

[1]Departamento de Ensino – Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA) – Santo Amaro – BA - Brasil

franciscoborges@ifba.edu.br

Abstract. *The experiments in real environments for high performance applications are often limited to certain scenarios. When a solution for high performance computing is proposed it is necessary to do experiments in different contexts. However, it is not always possible to do validation using multiple parameters and scenarios for logistics. The SimGrid framework allows by simulating the validation of research in high performance computing in various scenarios, settings and parameters. With SimGrid is possible to model applications and architecture of high performance computing and simulation by studying the various challenges in the area.*

Resumo. *As experimentações em ambientes reais de aplicações de alto desempenho são muitas vezes limitadas a determinados cenários. Quando uma solução para computação de alto desempenho é proposta faz-se necessário realizar a experimentação em diversos contextos. No entanto, nem sempre é possível fazer a validação utilizando múltiplos parâmetros e cenários por questões logísticas. O framework SimGrid permite, por meio da simulação, a validação de pesquisas em computação de alto desempenho em diversos cenários, configurações e parâmetros. Com o SimGrid é possível modelar aplicações e arquitetura de computação de alto desempenho e através da simulação estudar os diversos desafios na área.*

1. Introdução

A comunidade científica, atualmente, dispõe de quatrilhões de cálculos por segundo (petaflops) para o desenvolvimento de pesquisas e experimentos. São centenas de milhares de núcleos disponíveis para a execução dos mais complexos problemas. A cada lista do TOP500 [TOP500 2011] surgem sempre novos computadores que ocupam a posição de outras super máquinas, seja por upgrade ou por melhoria na tecnologia. Segundo Casanova, Legrand e Quinson (2008), muitos desafios em aplicações distribuídas ainda permanecem, tais como: obter a melhor performance, melhor escalabilidade, melhor tolerância a falhas; e que, constantemente, novos desafios estão surgindo graças à evolução da tecnologia e das aplicações.

Para um bom aproveitamento dessas tecnologias, inevitavelmente, deve existir o desenvolvimento de técnicas e estratégias de software. Como distribuir uma aplicação de forma que nenhum processador fique sobrecarregado ou ocioso? Como minimizar a comunicação entre os *hosts*? Como trabalhar de forma eficiente com tantos processadores/núcleos? Como deve ser a distribuição de dados quando um *host* falhar? Essas são algumas das razões de se estudar estratégias de *scheduling*, distribuição e mapeamento dos dados e/ou comunicação em aplicações distribuídas e paralelas. Essas são questões recorrentes e ainda desafiadoras.

Em Processamento de Alto Desempenho - PAD, ou na área de computação de uma forma geral, é possível fazer pesquisas experimentais reais ou simuladas. A experimentação real nos remete a dois principais problemas: (i) quanto o experimento é abrangente e (ii) à possibilidade de reprodução dos resultados por outros pesquisadores. O pesquisador, quando desenvolver um experimento, nem sempre tem disponíveis recursos computacionais suficientes para tornar a experimentação o mais abrangente possível. Imagine que um algoritmo de *scheduling* foi proposto. Para comprovar a eficiência desse algoritmo seria necessário aplicá-los a um número crescente de *hosts*. Não é uma tarefa fácil conseguir uma quantidade suficiente de *host* para que a validação de seus dados seja confiável. O outro problema passa pela questão metodológica da pesquisa. Os experimentos devem poder ser reproduzidos, ou seja, o experimento deve fornecer condições para que outros pesquisadores consigam obter os mesmos resultados. Em uma experimentação real de PAD, quando existe uma infraestrutura complexa dando suporte, essa tarefa pode ser um tanto difícil. Na experimentação real existem variáveis e eventos externos que não podem ser controlados.

A simulação não precisa que um sistema real seja construído, permite que os experimentos sejam controlados, repetidos e aplicados a diversos cenários e parâmetros, além de permitir a reprodução dos resultados por outros pesquisadores. A simulação permite predizer o comportamento do algoritmo usando um modelo matemático aproximado. Neste contexto, o SimGrid é um framework de simulação de grades e aplicações distribuídas, amplamente utilizado e aceito pela comunidade científica e acadêmica, que permite o estudo de heurísticas, estratégias e algoritmos para soluções em PAD.

O objetivo desse texto é apresentar o SimGrid e mostrar como ele poder ser utilizado para o estudo de PAD. Esse texto está organizado da seguinte forma: a seção 2 apresenta os conceitos de simulação e mostra a importância de utilizar esse recurso na experimentação

científica; a seção 3 fala sobre o SimGrid, apresenta outros simuladores e mostra através de seu histórico a evolução da ferramenta; a seção 4 detalha a atual arquitetura do SimGrid; a seção 5, através de exemplo, mostra como é fácil fazer uma simulação no framework; a seção 6 fala sobre a visualização das simulações; a seção 7 mostra os aspectos de experimentação de PAD usando o SimGrid; e, por fim, a seção 8 as considerações finais.

2. Simulação

Segundo Quinson (2010), as abordagens clássicas usadas em ciências e engenharia para o estudo de um problema científico são teórica e experimental. Na primeira abordagem, o foco é no desenvolvimento de equações e teoremas. Na experimental, o pesquisador desenvolve seus estudos através de instrumentos científicos, em alguns casos protótipo. No entanto, essas abordagens nem sempre são desejáveis ou mesmo possíveis. Alguns fenômenos são intratáveis teoricamente ou são impossíveis de efetuar experimentos, como por exemplo, comportamento dos buracos negros, mudanças climáticas, mapeamento genético. Em outros casos, desenvolver o experimento é muito caro, difícil de ser feito e a apresentação de resultados pode demorar.

Atualmente, o outro modo de fazer pesquisa é através da Computação Científica, pela modelagem e simulação de seus experimentos em computadores [Quinson 2010]. Antes da era da computação, os cientistas e engenheiros acreditavam que efetuar centenas de cálculos aritméticos em um segundo estava próximo do imaginável. De qualquer modo, assim que isso foi possível, eles já desejavam milhões, bilhões, trilhões de operações por segundo, ou seja, mais poder computacional [Pacheco 2006]. Quatrilhões de operações por segundo (petaflops), hoje, são uma realidade graças ao constante desenvolvimento das arquiteturas e tecnologias. O TOP500 (2011) apresenta os computadores mais potentes do mundo, de seis em seis meses. Para ilustrar a evolução tomemos como exemplo o *K Computer* o número um da lista de Junho de 2011. O supercomputador japonês é mais potente do que os outros cinco computadores mais potentes juntos.

Através dos supercomputadores, *grid computing* e cluster, a computação científica tornou-se uma ferramenta essencial para o desenvolvimento da ciência moderna. Hoje, grandes desafios como viagens espaciais, dinâmica química e molecular, dinâmica de fluidos, exploração petrolífera, por exemplo, podem ser estudados mais detalhadamente. Aliado ao desenvolvimento das máquinas há o desenvolvimento e aprimoramento das técnicas, softwares e ferramentas de PAD que precisam tirar o melhor proveito da capacidade computacional disponível nos *hardware*. Nesse contexto, faz-se necessário também o desenvolvimento de pesquisas nesse campo [Casanova, Legrand e Quinson 2008].

Segundo Quinson (2011) a experimentação em PAD pode ocorrer através de três abordagens: real, simulado ou emulado. A emulação consiste em executar uma aplicação em um ambiente específico e interceptar as chamadas do sistema fazendo a mediação, sem que seja necessário alterar a aplicação [Casanova, Legrand e Quinson 2008].

Experimento real é aquele que se implementa uma solução para um determinado objeto de estudo em um ambiente real. Como exemplo, apresentamos Santana (2007) que propõem um modelo de balanceamento de carga para cluster heterogêneo. Para validar o modelo, o autor desenvolveu uma aplicação que multiplica matrizes de diversos tamanhos

sobre o modelo proposto. O resultado da execução é avaliado ao final para avaliações e análises. O ponto positivo da experimentação real é que demonstra a aplicabilidade da estratégia proposta [Quinson 2011]. Para esse tipo de experimentação, a comunidade cientifica disponibiliza diversos projetos que permitem o desenvolvimento de aplicações e estudos em PAD onde podem ser executados em um ambiente real, tais como o Grid'5000 [Bolze at. al 2006] e PlanetLab [Chun at al 2003].

No entanto, alguns pontos negativos devem ser destacados na abordagem real: (i) investimento de muito tempo e trabalho; (ii) dificuldade em escolher os cenários para validar o trabalho científico; (iii) limitação nos resultados aos cenários onde o experimento foi executado; (iv) impossibilidade de controlar o ambiente; (v) impossibilidades de reprodução por outros pesquisadores, que é essencial na metodologia científica, dos experimentos.

A simulação resolve alguns dos problemas encontrados na experimentação real. Na simulação: (i) não é necessário desenvolver uma aplicação real; (ii) existe a possibilidade de controlar o ambiente e de repetir os experimentos; (iii) os cenários para testes não são limitados; e (iv) há possibilidade do experimento ser repetido por outros pesquisadores [Quinson 2011]. A tradução livre de simulação é a tentativa de predizer o comportamento de algum sistema através da criação de um modelo matemático aproximado [Howe 2011].

Pacheco (2006) apresenta um problema que podemos extrair algumas vantagens de se fazer simulação: deseja-se saber se vai chover nos Estados Unidos e no Canadá nos próximos dois dias e em cada hora. Suponha, também, que se deseja modelar a atmosfera do nível do mar até 20km de altura. Um abordagem comum para resolver esse tipo de problema é cobrir toda a região de interesse com uma "grade" e predizer se vai chover em cada vértice da grade. Algumas questões técnicas surgem para a solução desse problema, tais como: (i) como será a distribuição dos dados entres os processadores? Mandaremos a mesma carga para todos os nós? (ii) como será a comunicação entre os *hosts*? (iii) como será feito o mapeamento dos dados? (iv) se um *host* falhar como será a recuperação dos dados? Essas são apenas algumas questões que podem surgir com um problema dessa natureza. As soluções técnicas para essa questões existem, mas qual é a mais eficiente para o problema citado? Às vezes, o problema possui uma determinada particularidade que uma técnica seja menos indicada que outra. No entanto, esse comportamento somente é observado quando a aplicação já foi toda implementada. Com a simulação, essas questões podem facilmente serem avaliadas com um esforço bem menor e com menor custo. As simulações permitem que diversas configurações sejam testadas de modo a permitir que ajustes nas estratégias sejam efetuadas com o objetivo de garantir a eficiência do método aplicado. É mais fácil, rápido e eficiente, pensar em uma estratégia e fazer a simulação antes de implementá-la. Com a simulação é possível implementar a estratégia, testar, debugar para depois você finalmente implementar o código.

Fazer experimentações por simulação de PAD é uma opção bastante interessante visto que as possibilidades de testes podem ser extrapoladas com a utilização de diversos parâmetros e cenários. No SimGrid a simulação pode ser desenvolvida apenas com o propósito de estudo de uma solução ou problema. Mas a simulação pode ser convertida em uma aplicação real, como apresentaremos na próxima seção.

3. SimGrid

O SimGrid é um framework de simulação para aplicações distribuídas [Casanova, Legrand e Quinson 2008]. Ele pode ser usado para pesquisas em *cluster*, *grid*, algoritmo P2P, computação voluntária e estratégias e heurísticas para algoritmos paralelos. Conforme Casanova, Legrand e Quinson (2008) as principais características do SimGrid são: (i) o *kernel* da simulação é escalável, extensível e implementa alguns modelos simulados, o que permite a simulação de inúmeras topologias de rede, computação dinâmica, recursos de rede disponíveis, bem como suas falhas; (ii) API que possibilita, a pesquisadores em computação distribuída, o desenvolvimento rápido de simulações; (iii) APIs para o desenvolvimento de aplicações distribuídas que podem ser executadas em ambiente simulado ou no real.

Na literatura existem diversos simuladores para o desenvolvimento de computação distribuída, como pode ser observador na figura 1. A escalabilidade do SimGrid, na versão atual – 3.6.1, já ultrapassou esses valores. Em relação às outras ferramentas, o SimGrid se destaca na escalabilidade, na possibilidade de controlar as configurações da simulação e no modelo dos recursos simulados.

	CPU	Disk	Network	Application	Requirement	Settings	Scale
Grid'5000 [4]	direct	direct	direct	direct	access	fixed	< 5000
PlanetLab [8]	virtualize	virtualize	virtualize	virtualize	none	uncontrolled	< 850
ModelNet [20]	-	-	emulation	emulation	lot material	controlled	100 nodes/real host
MicroGrid [21]	emulation	-	fine d.e.s.	emulation	none	controlled	few 100
ns2 [1]	-	-	fine d.e.s.	coarse d.e.s.	C++ and Tcl	controlled	<1,000 [18]
SSFNet [9]	-	-	fine d.e.s.	coarse d.e.s.	Java	controlled	<100,000 [18]
GTNetS [18]	-	-	fine d.e.s.	coarse d.e.s.	C++	controlled	<177,000 [18]
ChicSim [17]	coarse d.e.s.	-	fine d.e.s.	coarse d.e.s.	C	controlled	thousands
OptorSim [2]	coarse d.e.s.	amount	math/d.e.s.	coarse d.e.s.	Java	controlled	few 100
GridSim [5]	coarse d.e.s.	fine d.e.s.	fine d.e.s.	coarse d.e.s.	Java	controlled	few 100
PlanetSim [11]	-	-	constant time	coarse d.e.s.	Java	controlled	100,000 [15]
PeerSim [12]	-	-	-	state machine	Java	controlled	1,000,000 [15]
SimGrid	coarse d.e.s.	-	math/d.e.s.	d.e.s./emul	C or Java	controlled	few 10000

Figura 1. Ferramentas de experimentação para aplicações distribuídas. Fonte: [Casanova, Legrand e Quinson 2008].

Na literatura existe a publicação de mais de 70 artigos sobre o SimGrid [Quinson 2010]. Entre os artigos que usam o SimGrid em suas experimentações podemos citar como exemplo: Santos-Neto at al (2004) apresentam uma heurística para aplicações do tipo *Bag of Task* (BoT) que manipulam grandes volumes de dados e compara a performance com outras heurísticas já conhecidas na literatura; Caniou, Charrier e Desprez (2011) simulam a realocação de tarefas entre *clusters;* Cornea e Bourgeois (2011) conseguem predizer o tempo de execução de algoritmos paralelos através da capacidade do SimGrid de executar simulações a partir de arquivos de *trace* da rede; Hunold (2010) utiliza o SimGrid para comparar e avaliar o algoritmo de *scheduling* proposto; Munck, Vanmechelen e Broeckhove (2009) utilizam um roteamento dinâmico para melhorar a escalabilidade do SimGrid; Senger, Silva, Miranda Filho (2010) estudam a escalabilidade de aplicações do tipo BoT em plataforma *master/slave*. Esses trabalham ilustram a abrangência do framework e

possibilidades de experimentações que podem ocorrer. Uma lista mais ampla de publicações relacionadas ao SimGrid pode ser obtida em SimGrid(2011).

Os autores do SimGrid são Henri Casanova, da Universidade do Havaí; Arnaud Legrand, do Laboratório LIG; e Martin Quinson, da Universidade de Nancy; além de possuir diversos colaboradores. A primeira versão do SimGrid surgiu em 1999 quando Casanova tornou-se membro do grupo de pesquisa AppleS do Departamento de Ciência da Computação e Engenharia da Universidade da Califórnia. A principal linha de interesse desse grupo eram algoritmos de *scheduling* para aplicações científicas em ambientes distribuídos e heterogêneos. No desenvolvimento dos projetos de pesquisa, Casanova sentiu a necessidade de executar seus experimentos através de simulação ao invés de um ambiente real. Nesse momento Legrand e Casanova desenvolveram um simulador *ad-hoc*. Casanova observou que os pesquisadores do AppleS eventualmente precisavam executar simulações, implicando dessa forma que parte de seus códigos desenvolvidos para ambiente real fossem reescritos. Casanova, então, fez algumas alterações no simulador desenvolvido em conjunto com Legrand: (i) tornou o simulador mais genérico; e (ii) disponibilizou uma API simples. Essa versão foi chamada de SimGrid v1.0 (SG) e é descrita em Casanova (2001).

Em 2001, Legrand inicia a sua tese de doutorado. Seu trabalho tinha como objeto de investigação as heurísticas de *scheduling* descentralizada. A versão 1.0 do SimGrid dava esse suporte, no entanto usar essa versão seria muito complicado além de possuir um escopo muito restrito. Então, Legrand adicionou no topo do SG - figura 2 - uma camada denominada de MSG (Meta-SimGrid) que é apresentada em Legrand e Lerouge (2002). Foi incluído nessa camada o suporte a *threads*, permitindo a execução de processos de computo e comunicação independentemente viabilizando, dessa forma, o modo assíncrono.

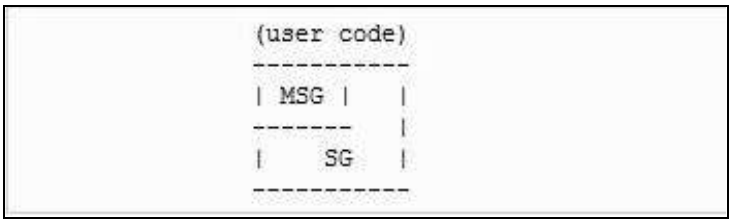

Figura 2. Componentes do SimGrid v2.0. Fonte: [SimGrid 2011].

A principal limitação do SG era a impossibilidade de simular com realismo a comunicação de rede *multi-hop*. Em 2003, Loris Marchal sob orientação de Casanova implementa um modelo de rede para o SG. Este modelo incrementou bastante o nível de realismo das simulações no SimGrid e foi descrito em Casanova e Marchal (2002). O trabalho de Marchal e Casanova foram incorporados ao SimGrid, tornando-se em versão 2.0 [Legrand, Marchal e Casanova 2003].

O SimGrid v2.0 possuía diversos recursos que permitiu aumentar a base de usuário, pesquisas e colaboradores. Um desses foi Martin Quinson que percebeu que os códigos das simulações precisavam ser reescrito quando levados para o ambiente de produção. Quinson trabalhou no GRAS, cujo objetivo seria compilar o mesmo código da simulação na experimentação real. O GRAS executa sobre a camada MSG para a simulação e na camada

sockets para a experimentação real. O GRAS foi descrito em Quinson (2006) e aborda a arquitetura apresentada na figura 3.

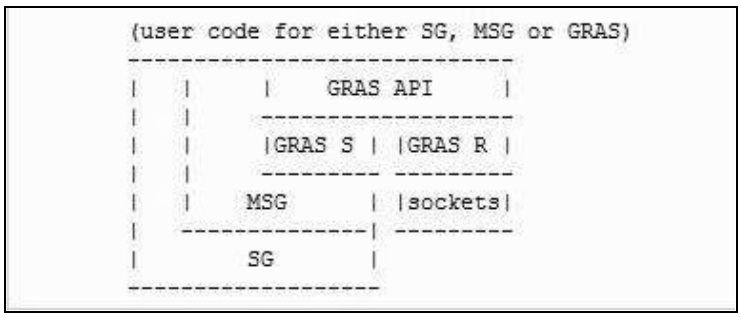

Figura 3. Inclusão do GRAS no SimGrid v2.0. Fonte: [SimGrid 2011].

Naquele momento, o SG do SimGrid v1.0 estava limitado à escalabilidade e à performance e não atendia as simulações mais complexas que estavam sendo desenvolvidas. Por isso, em 2005, Legrand substituiu o SG pelo *kernel* de simulação chamado SURF. O SURF ficou mais extensível e viabilizou a evolução dos modelos de simulação no SimGrid. Houve o registro pelos usuários de um incremento na performance. Quinson adicionou no topo do GRAS o AMOK, camada que provia serviços necessários a aplicações distribuídas. Com essas alterações surge o SimGrid v3.0 descrito na figura 4.

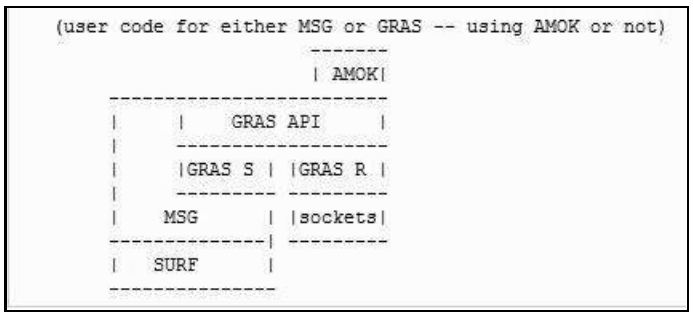

Figura 4. Troca do *kernel* de simulação SG pelo SURF e inclusão do AMOK Fonte: [SimGrid 2011].

Christophe Thiery, durante um estágio com Martin Quinson, desenvolveu o SimDAG. O SimDAG prover uma interface mais simples que a MSG para o estudo de *scheduling* centralizado similar a API SG disponível na versão 1.0 e 2.0 do SimGrid. Essa camada foi incorporada a versão 3.1 do SimGrid representada na figura 5.

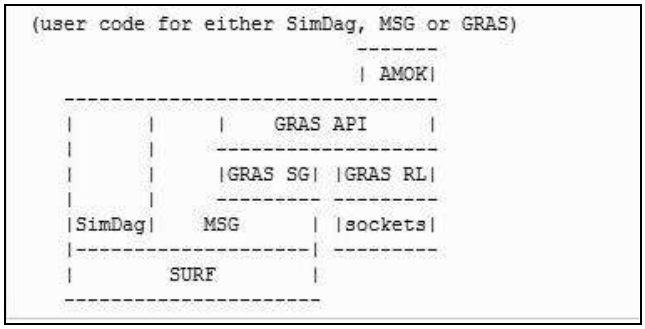

Figura 5. Componentes do SimGrid v3.1. Fonte: [SimGrid 2011].

A partir da versão 3.2 o SimGrid começou a prover suporte parcial ao sistema operacional Windows.

Durante o estágio com Legrand, Bruno Donassolo desenvolveu uma camada intermediária, denominada de SIMiX, entre o *kernel* do simulador, o SURF, e as API de mais alto nível . Isso implicou na reimplementação do MSG e do GRAS e corrigiu uma falha de projeto da versão anterior, onde o GRAS executava sobre o MSG. O SIMiX, além de melhorar o projeto do SimGrid, facilitou o desenvolvimento de outras camadas como o SMPI. O SMPI foi desenvolvido por Mark Stillwell, que trabalha com Casanova. O SMPI permite a execução do mesmo código MPI, sem alteração, no modo de simulação ou no ambiente real. Malek Cherier, junto com Quinson, disponibilizou também uma versão Java para a MSG - jMSG. A figura 6 mostra as alterações efetuadas.

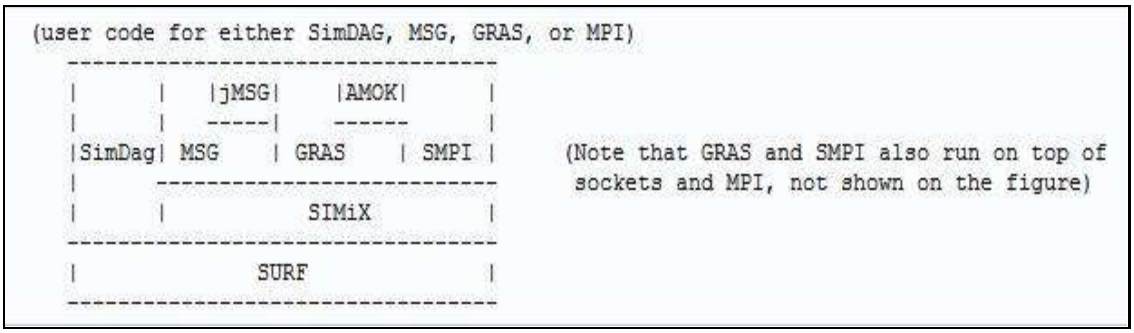

Figura 6. Inclusão do SIMiX, SMPI e jMSG no SimGrid. Fonte: [SimGrid 2011].

Diversos trabalhos de pesquisa do SimGrid estão relacionados aos modelos de simulação que estão implementados no SURF. Dessa forma, Legrand fez a refatoração do SURF para deixá-lo mais extensível e consequentemente poder executar a simulação com novos modelos desenvolvidos. Kayo Fujiwara com Casanova integrou o SURF com o simulador GTNetS – figura 7.

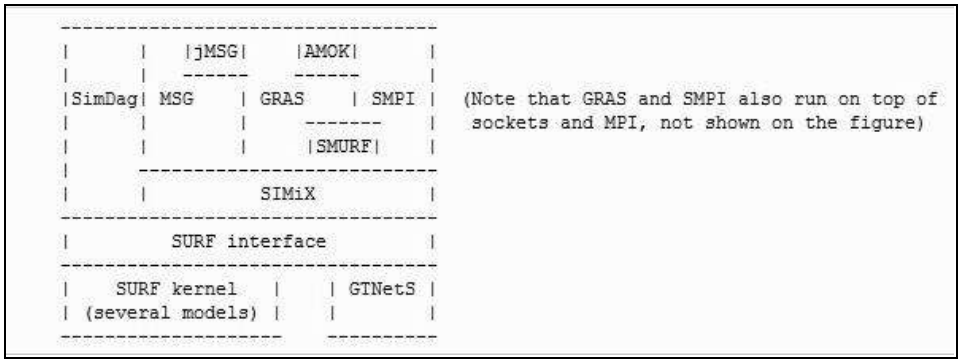

Figura 7. Refatoração do SURF e integração com o GTNetS. Fonte: [SimGrid 2011].

O projeto do SimGrid é constantemente atualizado e a base de usuários aumenta progressivamente. Com isso a demanda por resolução de *bugs* e novas características aumentam. Isso corrobora com que a documentação on-line da ferramenta não consiga seguir o mesmo ritmo de atualização. A exemplo, o histórico das atualizações disponível em SimGrid (2011) somente está disponível até a versão 3.3, quando a versão atual disponibilizada no

repositório é a 3.6.1. O modelo de componentes do SimGrid mais atual documentado é o disponível em Casanova, Legrand e Quinson (2008) apresentado na seção seguinte.

4. Arquitetura do SimGrid

A arquitetura de componentes mais atual documentada do SimGrid é apresentada na figura 8. O SimGrid possui quatro APIs [Casanova, Legrand e Quinson 2008]:

- SimDAG é herança do SimGrid v1 e é projetado para trabalhar com heurísticas de *scheduling* para aplicações do tipo *task graphs*;
- MSG permite o estudo de aplicações do tipo *concurrent sequential processes* (CSP);
- GRAS permite desenvolver aplicações distribuídas reais;
- SMPI permite simular código MPI diretamente;

Os módulos mais internos são [Casanova, Legrand e Quinson 2008]:

- XBT é um módulo *toolbox* utilizado em todo o software. Ele disponibiliza funções que implementam recursos como log , tratamento de erro, suporte a configuração, portabilidade e estruturas de dados;
- SURF é o módulo onde está implementado o *kernel* da simulação;
- SIMiX é um módulo ponto de extensão do SimGrid, pois prove uma interface *POSIX-like* que facilita o desenvolvimento de APIs para simulação, como por exemplo *openMP*;
- SMURF permite a distribuição dos processos da simulação por um *cluster*, antes desse módulo as simulações ficam limitadas a memória local;

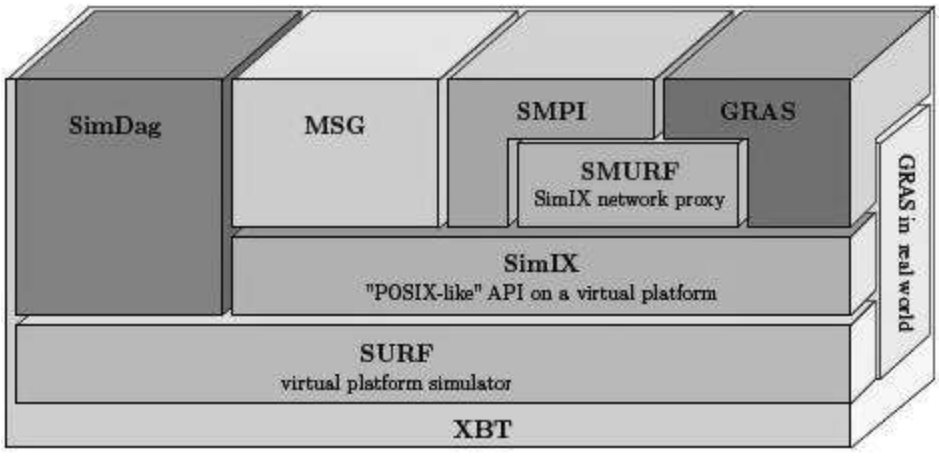

Figura 8. Componentes do SimGrid. Fonte: [Casanova, Legrand e Quinson 2008].

Os principais conceitos do SimGrid são [Quinson 2006], [Legrand, Marchal e Casanova 2003]:

- *Agents* – são processos individuais que fazem parte da aplicação distribuída;

- *Sockets/ Channel* – é o canal de comunicação entre os *Agents*;
- *Messages* - representa as informações que são trocadas entre os *Agents*;
- *Callbacks* – são funções definidas por usuário que são executas automaticamente de acordo com o tipo de mensagem recebida;
- *Link* – prove a abstração para o roteamento entre *host*.

Casanova, Legrand e Quinson (2008) categorizam as API de alto nível – SimDAG, MSG, GRAS e SMPI - em duas: para pesquisadores e para desenvolvedores. A razão para isso é que o SimDAG e MSG estão relacionadas com os estudos de estratégias e heurísticas para problemas de *scheduling*. O GRAS e o SMPI permitem que aplicações reais sejam executadas no modo de simulação e no ambiente real sem alteração do código fonte.

Na computação distribuída e paralela um problema recorrente é o de *scheduling*. Segundo Casanova e Marchal (2002), o objetivo do *scheduling* é encontrar a melhor solução que atribua um conjunto de tarefas a determinado conjunto de recursos disponíveis, de forma que o tempo de execução total da aplicação seja menor. A complexidade dos algoritmos de *scheduling* aumenta quando as tarefas possuem dependência de execução. As aplicações desse tipo muitas vezes podem ser descritas como *Direct Acyclic Graph* (DAG) ou *task graphs*. Nesse contexto, o SimDAG permite desenvolvimento algoritmos de *scheduling* aplicando heurísticas com o objetivo de aproximar da solução otimizada. Dessa forma, diversas heurísticas para *scheduling* podem ser simuladas e comparadas.

O MSG foi desenvolvido para modelar processos sequenciais concorrentes, modelar problemas teóricos e comparar heurísticas. Inicialmente essa API foi desenvolvida para o estudo de *scheduling*, no entanto tornou-se aplicável em outros contextos, como por exemplo, para computação voluntária. Na versão 3.3 foi incluída a versão jMSG, que permite o uso da API em Java [Casanova, Legrand e Quinson 2008].

Para o desenvolvimento de aplicações de *Grid* e P2P, o SimGrid disponibiliza o GRAS - *Grid Reality And Simulation*. A simulação desenvolvida nesse módulo pode facilmente ser disponibilizada no ambiente real. Basta o desenvolvedor fazer a compilação com a uma das bibliotecas implementadas – simulada ou real [Casanova, Legrand e Quinson 2008]. O GRAS foi projetado para construir infraestrutura distribuída, através de serviços específicos, para aplicações distribuídas e *middlewares* [Quinson 2006]. O GRAS baseia-se no paradigma *active message* e as aplicações oriundas dele seguem o modelo *event-driven.*

O módulo SMPI permite que qualquer código MPI seja executado em modo de simulação sem nenhuma alteração. O SimGrid consegue fazer isso interceptando a primitivas MPI.

O *kernel* do SimGrid está implementado no SURF. Nesse módulo está presente o código responsável pela simulação e foi desenvolvido com dois objetivos principais: ser completamente modular para permitir o desenvolvimento de outros modelos de recursos, facilitando dessa forma a extensão do SimGrid; e ser o mais otimizado possível para que a performance da simulação não seja prejudicada [Casanova, Legrand e Quinson 2008].

A simulação possui dois elementos principais: (i) um conjunto de recursos – link, CPU, por exemplo; e (ii) uma aplicação que usa aqueles recursos para executar determinadas ações, transferir arquivo e executar um cálculo são alguns exemplos. O papel do SURF é calcular o tempo gasto para executar as ações. Para isso o *kernel* solicita a cada modelo, que controla os recursos associados, quando a ação será finalizada; calculando, assim, o tempo mínimo de término daquelas ações. O SURF atualiza com esse valor o tempo da simulação e o estado das ações e recursos, além de informar ao usuário as ações já finalizadas. Nesse momento, o código de simulação escrito pelo usuário tem a oportunidade de iniciar a execução de nova ação – comunicação ou computo, isso faz com que o SURF inicie outro ciclo de simulação[Casanova, Legrand e Quinson 2008]. Esse *loop* é executado até que todas as ações tenham sido finalizadas.

A precisão da simulação é abordada em Casanova, Legrand e Quinson (2008), que apresentam os percentuais de erros e em quais modelos e situações. Já os cálculos efetuados no SURF estão descritos nos trabalhos em Casanova e Marchal (2002) e Quinson (2010).

5. Construindo uma simulação com o SimGrid

Para construir um simulação no SimGrid basta seguir os seguintes passos [Legrand, Marchal e Casanova 2003]:

- Modelar a aplicação, definindo o código de cada agente;
- Modelar a plataforma física definindo os recursos. Isso consiste em definir os *hosts*, *links* e a topologia;
- Especificar o arquivo de *deployment* da aplicação, onde é especificado a localização da criação e alocação dos agentes;
- Executar a simulação.

Para demonstrar como uma simulação é desenvolvida no SimGrid, detalharemos um exemplo de aplicação *master/slave* disponibilizada em SimGrid(2011). Este exemplo é implementado utilizando a API MSG.

Para utilizar o SimGrid é necessário algumas declarações preliminares. Alguns *includes* são obrigatórios para ter acesso a determinadas funções. Destaque para o *msg.h* que disponibiliza as funções da API MSG e as funções do XBT, que irão permitir gerar as saídas para análise da execução da simulação. É declarado também o protótipo de quatro funções: *master*, *slave, forwarder* e *test_all* . As três primeiras irão assumir o papel corresponde aos seus respectivos nomes. A *test_all,* nesse exemplo, é o núcleo da simulação. A figura 9 apresenta essas declarações preliminares.

```
#include <stdio.h>
#include "msg/msg.h"            /* Yeah! If you want to use msg, you need to include msg/msg.h */
#include "xbt/sysdep.h"         /* calloc, printf */

/* Create a log channel to have nice outputs. */
#include "xbt/log.h"
#include "xbt/asserts.h"
XBT_LOG_NEW_DEFAULT_CATEGORY(msg_test,
                             "Messages specific for this msg example");

int master(int argc, char *argv[]);
int slave(int argc, char *argv[]);
int forwarder(int argc, char *argv[]);
MSG_error_t test_all(const char *platform_file,
                     const char *application_file);

typedef enum {
  PORT_22 = 0,
  MAX_CHANNEL
} channel_t;
```

Figura 9. Declarações preliminares. Fonte: [SimGrid 2011].

O código do *master* é apresentado nas figuras 10 e 11. Na figura 10, observa-se a declaração de algumas variáveis, a atribuição a essas variáveis dos parâmetros que serão informados via linha de comando e a criação das tarefas que serão simuladas. Essa parte do código é responsável por criar as tarefas. Duas funções importantes utilizadas nesse código são a *xbt_new0* e *MSG_task_create*. A primeira aloca memória para armazenar as tarefas que serão criadas. A segunda cria as tarefas que serão simuladas.

```
#define FINALIZE ((void*)221297)        /* a magic number to tell people to stop working */

int master(int argc, char *argv[])
{
  int slaves_count = 0;
  m_host_t *slaves = NULL;
  m_task_t *todo = NULL;
  int number_of_tasks = 0;
  double task_comp_size = 0;
  double task_comm_size = 0;

  int i;

  int res = sscanf(argv[1], "%d", &number_of_tasks);
  xbt_assert(res,"Invalid argument %s\n", argv[1]);
  res = sscanf(argv[2], "%lg", &task_comp_size);
  xbt_assert(res, "Invalid argument %s\n", argv[2]);
  res = sscanf(argv[3], "%lg", &task_comm_size);
  xbt_assert(res, "Invalid argument %s\n", argv[3]);

  {                               /*  Task creation */
    char sprintf_buffer[64];

    todo = xbt_new0(m_task_t, number_of_tasks);

    for (i = 0; i < number_of_tasks; i++) {
      sprintf(sprintf_buffer, "Task_%d", i);
      todo[i] =
          MSG_task_create(sprintf_buffer, task_comp_size, task_comm_size,
                          NULL);
    }
  }
```

Figura 10. Código do *master* - parte 1. Fonte: [SimGrid 2011].

O código do *master* responsável pela organização do processo é exibido na figura 11. Nesse código inicialmente é alocado espaço para armazenar referências aos *slaves*. As funções *XBT_INFO* e *XBT_DEBUG* registram no *log* uma mensagem com status de informação e debug respectivamente. O principal *loop* desse código vai enviar para todos os *slaves* as tarefas que devem ser simuladas através da função: *MSG_task_put* . Depois que todas as tarefas foram enviadas para os *slaves;* outra tarefa, com o nome de "*finalize*", é criada com o propósito de informar a eles que as tarefas de simulação acabaram. No código do *slave,* que será apresentado a seguir, haverá uma verificação no *loop* da simulação para verificar se alguma tarefa foi criada com esse nome. Dessa forma, o *slave* sabe se a simulação finalizou.

```
  {                              /* Process organisation */
    slaves_count = argc - 4;
    slaves = xbt_new0(m_host_t, slaves_count);

    for (i = 4; i < argc; i++) {
      slaves[i - 4] = MSG_get_host_by_name(argv[i]);
      xbt_assert(slaves[i - 4] != NULL, "Unknown host %s. Stopping Now! ",
                 argv[i]);
    }
  }

  XBT_INFO("Got %d slaves and %d tasks to process", slaves_count,
        number_of_tasks);
  for (i = 0; i < slaves_count; i++)
    XBT_DEBUG("%s", slaves[i]->name);

  for (i = 0; i < number_of_tasks; i++) {
    XBT_INFO("Sending \"%s\" to \"%s\"",
          todo[i]->name, slaves[i % slaves_count]->name);
    if (MSG_host_self() == slaves[i % slaves_count]) {
      XBT_INFO("Hey ! It's me ! :)");
    }

    MSG_task_put(todo[i], slaves[i % slaves_count], PORT_22);
    XBT_INFO("Sent");
  }

  XBT_INFO
      ("All tasks have been dispatched. Let's tell everybody the computation is over.");
  for (i = 0; i < slaves_count; i++) {
    m_task_t finalize = MSG_task_create("finalize", 0, 0, FINALIZE);
    MSG_task_put(finalize, slaves[i], PORT_22);
  }

  XBT_INFO("Goodbye now!");
  free(slaves);
  free(todo);
  return 0;
}                              /* end_of_master */
```

Figura 11. Código do *master* - parte 2. Fonte: [SimGrid 2011].

O código do *slave*, figura 12, é responsável por receber as tarefas enviadas pelo *forwarder* e executá-las. As funções que assumem esse papel são *MSG_task_get* e *MSG_task_execute,* respectivamente. A *MSG_task_execute* bloqueia a execução até que a tarefa seja finalizada. Essas funções são executadas em um *loop* até que o *slave* receba uma tarefa indicando que o processamento deve finalizar.

```
int slave(int argc, char *argv[])
{
  m_task_t task = NULL;
  int res;
  while (1) {
    res = MSG_task_get(&(task), PORT_22);
    xbt_assert(res == MSG_OK, "MSG_task_get failed");

    XBT_INFO("Received \"%s\"", MSG_task_get_name(task));
    if (!strcmp(MSG_task_get_name(task), "finalize")) {
      MSG_task_destroy(task);
      break;
    }

    XBT_INFO("Processing \"%s\"", MSG_task_get_name(task));
    MSG_task_execute(task);
    XBT_INFO("\"%s\" done", MSG_task_get_name(task));
    MSG_task_destroy(task);
    task = NULL;
  }
  XBT_INFO("I'm done. See you!");
  return 0;
}                                /* end_of_slave */
```

Figura 12. Código do *slave*.Fonte: [SimGrid 2011].

As figuras 13 e 14 mostram o código do *forwarder* que tem o papel, nesse exemplo, de esperar as tarefas criadas pelo *master* e em seguida enviá-las para os *slaves*.

```
int forwarder(int argc, char *argv[])
{
  int i;
  int slaves_count;
  m_host_t *slaves;

  {                              /* Process organisation */
    slaves_count = argc - 1;
    slaves = xbt_new0(m_host_t, slaves_count);

    for (i = 1; i < argc; i++) {
      slaves[i - 1] = MSG_get_host_by_name(argv[i]);
      if (slaves[i - 1] == NULL) {
        XBT_INFO("Unknown host %s. Stopping Now! ", argv[i]);
        abort();
      }
    }
  }
```

Figura 13. Código do *forwarder* - parte 1. Fonte: [SimGrid 2011].

```
  i = 0;
  while (1) {
    m_task_t task = NULL;
    int a;
    a = MSG_task_get(&(task), PORT_22);
    if (a == MSG_OK) {
      XBT_INFO("Received \"%s\"", MSG_task_get_name(task));
      if (MSG_task_get_data(task) == FINALIZE) {
        XBT_INFO
            ("All tasks have been dispatched. Let's tell everybody the computation is over.");
        for (i = 0; i < slaves_count; i++)
          MSG_task_put(MSG_task_create("finalize", 0, 0, FINALIZE),
                       slaves[i], PORT_22);
        MSG_task_destroy(task);
        break;
      }
      XBT_INFO("Sending \"%s\" to \"%s\"",
           MSG_task_get_name(task), slaves[i % slaves_count]->name);
      MSG_task_put(task, slaves[i % slaves_count], PORT_22);
      i++;
    } else {
      XBT_INFO("Hey ?! What's up ? ");
      xbt_die("Unexpected behavior");
    }
  }
  xbt_free(slaves);

  XBT_INFO("I'm done. See you!");
  return 0;
}                               /* end_of_forwarder */
```

Figura 14. Código do *forwarder*- parte 2. Fonte: [SimGrid 2011].

O código escrito em *test_all*, figura 15, é o responsável pela parte principal da simulação do usuário. Essa função recebe como parâmetro o caminho de dois arquivos XML compatível com o SURF. O primeiro arquivo descreve a plataforma e o segundo a aplicação. A função *MSG_create_environment* cria um ambiente realístico conforme arquivo XML recebido por parâmetro. A função *MSG_launch_application*, conforme arquivo XML, cria os *agents* nos locais apropriados e a *MSG_main* executa a simulação. Os *agents master*, *slave* e *forwarder* não podem ser chamados diretamente, por isso é necessário a função *MSG_function_register*. Essa função registra o *agent* em uma tabela global, que é usada pela *MSG_launch_application.*

```
MSG_error_t test_all(const char *platform_file,
                     const char *application_file)
{
  MSG_error_t res = MSG_OK;

  /* MSG_config("workstation/model","KCCFLN05"); */
  {                                /*  Simulation setting */
    MSG_set_channel_number(MAX_CHANNEL);
    MSG_create_environment(platform_file);
  }
  {                                /*   Application deployment */
    MSG_function_register("master", master);
    MSG_function_register("slave", slave);
    MSG_function_register("forwarder", forwarder);
    MSG_launch_application(application_file);
  }
  res = MSG_main();

  XBT_INFO("Simulation time %g", MSG_get_clock());
  return res;
}                                /* end_of_test_all */
```

Figura 15. Código da função *test_all*. Fonte: [SimGrid 2011].

A função *main,* figura 16, verifica se os arquivos XML foram informados e, em seguida, chama a função *test_all.*

```
int main(int argc, char *argv[])
{
  MSG_error_t res = MSG_OK;

  MSG_global_init(&argc, argv);
  if (argc < 3) {
    printf("Usage: %s platform_file deployment_file\n", argv[0]);
    printf("example: %s msg_platform.xml msg_deployment.xml\n", argv[0]);
    exit(1);
  }
  res = test_all(argv[1], argv[2]);
  MSG_clean();

  if (res == MSG_OK)
    return 0;
  else
    return 1;
}                                /* end_of_main */
```

Figura 16. Código da função *main*. Fonte: [SimGrid 2011].

Os códigos apresentados anteriormente representam uma simulação simples da uma aplicação *master/slave* utilizando a API MSG. Como pode ser observado, o código utiliza dois arquivos XML de configuração. É através desses arquivos que as possibilidades de simulação no SimGrid se tornam inúmeras. Um arquivo chamado de *application*/ *deployment* representa quais são os *host* disponíveis, a função de cada um na simulação e indica também parâmetros que serão utilizados na simulação. Na figura 17, como pode ser observado, foi definido no arquivo que a simulação terá 5 *hosts*. Sendo que o *Tremblay* que é *master*, também, pode fazer processamento como *slave*. Como parâmetros do *master* foram definidos: a quantidade de tarefas, o tamanho do computo e comunicação das tarefas.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
  <!-- The master process (with some arguments) -->
  <process host="Tremblay" function="master">
     <argument value="20"/>       <!-- Number of tasks -->
     <argument value="50000000"/>  <!-- Computation size of tasks -->
     <argument value="1000000"/>   <!-- Communication size of tasks -->
     <argument value="Jupiter"/>  <!-- First slave -->
     <argument value="Fafard"/>   <!-- Second slave -->
     <argument value="Ginette"/>  <!-- Third slave -->
     <argument value="Bourassa"/> <!-- Last slave -->
     <argument value="Tremblay"/> <!-- Me! I can work too! -->
  </process>
  <!-- The slave process (with no argument) -->
  <process host="Tremblay" function="slave"/>
  <process host="Jupiter" function="slave"/>
  <process host="Fafard" function="slave"/>
  <process host="Ginette" function="slave"/>
  <process host="Bourassa" function="slave"/>
</platform>
```

Figura 17. Arquivo XML de configuração da aplicação. Fonte: [SimGrid 2011].

O outro arquivo XML é chamado de *platform* e nele pode ser modelado o poder de processamento dos *hosts*, a banda e latência dos *links* que interligam os *hosts* e a topologia da rede. A figura 18 mostra apenas uma parte desse arquivo. Os parâmetros disponíveis nesses arquivos não se restringem aos apresentados. Existem diversos outros que podem ser utilizados para modelar a aplicação e plataforma a mais realística possível.

```
<?xml version='1.0'?>
 <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
 <platform version="3">
 <AS  id="AS0"  routing="Full">
   <!-- ljlkj -->
   <host id="Tremblay" power="98095000"/>
   <host id="Jupiter" power="76296000"/>
   <host id="Fafard" power="76296000"/>
   <host id="Ginette" power="48492000"/>
   <host id="Bourassa" power="48492000"/>
   <link id="6" bandwidth="41279125" latency="5.9904e-05"/>
   <link id="11" bandwidth="252750" latency="0.00570455"/>
   <link id="3" bandwidth="34285625" latency="0.000514433"/>
   <link id="7" bandwidth="11618875" latency="0.00018998"/>
   <link id="9" bandwidth="7209750" latency="0.001461517"/>
   <link id="12" bandwidth="1792625" latency="0.007877863"/>
   <link id="2" bandwidth="118682500" latency="0.000136931"/>
   <link id="8" bandwidth="8158000" latency="0.000270544"/>
   <link id="1" bandwidth="34285625" latency="0.000514433"/>
   <link id="4" bandwidth="10099625" latency="0.00047978"/>
   <link id="0" bandwidth="41279125" latency="5.9904e-05"/>
   <link id="10" bandwidth="4679750" latency="0.000848712"/>
   <link id="5" bandwidth="27946250" latency="0.000278066"/>
   <link id="loopback" bandwidth="498000000" latency="0.000015" sharing_policy="FATPIPE"/>
   <route src="Tremblay" dst="Tremblay"><link_ctn id="loopback"/></route>
   <route src="Jupiter" dst="Jupiter"><link_ctn id="loopback"/></route>
   <route src="Fafard" dst="Fafard"><link_ctn id="loopback"/></route>
   <route src="Ginette" dst="Ginette"><link_ctn id="loopback"/></route>
   <route src="Bourassa" dst="Bourassa"><link_ctn id="loopback"/></route>
   <route src="Tremblay" dst="Jupiter">
     <link_ctn id="9"/>
   </route>
   <route src="Tremblay" dst="Fafard">
     <link_ctn id="4"/><link_ctn id="3"/><link_ctn id="2"/><link_ctn id="0"/><link_ctn id="1"/><link_ctn id="8"/>
   </route>
```

Figura 18. Arquivo XML de configuração da plataforma. Fonte: [SimGrid 2011].

Para compilar o código descrito, execute a seguinte linha de comando:

gcc -I/<PATH_INSTALL_SIMGRID>/include -L/<PATH_INSTALL_SIMGRID>/lib masterslave/masterslave_forwarder.c -lsimgrid -o masterslave/masterslave_forwarder

Para executar o código descrito, execute a seguinte linha de comando:

./masterslave/masterslave_forwarder msg_platform.xml masterslave/ deployment_masterslave_forwarder.xml

As linhas de código anteriores consideram que o usuário está na pasta de exemplos MSG do SimGrid: /simgrid/doc/simgrid/examples/msg.

Nesse exemplo, a saída *default* é a exibida na figura 19.

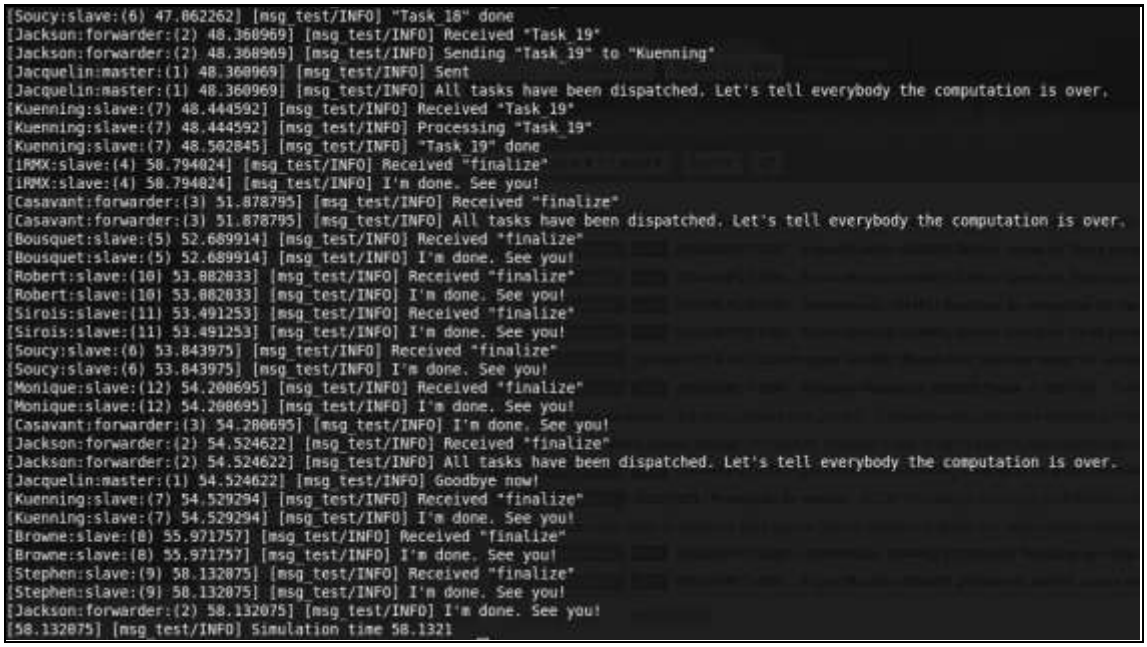

Figura 19. Saída padrão de uma simulação. Fonte: Autor.

Como pode ser observada na figura 19 a saída não é amigável. Para simulações pequenas, como a apresentada, existe uma alternativa desenvolvida por SimGrid (2011) que customiza a saída com cores. Essa solução é possível através de script desenvolvido em *perl* que está disponível na pasta: *tools/MSG_visualization/colorize.pl*. Para executar o código com a saída customizada, digite a seguinte linha de comando:

./masterslave/masterslave_forwarder msg_platform.xml masterslave/ deployment_ masterslave_forwarder.xml 2>&1 |../../../../build_dir/simgrid-3.6.1/tools/ MSG_visualization /colorize.pl

Fazer uma análise de dados com uma saída nesse formato, mesmo com a saída um pouco mais amigável, não é uma tarefa viável, principalmente em aplicações distribuídas que o volume de dados é muito grande. Para simulações maiores existem soluções mais robustas desenvolvidas por terceiros que auxiliam na análise dos dados gerados pelas simulações. Abordamos na próxima seção essas soluções.

6. Visualizando os dados da simulação

Segundo SimGrid(2011), a visualização de *trace* é uma técnica amplamente utilizada para se observar e entender o comportamento de algoritmos paralelos e distribuídos. A utilização dessa técnica consiste em duas etapas: (i) inserir linhas de código que registrem dados que merecem análise no arquivo de *trace*; (ii) os dados registrados no *trace* serão analisados depois da execução.

A visualização desses dados permite observar comportamentos inesperados, gargalos e pode ajudar a corrigir problemas nos algoritmos paralelos. Existem diversas funções disponíveis no SimGrid que permitem o usuário registrar em seus códigos o *trace* . Um exemplo pode ser observado na figura 20.

```
int main (int argc, char **argv)
{
  MSG_global_init (&argc, &argv);

  //(... after deployment ...)

  //note that category declaration must be called after MSG_create_environment
  TRACE_category_with_color ("request", "1 0 0");
  TRACE_category_with_color ("computation", "0.3 1 0.4");
  TRACE_category ("finalize");

  m_task_t req1 = MSG_task_create("1st_request_task", 10, 10, NULL);
  m_task_t req2 = MSG_task_create("2nd_request_task", 10, 10, NULL);
  m_task_t req3 = MSG_task_create("3rd_request_task", 10, 10, NULL);
  m_task_t req4 = MSG_task_create("4th_request_task", 10, 10, NULL);
  TRACE_msg_set_task_category (req1, "request");
  TRACE_msg_set_task_category (req2, "request");
  TRACE_msg_set_task_category (req3, "request");
  TRACE_msg_set_task_category (req4, "request");

  m_task_t comp = MSG_task_create ("comp_task", 100, 100, NULL);
  TRACE_msg_set_task_category (comp, "computation");

  m_task_t finalize = MSG_task_create ("finalize", 0, 0, NULL);
  TRACE_msg_set_task_category (finalize, "finalize");

  //(...)

  MSG_clean();
  return 0;
}
```

Figura 20. Exemplo de código com funções de *trace*. Fonte: [SimGrid 2011].

No código acima são criadas três categorias: *request*, *computation* e *finalize*. As categorias são criadas para representar as diferentes tarefas que serão criadas durante a execução da simulação. A função *TRACE_msg_set_task_category* associa uma tarefa criada anteriormente a determinada categoria. Dessa forma, será registrado no arquivo de *trace* correspondente o quanto de computo e banda são usados por cada *host* e *link* definido na plataforma. O propósito é que o *trace* apresente uma visão geral da utilização dos recursos, possibilitando identificar possíveis problemas. As tarefas, que não forem associadas à determinada categoria, não serão registradas. As funções de *trace* são disponíveis nas APIs: MSG, SMPI e SimDAG. Para que as funções de *trace* funcionem é necessário que o SimGrid tenha sido configurado com o parâmetro *-Denable_tracing=ON* no momento da instalação [SimGrid 2011].

Depois que o trace foi gerado é necessário fazer a análise dos dados. Os arquivos criados pelo *trace* do SimGrid são no formato Pajé [Kergommeaux e Stein 2000]. Este arquivo possui todas as informações dos recursos usados na simulação. Esse formato pode ser visualizado pela ferramenta Pajé ou pelo Triva.

O Triva implementa diversas técnicas de visualização e permite a criação de novas técnicas de visualização de dados. As diferentes representações ajudam a detectar anomalias e entender alguns comportamentos inesperados [Schnorr, Legrand e Vincent 2010]. O

SimGrid foi apresentado na SuperComputing 2010 em New Orleans, nessa conferência foram exibidas três demonstrações relacionadas à análise de simulações. Essas demonstrações estão disponíveis em: Video1(2011), Video2(2011) e Video3(2011) e mostram como a técnica de visualização de *trace* pode ser utilizada para detectar problemas.

7. Aspectos de experimentações em PAD

Quinson, em sua página pessoal [Quinson 2011], descreve o SimGrid da seguinte forma: "SimGrid: o Chuck Norris da simulação em sistema distribuído". De fato, como observamos nas seções anteriores, o SimGrid permite efetuar simulação em P2P, Computação voluntária, Sistemas de Distribuídos em larga escala, Computação em Nuvem e Computação de Alto Desempenho com diversos parâmetros, configurações e cenários.

Apesar de o SimGrid suportar diversas soluções em aplicações distribuídas e paralelas, é importante destacar que algumas delas possuem requisitos diferentes das soluções de PAD. Em PAD o requisito principal é eficiência no tempo de processamento. Na literatura, as principais soluções em PAD são disponíveis em supercomputadores dedicados, *grid computing* e os *commodity clusters.* Podem existir diversas configurações de uma solução de alto desempenho. Nesse escopo, o que diferencia as soluções de PAD das outras soluções de sistemas distribuídos é como o problema será modelado no SimGrid, tanto no aspecto da API quanto na plataforma. Os arquivos de configurações do SimGrid oferecem vários parâmetros que auxiliam essa modelagem. Na figura 21, por exemplo, é modelado o *deployment* de uma aplicação que será executada em uma máquina *multicore.*

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
  <!-- The master process (with some arguments) -->
  <process host="Tremblay" function="master">
     <argument value="20"/>       <!-- Number of tasks -->
     <argument value="50000000"/>  <!-- Computation size of tasks -->
     <argument value="1000000"/>   <!-- Communication size of tasks -->
     <argument value="6"/>         <!-- Number of slaves -->
  </process>
  <!-- The slave processes (with mailbox to listen on as argument) -->
  <process host="Tremblay" function="slave">
    <argument value="0"/>
  </process>
  <process host="Tremblay" function="slave">
    <argument value="1"/>
  </process>
  <process host="Tremblay" function="slave">
    <argument value="2"/>
  </process>
  <process host="Tremblay" function="slave">
    <argument value="3"/>
  </process>
  <process host="Tremblay" function="slave">
    <argument value="4"/>
  </process>
  <process host="Tremblay" function="slave">
    <argument value="5"/>
  </process>

</platform>
```

Figura 21. Exemplo de arquivo de *deployment*. Fonte: [SimGrid 2011].

Outro exemplo é a figura 22 que modela uma topologia de dois *clusters* homogêneos, cada um com 500 *nodes* ligados por um *backbone* de alta velocidade. A pasta de exemplos do SimGrid possuem diversos exemplos de código e descritores de *deployment* e *platform* como esses.

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
<AS  id="AS0"  routing="Full">
  <cluster id="my_cluster_1" prefix="c-" suffix=".me"
               radical="0-499" power="1000000000"    bw="125000000"     lat="5E-5"
       bb_bw="2250000000" bb_lat="5E-4"/>

  <cluster id="my_cluster_2" prefix="c-" suffix=".me"
           radical="500-999" power="1000000000"        bw="125000000"  lat="5E-5"
           bb_bw="2250000000" bb_lat="5E-4"/>

    <link id="backbone" bandwidth="1250000000" latency="5E-4"/>

    <ASroute src="my_cluster_1" dst="my_cluster_2"
        gw_src="c-my_cluster_1_router.me"
        gw_dst="c-my_cluster_2_router.me">
              <link_ctn id="backbone"/>
    </ASroute>
    <ASroute src="my_cluster_2" dst="my_cluster_1"
        gw_src="c-my_cluster_2_router.me"
        gw_dst="c-my_cluster_1_router.me">
              <link_ctn id="backbone"/>
    </ASroute>
</AS>
</platform>
```

Figura 22. Exemplo de arquivo *platform*. Fonte: [SimGrid 2011].

Para efetuar simulações em PAD, o pesquisador pode a princípio ter dois cenários: (i) possuir uma topologia de PAD onde deseja validar uma solução específica; ou (ii) uma solução que deseja testar em diversas topologias de PAD.

No primeiro cenário, como o propósito é avaliar uma solução específica, a primeira etapa é mapear a infra de PAD para os arquivos de XML de configuração. A máquina é um computador que possui diversos núcleos? A máquina possui memória distribuída? Qual é a latência da rede? Existe *switch*? Qual é a topologia da rede? É necessário que o problema seja mapeado para os arquivos de configuração do SimGrid. Para auxiliar a descrever a topologia existe a ferramenta SIMULACRUM, que é descrita por Quinson, Bobelin e Suter (2010). A segunda etapa é definir qual API utilizar. O MSG deve ser escolhido se a pesquisa for estudar heurísticas ou estratégias de algoritmos paralelos. O SimDAG deve ser a opção quando a solução proposta é do tipo DAG; pois, nessa API, é possível analisar as estratégias utilizadas para distribuir de forma eficiente as tarefas que possuem dependências. Se o pesquisador tiver o propósito de desenvolver uma aplicação em MPI ou quiser estudar essa interface a escolha deve ser o SMPI. Se a topologia de PAD estiver configurada em um Grid a solução é o GRAS. Vale observar que SMPI e GRAS permitem que o mesmo código utilizado na simulação seja usado no ambiente real.

Para o segundo cenário, as considerações anteriores sobre as API também se aplicam. Para o caso da topologia o propósito é verificar o comportamento, nos mais diversos parâmetros, cenários e configurações. No entanto, especificar a topologia no arquivo XML não é uma atividade trivial, principalmente se tiver centenas e milhares de *hosts*, *links* e processadores. O *Platform Description Archive* – PDA [PDA 2011] é um projeto cujo esforço é disponibilizar arquivos com os descritores de plataforma para os usuários de simulação. O PDA disponibiliza arquivos que podem ser utilizados para efetuar as validações em diversos cenários.

Pelo menos por enquanto, um modelo para a simulação de GPGPU e programação hibrida ainda não foi implementado no SimGrid. Segundo Quinson (2001) a simulação de GPGPU é realmente uma questão complexa; porém, inevitavelmente, terá que ser disponibilizado.

8. Considerações Finais.

Em Processamento de Alto Desempenho, as soluções implementadas vão de hardware especializado até software. Os avanços das pesquisas em hardware são diversos e inúmeros. A prova disso é a lista do TOP500 [TOP500], que há mudanças constantes dos dez primeiros supercomputadores mais potentes do mundo. Dessa forma, é necessário que o software evolua de forma que esses benefícios possam, de fato, ser aproveitados. É nesse contexto que os estudos por novos algoritmos, estratégias de scheduling, estratégias de mapeamento, balanceamento de carga, entre outros desafios da área, são necessários. Para auxiliar as pesquisas nessa área uma solução bastante utilizada pela comunidade cientifica é a simulação.

O SimGrid é uma framework para o desenvolvimento de simulações em aplicações distribuídas e paralelas que pode auxiliar no estudo de soluções em PAD. O SimGrid disponibiliza para os usuários diversas APIs que permitem a modelagem do software. Já a parte da infraestrutura é modelada através dos arquivos de configuração XML que representam a aplicação e a plataforma. Dessa forma, o SimGrid permite que os desafios em PAD possam ser simulados e pesquisados.

O SimGrid é um projeto atual e que constantemente tem incorporado novas características. Em paralelo existe também o desenvolvimento de novas ferramentas que dão suporte ao *framework*. Ao longo dos anos, a base de usuário do SimGrid aumentou mostrando a qualidade da ferramenta em simulação. São mais de 70 trabalhos publicados, em eventos, conferências e *journals* sobre a ferramenta além de existir centenas de pesquisadores ao redor do mundo desenvolvendo seus trabalhos com o SimGrid.

O maior problema no SimGrid é a documentação. Não é difícil encontrar páginas em SimGrid (2011) desatualizadas ou com informações incompletas. As alternativas para resolver essa questão são: (i) buscar artigos recentemente publicados; (ii) baixar o código fonte do repositório mais atual e procurar a informação desejada; (iii) solicitar ou buscar informações na lista de discussão [SIMGRID-USER 2011]; (iv) analisar os diversos exemplos disponíveis no pacote de instalação do SimGrid. Apesar da documentação do SimGrid não ser boa, isso não é um impeditivo para ser utilizado. Os pontos positivos superam as possíveis dificuldades documentais.

O ponto inicial de estudos do SimGrid é a *FAQ* disponível em SimGrid(2011). Lá existem diversas respostas e orientações de como iniciar as simulações. O SimGrid se demonstra como uma excelente ferramenta para o desenvolvimento de experimentos simulados e tem demonstrado ser um ótimo caminho para se obter rápidos resultados e publicações na área da computação distribuída e paralela.

Referências

Bolze, R.; Cappello F.; Caron E.; Daydé M.; Desprez F.; Jeannot E.; Jégou Y.; Lanteri S.; Leduc J.; Melab N.; Mornet G.; Namyst R.; Primet P.; Quetier B.; Richard O.; Talbi E. G.; and Touche I.. Grid'5000: A Large Scale And Highly Recon_gurable Experimental Grid Testbed. International Journal of High Performance Computing Applications, 20(4):481-494, 2006.

Caniou, Yves; Charrier, Ghislain; Desprez, Frédéric. Evaluation of Reallocation Heuristics for Moldable Tasks in Computational Grids. In ACM, editor, Proceedings of the 9th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2011), Perth, Australia, pages 10, January 17-20 2011.

Casanova, H.; Legrand A.; Quinson, M. SimGrid: a Generic Framework for Large-Scale Distributed Experimentations. In Proceedings: 10th IEEE International Conference on Computer Modeling and Simulation, 2008.

Casanova, H.; Marchal, L. A Network Model for Simulation of Grid Application. Research Report 2002-40, LIP, ENS Lyon, France, 2002.

Casanova, Henri. SimGrid: A Toolkit for the Simulation of Application Scheduling. In Proceedings: First IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 15-18, 2001, Brisbane, Australia. IEEE Computer Society, 2001.

Chun B.; Culler D.; Roscoe, T.; Bavier A.; Peterson L.; Wawrzoniak M.; and Bowman M. PlanetLab: an overlay testbed for broad-coverage services. ACM SIGCOMM Comput. Commun. Rev., 33(3):3-12, 2003.

Cornea, Bogdan Florin; Bourgeois Julien. Performance Prediction of Distributed Applications Using Block Benchmarking Methods. In PDP '11: 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing, 2011. IEEE Computer Society. Keyword(s): dPerf, performance prediction, block benchmarking, static analysis, trace-based simulation, heterogeneous systems, MPI, P2PSAP.

Grid5000. Grid 5000 Project. Disponível em www.grid5000.fr/. Acesso em: 13/08/2011.

Howe, Denis. Free On-line Dictionary Of Computing. Disponível em http://foldoc.org. Acesso em: 07/08/2011.

Hunold, Sascha. Low-Cost Tuning of Two-Step Algorithms for Scheduling Mixed-Parallel Applications onto Homogeneous Clusters. In Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010), Melbourne, Australia, pages 253-262, May 2010.

Kergommeaux , J. Chassin; Stein, B. de Oliveira. Pajé: an extensible environment for visualizing multi-threaded programs executions. EuroPar2000, Proceedings of the 6th International EuroPar Conference, A. Bode, W. Ludwig, T. Karl and R. Wismüller (ed.), LNCS, 1900, Springer, pp. 133-140, Munich, 2000.

Legrand, A.; Lerouge, L. MetaSimGrid: Towards realistic scheduling simulation of distributed applications, Research Report 2002-28, LIP, ENS Lyon, France, 2002.

Legrand, A.; Marchal L.; Casanova, H. Scheduling Distributed Applications: the SimGrid Simulation Framework. In Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan, pages 138-145, May 2003.

Munck, Silas De; Vanmechelen, Kurt; Broeckhove, Jan. Improving The Scalability of SimGrid Using Dynamic Routing. In Proceedings of the 9th International Conference on Computational Science (ICCS'09), volume 5544 of Lecture Notes in Computer Science, Baton Rouge, LA, pages 406-415, 2009. Springer.

Pacheco, Peter. Parallel Programming with MPI. Morgan Kaufmann Publishers Inc. 1st edition, 1996.

PDA. The Platform Description Archive. Disponível em http://pda.gforge.inria.fr/. Acesso em: 30/08/2011.

Quinson, M.; Bobelin L.; Suter F. Synthesizing Generic Experimental Environments for Simulation, Fifth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (2010), November 2010, Fukuoka, Japan.

Quinson, Martin. Experimenting HPC Systems with Simulation. Tutorial at the 8th ACM/IEEE Internacional Conference on High Performance Computing & Simulation (HPCS'10), Caen, France, June 28, 2010.

Quinson, Martin. Gras: A Research & Development Framework for Grid and P2P Infrastructures. In Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'06), Dallas, TX, November 2006.

Quinson, Martin. Página de Martin Quinson. Disponível em http://www.loria.fr/~quinson/. Acesso em: 30/08/2011.

Santana, Francisco J. S. B. Um Modelo de Balanceamento de Carga para Inclusão Dinâmica de Workers em HNOW. 2007. 238 f. Dissertação (Mestrado em Modelagem Computacional) – Fundação Visconde de Cairu, Centro de Pós-Graduação de Pesquisa Visconde de Cairu. Salvador. 2007.

Santos-Neto, E.; Cirne W.; Brasileiro, F. and Lima A.. Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids. In Proceedings of 10th Job Scheduling Strategies for Parallel Processing, June 2004.

Schnorr , L. M.; Legrand, A.; Vincent J.-M. Visualization and Detection of Resource Usage Anomalies in Large Scale Distributed Systems. Research Report RR-7438, INRIA, 10 2010.

Senger, Hermes; Silva, Fabricio A.B. da; Miranda Filho, Luciano J., Influence of Communication Models on the Scalability of Master-Slave Platforms Running Bag-of-Tasks Applications, Computing Systems, Symposium on, pp. 25-32, 2010 11th Symposium on Computing Systems, 2010.

SimGrid. SimGrid Project. Disponível em http://simgrid.gforge.inria.fr/. Acesso em: 16/08/2011.

SIMGRID-USER. Lista mantida pelo INRIA. Disponível em: <simgrid-user@lists.gforge.inria.fr>. Acesso em: 29 ago. 2011.

TOP500. TOP500 Supercomputing Sites. Disponível em http://top500.org. Acesso em: 07/08/2011.

Video1. Simulating MPI Applications - Gantt Chart Analysis. Disponível em http://www.youtube.com/watch?v=ivSm4FaS0HU. Acesso em: 30/08/2011.

Video2. Pinpointing Network Bottlenecks of MPI Application. Disponível em http://www.youtube.com/watch?v=NOxFOR_t3xI. Acesso em: 30/08/2011.

Video3. Simulating and Analyzing Very Large Volunteer Comp (BOINC). Disponível em http://www.youtube.com/watch?v=OiDmYXe3008. Acesso em: 30/08/2011.