HLogP: Um Modelo de Escalonamento para a Execução de Aplicações MPI em Grades Computacionais

Helder de Amorim Mendes

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação Aplicada e Automação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Processamento Distribuído e Paralelo.

Orientador: Prof. Eugene Francis Vinod Rebello. PhD.

Niterói, 21 de dezembro de 2004.

HLogP: Um Modelo de Escalonamento para a Execução de Aplicações MPI em Grades Computacionais

Helder de Amorim Mendes

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação Aplicada e Automação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Processamento Distribuído e Paralelo.

Aprovada por:

Prof. Eugene Francis Vinod Rebello, PhD - IC/UFF (Presidente)

Profa. Maria Cristina Silva Boeres, PhD - IC/UFF

Prof. Alberto Ferreira De Souza, PhD - DI/UFES

Niterói, Dezembro de 2004

"*Dedico este trabalho a minha esposa, meus filhos e minha Mãe.*"

Agradecimentos

A Deus por me conceder capacidade e o privilégio de poder ser um dos poucos Brasileiros que conseguem obter o título de Mestre.

À minha esposa e meus filhos pela paciência. Em muitos momentos pensei que eles não iriam aguentar a minha ausência e, quando na presença, os momentos de nervosismo.

À minha mãe pelo apoio e incentivo para que nunca desistisse.

Ao meu orientador Vinod, pelo apoio e dedicação nas horas mais difíceis onde, sem o seu conhecimento e apoio, este trabalho não seria possível.

À Fundação Fafile de Carangola pelo apoio financeiro.

Aos meus colegas de mestrado: Eyder, Stênio, Eduardo, Rodrigo, Jacques, Daniela, Viviane, Cristiane Jonivan entre outros que se não citados foram da mesma forma muito importantes na ajuda e no apoio nas horas difíceis.

Resumo da Dissertação de Mestrado apresentada à UFF como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada e Automação (M.Sc.)

HLogP: Um Modelo de Escalonamento para a Execução de Aplicações MPI em Grades Computacionais

Helder de Amorim Mendes

Novembro, 2004

Orientador: Eugene Francis Vinod Rebello, PhD.
Pós-Graduação em Computação Aplicada e Automação

Para que aplicações paralelas sejam executadas eficientemente em Grades (*Grids*), suas tarefas devem ser escalonadas da melhor forma possível. Para que um escalonador seja capaz de tomar decisões apropriadas, deve-se utilizar um modelo que retrate de forma mais precisa possível as características do ambiente sobre o qual a aplicação será executada. A maioria dos modelos existentes atualmente, não consideram características importantes dos ambientes Grid como, por exemplo, a heterogeneidade e a comunicação. Neste trabalho propomos um modelo para o escalonamento de aplicações paralelas MPI em ambientes Grid. Os parâmetros considerados no modelo capturam as características heterogêneas dos ambientes Grid, permitindo que escalonamentos mais eficientes sejam obtidos. Devido ao comportamento dinâmico de ambientes Grid, o modelo tem que ser calibrado cada vez que uma aplicação vai ser executada. É proposta, também, uma ferramenta para a calibragem dos valores dos parâmetros do modelo para utilização por escalonadores estáticos e dinâmicos em ambientes Grid. Além disso, são feitos comentários sobre a observação de que, em alguns casos, as duas implementações da plataforma MPI existentes, habilitadas a Grid, apresentam comportamentos bastante diferentes.

Palavras-chave: grades computacionais, modelagem, aplicações MPI, escalonamento de processos (ou tarefas).

Abstract of the thesis submitted to UFF as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

HLogP: A Scheduling Model for the Execution of MPI Aplications in Grid Environments

Helder de Amorim Mendes

November 2004

Advisor: Eugene Francis Vinod Rebello
Department: Computer Science

In order to execute MPI applications efficently in grid environments, MPI processes need to be allocated appropriately to the resources available. Devising efficent process allocations requires a scheduling model which is sufficiently accurate to reflect the performance characteristics of the target system. It is still unclear if existing parallel program computing performance models are appropriate for grid environments. To date no model has been proposed for scheduling parallel applications in grids, due principally to the difficulty of taking in consideration characteristics such as the heterogenity of both computation and communication resources. This work proposes one such model for scheduling MPI applications in computational grids. The parameters of the model capture these heterogenous characteristics, enabling efficent schedules to be obtained. Due to the dynamic behaviour of grid environments, the model needs to be calibrated before the application is executed (for static scheduling) and during execution (for dynamic scheduling). A tool is presented which efficiently calibrates the parameters of the proposed model for use by static and dynamic grid schedulers. In addition, this work also comments on the observation that, in some cases, the two existing grid enabled implementations of MPI exhibit quite differing behavious.

Keywords: computational grids, modeling, MPI applications, task (process) scheduling.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Em diversas áreas do conhecimento humano, tais como a ambiental simulações oceânicas e climáticas, econômica (simulações financeiras), física ou química, entre outras, existem problemas de grande importância econômica que requerem um poder computacional muito intenso (ver por exemplo, os Grandes Desafios [38]). Ainda que processadores cada vez mais velozes sejam construídos, a solução desses problemas em tempo satisfatório e viável ainda é um desafio. A busca por maior capacidade de processamento, permitindo que soluções de grandes problemas computacionais sejam obtidas mais rapidamente, é muito importante.

Os sistemas de computação de alto poder computacional, baseavam-se em arquiteturas paralelas de recursos especializados, onde várias unidades de processamento estão interligadas por mecanismos de comunicação de alta velocidade. Tais sistemas, os mais rápidos denominados supercomputadores, têm um custo elevado, tornando este tipo de computação de alto desempenho uma alternativa fora do alcance de muitas pessoas. Além de proibitivos no que se refere aos custos de desenvolvimento, aquisição e manutenção, tais sistemas possuem um grau de dificuldade de utilização mais elevado do que sistemas sequenciais tradicionais. Nestes sistemas tradicionais, as instruções de uma aplicação são executadas por uma única unidade de processamento de forma sequencial. Já nos sistemas paralelos, cada unidade de processamento pode executar uma instrução (ou processo) ao mesmo tempo que as demais também podem estar executando a mesma instrução (processo) ou uma

outra instrução distinta. O desempenho de um sistema paralelo é influenciado não só pelo poder computacional das unidades de processamento (cpu, memória, disco local, etc), mas também pelas características da rede que conecta essas unidades. O desenvolvimento de aplicações para esses sistemas ainda é para programadores, cientistas e engenheiros, uma tarefa complicada, requerendo o estudo de novos métodos de programação e uma mudança nas implementações sequenciais das aplicações existentes. Para projetar aplicações eficientes, é necessário que os programadores conheçam bem a arquitetura do sistema de computação paralelo no qual a aplicação será executada.

Com a evolução dos sistemas de computação e das redes de comunicação, aliado a um barateamento no custo dos computadores pessoais, novas alternativas para a construção de sistemas de computação de alto desempenho surgiram. Uma destas alternativas, a computação distribuída, tem como princípio a utilização de vários recursos de processamento para a execução de uma aplicação, realizando a distribuição da aplicação entre os recursos de processamento. Essa idéia vem sendo largamente utilizada, sendo que entre os sistemas de computação mais poderosos existentes hoje vários são construídos a partir de recursos computacionais distribuídos. A computação em *cluster* [10] é o principal exemplo desse conceito, e é hoje uma alternativa bastante utilizada quando se deseja obter um grande poder computacional a um custo relativamente baixo [39]. *Clusters*, embora não sejam sistemas construídos especificamente para computação paralela, possuem algumas características que os tornam bastante similares a sistemas construídos com fins específicos, por exemplo: podem ser usados de forma dedicada, são bastante estáveis e o gerenciamento pode ser feito de forma centralizada, facilitando a solução de eventuais problemas com os equipamentos.

Embora muitos problemas computacionais possam ser solucionados utilizando-se *clusters*, existem problemas que exigem grande poder computacional onde seriam necessários um grande número de computadores trabalhando conjuntamente para que a solução dos mesmos fosse obtida em um tempo satisfatório. Avanços nas tecnologias de redes de comunicação de longa distância permitiram que computadores localizados não apenas em um mesmo local pudessem ser reunidos, formando um

grande computador paralelo virtual. Assim, a utilização de grandes quantidades de recursos computacionais não especializados, localizados em lugares geograficamente distribuídos e interligados através de meios de comunicação, como a *internet*, tornou-se uma alternativa para a construção de sistemas de alto poder computacional. Sistemas existentes em lares, empresas e universidades são na maioria das vezes sub-utilizados, passando longos períodos com poder de processamento ocioso e também com dispositivos de armazenamento com capacidade ociosa. Essa tecnologia, inicialmente chamada de metacomputação [13] e recentemente Grid [23], mostra-se hoje como uma alternativa interessante para a computação de alto desempenho a um custo relativamente baixo, comparado com o custo dos supercomputadores. Em sistemas Grid, recursos computacionais, desde um simples computador portátil a até mesmo supercomputadores, dispositivos com grande capacidade de armazenamento e equipamentos de coleta de dados, podem ser utilizados de forma conjunta para a solução de problemas. Usuários que precisem de recursos para a solução de um problema, além dos que lhes são disponíveis localmente, podem utilizar recursos localizados em uma outra instituição, geograficamente distante, de forma a reunir poder computacional suficiente.

Em ambientes de computação paralela típicos, como supercomputadores e clusters, os recursos são usualmente homogêneos e a quantidade de recursos requeridos por uma aplicação é normalmente reservada com exclusividade antes da execução para o período total de execução. Já um ambiente Grid possui características diferentes que dificultam sua utilização. A heterogeneidade dos processadores e das redes de interconexão, a distribuição geográfica dos equipamentos, podendo estar a quilômetros de distância uns dos outros e a dinamica da disponibilidade dos mesmos tornam necessários mecanismos que possibilitem o controle e o gerenciamento do ambiente. Informações sobre os recursos, como velocidade de processamento, carga dos sistemas, capacidade de armazenamento, entre outras, são necessárias para que as aplicações obtenham o melhor desempenho possível do sistema. Informações sobre a execução das aplicações precisam ser obtidas para que ajustes na execução possam ser feitos em função de mudanças nas características do ambiente como: entrada e/ou saída de novos recursos no Grid, variações na sobrecargas dos recursos do am-

biente em função da entrada ou saída de aplicações e falhas em recursos previamente alocados para utilização.

As características dos ambientes Grid geraram a necessidade do desenvolvimento de mecanismos de coleta, armazenamento e disponibilização de informações sobre os recursos disponíveis no Grid. Outros mecanismos de monitoramento da execução de aplicações, tolerância a falhas e escalonamento de tarefas também são necessários. No sentido de facilitar o uso de ambientes Grid, vários esforços de desenvolvimento de *middlewares* que tiram dos programadores e usuários a necessidade de realizar tal monitoramento, estão em execução em diversos centros de pesquisa. Projetos de desenvolvimento de ferramentas de gerenciamento de recursos do Grid como, por exemplo, Globus [22] e Sun Grid Engine [26]; de escalonamento de recursos, como AppLeS [4]; ou de criação de ambientes denominados PSE (Problem Solving Enviroments), como Cactus [18]; disponibilizam ferramentas e aplicações que facilitam o uso e controle dos recursos de ambientes Grid. Apesar dos desenvolvimentos alcançados, ferramentas de uso geral que permitam que problemas de naturezas diversas sejam executados no Grid ainda não existem. Muitos dos PSE's são hoje ferramentas específicas para determinadas classes de problemas como, por exemplo, a simulação de fenômenos físicos [3].

Muitas aplicações existentes hoje são escritas para serem executadas em ambientes paralelos tradicionais ou mesmo em *clusters*, segundo características estáticas dos recursos de computação e comunicação dos mesmos. As características dos ambientes Grid são geralmente dinâmicas, diferentemente dos outros sistemas. O estado dos recursos do ambiente sofrem muitas alterações, não oferecendo às aplicações as mesmas condições a cada execução. Para que se possa obter o melhor desempenho possível em ambientes Grid, as aplicações necessitam adaptações ou até mesmo serem reescritas, observando-se as características dos ambientes Grid.

O projeto EasyGrid [9] é um dos esforços da comunidade científica na busca de um ambiente Grid onde aplicações de diversas naturezas, já escritas em linguagem C/C++ e utilizando a biblioteca MPI, possam ser executadas eficientemente em ambientes Grid sem que alterações sejam necessárias. Com isso, a utilização de

ambientes Grid por parte da grande comunidade científica que necessita de grande poder computacional será facilitada, motivando uma rápida utilização dessa tecnologia. O projeto EasyGrid se baseia em um conceito onde grande parte do *middleware* necessário para que aplicações sejam executadas em ambientes Grid está embutido na aplicação. Serviços, como o monitoramento da execução da aplicação, o escalonamento das tarefas da aplicação sobre os recursos do Grid, mecanismos de tolerância a falhas entre outros, estão fortemente relacionados com a aplicação. Dessa forma, as aplicações se tornam independentes da arquitetura do Grid, podendo ser executadas em um número maior de configurações de ambientes Grid. Serviços mais básicos, como informações sobre recursos, gerenciamento de recursos e segurança, ficam a cargo de *middlewares* no nível de sistema (*core middlewares*)como, por exemplo, o Globus Toolkit [21] que é hoje utilizado em diversos projetos de ambientes Grid.

Em ambientes paralelos, a distribuição das tarefas entre as unidades de processamento do sistema é chamada de escalonamento, sendo de extrema importância para que uma aplicação seja executada no menor tempo possível. O escalonamento de uma aplicação influencia diretamente no tempo de execução da mesma, sendo que um escalonamento mal realizado pode levar a um tempo longo de execução. Para que um escalonamento eficiente possa ser realizado é necessário um modelo que identifique as características que mais influenciam o desempenho do ambiente onde a aplicação será executada. Também, quanto mais precisas forem as informações sobre os parâmetros do modelo, melhores chances existirão de que o escalonamento das tarefas da aplicação leve a um bom tempo de execução da aplicação.

Existem diversos modelos de computação paralela. Modelos mais teóricos, como o modelo PRAM [20], que são muito utilizados na análise de programas paralelos, mas são impróprios para vários problemas práticos pois desconsideram muitas características importantes dos ambientes paralelos. Por outro lado, já foram propostos modelos mais realísticos, como o modelo de Latência [37] e o modelo LogP [14, 2], mas que consideram parâmetros que dificultam a construção de escalonadores e de aplicações baseadas nestes modelos. De qualquer modo, em ambientes Grid ainda não se definiu um modelo ideal de computação paralela.

Esta dissertação propõe a utilização de uma variante do modelo LogP para o escalonamento de aplicações MPI em ambientes Grid com o objetivo de melhorar a utilização dos recursos e minimizar o tempo de execução da aplicação. São propostos também, mecanismos eficientes para calibração dos parâmetros do modelo específicos para sistemas Grid. A precisão do modelo e da eficiência da coleta dos parâmetros é verificada em um Grid computacional real. O benefício da utilização do modelo é avaliado pela comparação dos tempos de execução de aplicações escalonadas com base no novo modelo, com os tempos de execução das mesmas aplicações escalonadas com base em abordagens tradicionais.

O restante da dissertação está organizado da seguinte forma: o Capítulo 2 apresenta uma visão geral sobre as características dos ambientes Grid, do Globus Toolkit (um *middleware* básico para ambientes Grid), de algumas ferramentas para coleta de informações em ambientes distribuídos e do projeto EasyGrid, um *framework* para o desenvolvimento e execução de aplicações MPI em ambientes Grid. No Capítulo 3 é feita uma revisão geral sobre modelos de computação paralela e é proposto o modelo HLogP, um novo modelo de escalonamento voltado para ambientes heterogêneos. O Capítulo 4 descreve o desenvolvimento de três estratégias de calibragem dos parâmetros do modelo proposto e analisa os resultados dos testes de verificação. Enquanto os resultados do Capítulo 4 são obtidos em relação ao MPICH-G2, a versão padrão do MPI para ambientes Grid, o Capítulo 5 investiga o comportamento da versão MPI-LAM, uma versão alternativa recentemente adaptada para Grids. Este Capítulo também traz algumas observações importantes sobre o comportamento de processos nas versões LAM e CH-G2 da biblioteca de troca de mensagens MPI e propõe uma versão do modelador analítico para a versão LAM. O Capítulo 6 apresenta os resultados da validação do modelo proposto, comparando o resultado do escalonamento de uma aplicação paralela realizado com base no modelo HLogP com o resultado do escalonamento realizado com base no modelo de latência e no escalonador tradicional do MPI. No capítulo 7 são apresentadas as conclusões e discutidas propostas para trabalhos futuros na área.

Capítulo 2

Contexto e trabalhos relacionados

Este capítulo apresenta uma descrição das funcionalidades de *middlewares* Grid, inclusive do *middleware* básico Globus Toolkit. São apresentados também os mecanismos de coleta de informações sobre recursos de ambientes distribuídos, Network Wather Service (NWS) e Metacomputing Discovery System (MDS) do Globus; e o *framework* EasyGrid, uma ferramenta para execução eficiente de aplicações MPI em ambientes Grid.

2.1 Características de ambientes Grid

Grids são hoje uma alternativa viável para a computação de alto desempenho. Com avanços nas tecnologias de redes de longa distância, onde as velocidades de comunicação aumentaram significativamente, tornou-se possível a interligação de recursos computacionais geograficamente distribuídos sem que os atrasos nas comunicações tornem seu uso proibitivo.

Podemos classificar os ambientes Grid em três categorias em função do tipo de serviço oferecido.

- *Grids computacionais:* Sistemas que têm uma capacidade computacional agregada muito maior do que de qualquer uma das máquinas que fazem parte do

próprio sistema. Dependendo de como as aplicações utilizam essa capacidade agregada, o Grid computacional pode ser projetado para *supercomputação distribuída* ou *computação de alto throughput* (alta vazão). Um Grid para supercomputação distribuída executa aplicações paralelas em diversas máquinas em locais distintos com o objetivo de reduzir o tempo de execução destas aplicações (um único *job*). Por outro lado, um Grid de *alto throughput* almeja aumentar a taxa de finalização de execução de uma série de *jobs*. Aplicações que envolvem o estudo de parâmetros para explorar um conjunto de cenários de projeto como, por exemplo, testes de verificação de projeto de processadores, seriam mais eficientemente executados em um Grid desse tipo;

- *Grids de dados:* Sistemas que fornecem uma infra-estrutura para processar uma grande quantidade de dados. São usados em aplicações que sintetizam novas informações a partir de repositórios de dados, como bibliotecas digitais ou *data warehouses* distribuídas na rede;

- *Grids de serviço:* Sistemas que oferecem serviços que não são fornecidos por uma única máquina do sistema. Por exemplo, nesta classe de Grid se encontram aplicações colaborativas que se preocupam, a princípio, em habilitar e intensificar (em tempo real) interações humanas via um espaço de trabalho virtual.

Uma visão geral dos ambientes Grid pode ser definida em quatro camadas: recursos, *middleware* básico, *middleware* de serviços e aplicações, como apresentado na Figura 2.1. Na camada inferior encontramos os recursos computacionais, como por exemplo: computadores, infraestrutura de comunicação, meios de armazenamento, S.O. locais, entre outros. Na camada superior estão as aplicações dos usuários, que são executadas utilizando os recursos da camada inferior. Nas camadas intermediárias, denominadas *middleware*, temos os serviços que abstraem das aplicações todos os mecanismos necessários para que as mesmas sejam executadas eficientemente sobre os recusos disponíveis. A camada de *middleware* básico é responsável por garantir o acesso distribuído aos recursos do Grid. A camada de middleware de serviços é responsável por oferecer serviços que têm como objetivo

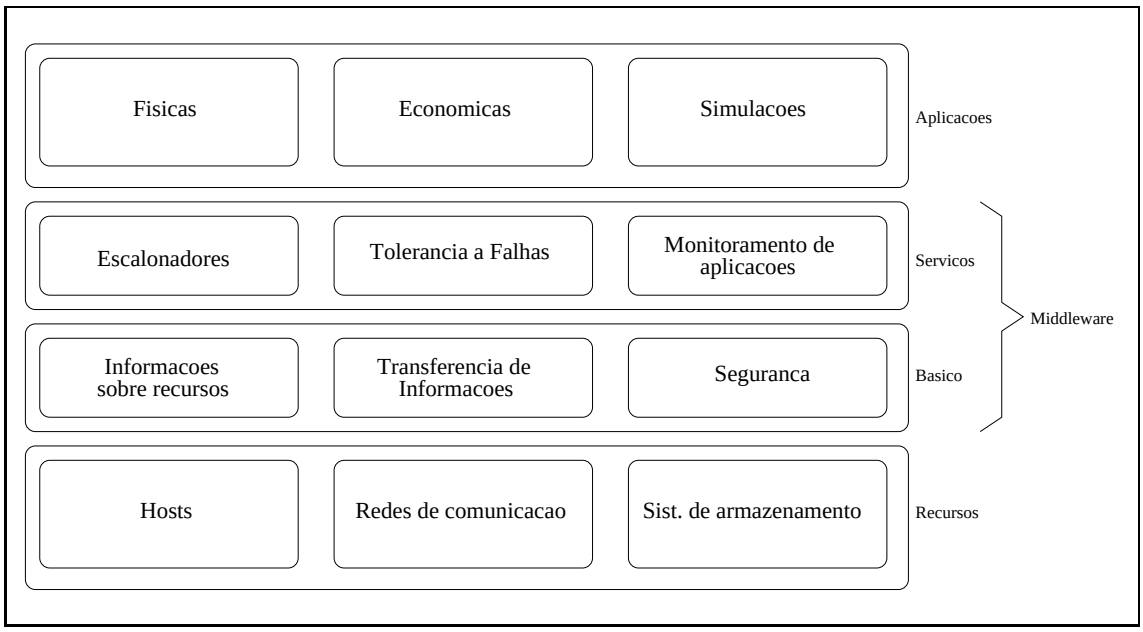


Figura 2.1: Visão geral de um ambiente Grid.

facilitar a execução das aplicações no ambiente Grid.

Atualmente existe um número de projetos de pesquisa para o desenvolvimento de ferramentas e serviços de *middleware* para ambientes Grid [19]. Alguns desenvolvem serviços básicos, como o Globus Toolkit [21], enquanto outros, como Condor [31] e Legion [36], oferecem serviços de gerenciamento de recursos tais como: distribuição de jobs, gerenciamento de carga, entre outros. Existem também os chamados ambientes de solução de problemas, como o Cactus [18] e Nimrod/G [11] que permitem a usuários, não especializados no uso de ambientes Grid, executarem aplicações específicas como, por exemplo, simulações parametrizadas.

Entre os serviços oferecidos pelo *middleware* básico de um ambiente Grid, podemos destacar alguns como, por exemplo:

- *Serviço de informações sobre os recursos do Grid:* Este serviço tem como objetivo principal a centralização das informações sobre os recursos do Grid, facilitando a obtenção de informações atualizadas dos recursos ativos. Os recursos podem ser de fabricantes diferentes e utilizarem sistemas operacionais diferentes, sendo necessário um mecanismo de coleta local que interaja com o sistema, coletando as informações e enviando-as para um gerenciador central

responsável pelo armazenamento. Dessa forma, as informações provenientes de recursos e sistemas operacionais heterogêneos, podem ser obtidas centralizadamente.

- *Segurança e acesso aos recursos:* Outro aspecto heterogêneo dos ambientes Grid é em relação à segurança. Cada organização participante do ambiente implementa políticas próprias de segurança e acesso aos recursos. Não deve ser necessário aos usuários e serviços de um ambiente Grid saber informações de segurança para cada recurso do ambiente. Assim, mecanismos de segurança centralizados são necessários para que o acesso aos recursos seja feito de maneira simplificada, permitindo a autenticação única dos usuários em todos os recursos do Grid.

- *Alocação de recursos:* O uso efetivo de algum recurso em ambientes Grid pode ser algo bastante árduo para o usuário ou outros serviços. As características heterogêneas dos recursos podem levar à necessidade de mecanismos diferentes de acesso aos mesmos. A criação de processos, por exemplo, pode variar dependendo do sistema operacional utilizado. São necessários mecanismos que abstraiam as particularidades de cada recurso dos usuários, fazendo com que a forma de acesso seja única para todos os tipos de recursos do ambiente.

Ambientes Grid são também dinâmicos, ou seja, recursos podem deixar de estar disponíveis ou passarem a ficar disponíveis para o ambiente ao longo do tempo. Um recurso pertencente a uma determinada organização pode ser requisitado por algum usuário da mesma e este ser indisponibilizado ao ambiente Grid por um adminstrador sem que seja necessária a comunicação aos usuários que por ventura o estejam utilizando. Por outro lado, um recurso como, por exemplo, um supercomputador que não estava anteriormente disponível, pode ser disponibilizado em determinado momento, alterando siginificativamente as características do ambiente. As configurações dos recursos do Grid podem também sofrer alterações, como a provocada por uma variação na carga dos mesmos pela execução ou encerramento de outros processos, ou atualização ou troca de bibliotecas de *software*.

Essas características geram a necessidade de mecanismos de ajustes na execução das aplicações. Problemas ocasionados por falhas de recursos durante a execução de aplicações devem ser contornados pelo próprio ambiente, para que não seja necessária a reexecução de toda a aplicação. Após o início da execução de uma aplicação, um novo recurso de maior poder computacional ou com menor carga pode ser disponibilizado, sendo interessante a realocação de parte dos processos da aplicação para este novo recurso, diminuindo-se assim o tempo de execução da aplicação. Alguns dos mecanismos necessários incluem:

- *Tolerância a falhas:* Os recursos de um ambiente Grid estão, na maioria das vezes, sob o controle dos administradores e usuários de cada organização e não sob controle do usuário que executa a aplicação. Em sistemas tradicionais, quem executa as aplicações pode facilmente detectar falhas nos recursos computacionais e tomar as devidas ações para corrigir o problema. Muitas soluções de tolerância a falhas já definidas para ambientes paralelos tradicionais não podem ser aplicadas a ambientes Grid. Novos mecanismos mais complexos que atendam às exigências geradas pelas características dinâmicas e distribuídas dos ambientes Grid devem ser desenvolvidos.

- *Escalonamento estático e dinâmico de tarefas:* O problema de escalonamento de tarefas não é uma área nova surgida com os ambientes Grid. O escalonamento de uma aplicação determina em qual recurso computacional cada tarefa da aplicação será executada tendo como objetivo minimizar o tempo de execução da aplicação. Em ambientes tradicionais, normalmente, somente o escalonamento estático é realizado, uma vez que as características dos recursos não se alteram durante a execução da aplicação. Já em ambientes dinâmicos, como o Grid, durante a execução das aplicações também são necessários mecanismos de re-escalonamento para que ajustes sejam feitos, em função da disponibilidade de poder computacional, de modo a se tentar minimzar o tempo de execução das aplicações.

- *Monitoramento da execução de aplicações:* Tanto os mecanismos de tolerância a falhas e de escalonamento necessitam de informações sobre as aplicações que

estão sendo executadas e sobre os recursos disponíveis, para que suas ações de ajuste possam ser executadas. Mecanismos de monitoramento de execução devem coletar informações sobre a execução das aplicações e disponibilizá-las para que os outros mecanismos as utilizem.

Existem inúmeras iniciativas de construção de sistemas para solução de problemas ou execução de aplicações em ambientes Grid de maneira mais fácil. Algumas soluções propõem serviços que atendem a um único tipo de problema como, por exemplo o Nimrod/G[11] que é um ambiente para simulações parametrizadas de problemas científicos. Outros oferecem apenas ferramentas de escalonamento como, por exemplo, o AppLeS[4]. Ambientes que atendam a qualquer tipo de classe de aplicações, facilitando a execução de aplicações desenvolvidas para ambientes tradicionais, ainda são um desafio. Os ambientes Grid ainda estão longe de poderem ser utilizados por pessoas que não detenham um grande domínio de ferramentas para solução das dificuldades impostas. A popularização do uso de ambientes Grid depende da construção de ambientes onde as características e as dificuldades impostas por esses tipos de ambientes sejam abstraídos para os usuários.

2.2 Globus

O Globus Toolkit versão 2.4.3 [22] é um pacote de serviços básicos para Grid que reúne um conjunto de ferramentas que disponibilizam mecanismos de controle dos recursos do sistema, realizando funções como: coleta e armazenamento de informações sobre os recursos do sistema, mecanismo de segurança com autenticação única e criação de processos. Estes serviços são normalmente utilizados por outras ferramentas para implementação de ambientes de solução de problemas mais complexos. Algumas versões de linguagens para programação paralela como C/C++, Fortran e Java já se encontram disponibilizadas para o desenvolvimento de aplicações para ambientes Grid utilizando os serviços do Globus Toolkit. Uma versão da biblioteca de troca de mensagens MPI[35] denominada MPICH-G2[33] utiliza os serviços de comunicação do Globus Toolkit para implementar a comunicação entre

os processos.

O pacote de serviços Globus Toolkit é composto principalmente pelos seguintes módulos: o módulo MDS, o módulo GRAM e o módulo GIS. O módulo GIS[16] (Grid Security Infrastructure) oferece serviços para garantir a segurança do sistema através de um mecanismo de autenticação global. O sistema é baseado na infraestrutura pública de chaves (Public Key Infrastructure - PKI) para fornecer identificação única para todos os recursos do Grid (*hosts*, usuários, etc), sendo que cada recurso possui um certificado próprio. Quando um recurso necessita utilizar outro recurso, ambos precisam se autenticar, isto é, os dois precisam verificar a identidade um do outro antes que a operação possa ser executada. Os certificados são baseados no formato de certificação X.509 e nos protocolos de comunicação SSL/TLS (Secure Socket Layer / Transport Layer Security), seguindo as especificações da API GSS (Generic Security Service). Quando um usuário deseja executar um *job* em um *host* do ambiente Grid, ele precisa criar um *proxy* de usuário, que é um processo responsável pela autenticação com todos os recursos a serem utilizados na execução de *jobs*. É necessário também que o usuário tenha autorização para executar *jobs* no recurso. A autorização é dada pelo administrador do site onde se encontra o recurso, através de um arquivo chamado *gridmapfile*, armazenado no próprio recurso. O arquivo identifica todos os usuários do Grid que têm acesso autorizado ao recurso. Ele relaciona cada usuário global do Grid a uma conta local que define as permissões do usuário global no recurso.

O módulo GRAM[17] (Grid Resource Alocation Manager) do Globus Toolkit, é responsável pela alocação de recursos em um ambiente Grid. Neste caso, devido à heterogeneidade dos recursos do ambiente, esta tarefa se torna complicada, pois para cada recurso diferente do ambiente existem mecanismos próprios de acesso. O Globus Toolkit define uma linguagem padrão de acesso a recursos, chamada RSL (Resource Specification Language), através da qual o usuário especifica os recursos que deseja ter acesso. O módulo GRAM é responsável por receber um pedido de acesso a um recurso, especificado na linguagem RSL, e então mapeá-lo para algum mecanismo local de alocação como, por exemplo, Fork ou LoadLeveler. A API módulo GRAM possui também funções para cancelamento e monitoramento de *jobs*

em execução.

O Globus Toolkit oferece também serviços como o GSIFTP e GSISSH. O serviço GSIFTP permite que usuários do Grid que possuam um certificado reconhecido façam a transferência de arquivos de um recurso para outro do Grid. O serviço GSISSH permite que os usuários do Grid acessem remotamente os recursos do Grid.

2.3 Mecanismos de coleta de informações de sistemas

Em ambientes Grid, assim como em qualquer sistema de computação, sempre se busca atingir o melhor desempenho possível dos sistemas. Administradores, programadores, usuários, escalonadores de processos, escalonadores de aplicações e mecanismos de tolerância a falhas, entre outros, necessitam tomar decisões de forma a se obter sempre o melhor desempenho possível. Portanto, são necessárias informações sobre os sistemas como, por exemplo, espaço livre em disco, quantidade de memória disponível, carga dos processadores, tempo de comunicação entre equipamentos, entre outras, para que se possa tomar medidas com o intuito de maximizar o desempenho e minimizar o uso dos recursos dos sistemas.

Existem diversas formas de se obter informações sobre as características e o desempenho de sistemas computacionais. Mecanismos mais simples são disponibilizados pela maioria das linguagens de programação. Em *C*, por exemplo, podemos utilizar a função *gethostname()* para se obter o nome do *host* onde uma determinada aplicação é executada. Outras informações também podem ser obtidas a partir do sistema operacional da máquina, como no caso do sistema operacional *LINUX*, onde informações sobre a carga do sistema, quantidade de memória disponível, quantidade de processos em execução, entre outras podem ser obtidas a partir de arquivos (*/proc/loadavg*) que são gerados pelo próprio sistema operacional.

Embora muitas informações possam ser obtidas de forma simples, outras, como é o caso das referentes a sistemas de comunicação não são disponibilizadas

pelos sistemas básicos. Outra questão importante é a heterogeneidade dos sistemas que dificulta o acesso a informações, sendo necessários mecanismos e procedimentos diferenciados para se obter informações iguais de sistemas diferentes. Nesse sentido foram propostos mecanismos de coleta e distribuição de informações sobre recursos computacionais mais complexos como, por exemplo, o NWS[44] e o MDS[16] do Globus Toolkit, que coletam e fornecem informações sobre os recursos de sistemas distribuídos heterogêneos. A seguir, um resumo sobre esses mecanismos (sistemas) é apresentado, destacando-se suas principais características.

2.3.1 Globus Metacomputing Discovery System (MDS)

O módulo MDS[16] é responsável pelo serviço de informações sobre os recursos disponíveis no sistema. As informações sobre os recursos são armazenadas em um banco de dados distribuído baseado no protocolo aberto de acesso a dados (*openLDAP*). O MDS está estruturado de forma hierárquica, sendo composto por provedores de informações (IPs) que rodam em cada recurso do Grid e fornecem informações específicas sobre os recursos. As informações coletadas são enviadas para um centralizador em cada recurso, denominado Grid Resource Information Service (GRIS), que armazena estas informações. Uma terceira estrutura denominada Grid Index Information Service (GIIS) recebe informações resumidas dos GRIS centralizando informações sobre todos os recursos disponíveis no Grid.

As informações disponibilizadas pelo MDS, mostradas na Figura 2.2 são, em sua maioria, estáticas como: quantidade total de memória, velocidade do processador, sistema operacional instalado, entre outras. Algumas informações dinâmicas como, a carga dos *hosts* e quantidade de memória livre são também disponibilizadas (tipicamente informações que podem ser obtidas facilmente através de chamadas do sistema). Além do sistema MDS oferecer poucas informações dinâmicas, essas não retratam de maneira eficiente a característica dinâmica dos ambientes Grid. Características de implementação do sistema, como o protocolo utilizado, que possui uma latência significante [24], e também o período entre as atualizações das informações por parte dos IPs (*Information Providers*), que é feita por padrão a cada 20 minutos,

tornam as informações disponibilizadas pelo sistema MDS pouco eficientes para o processo de escalonamento de aplicações.

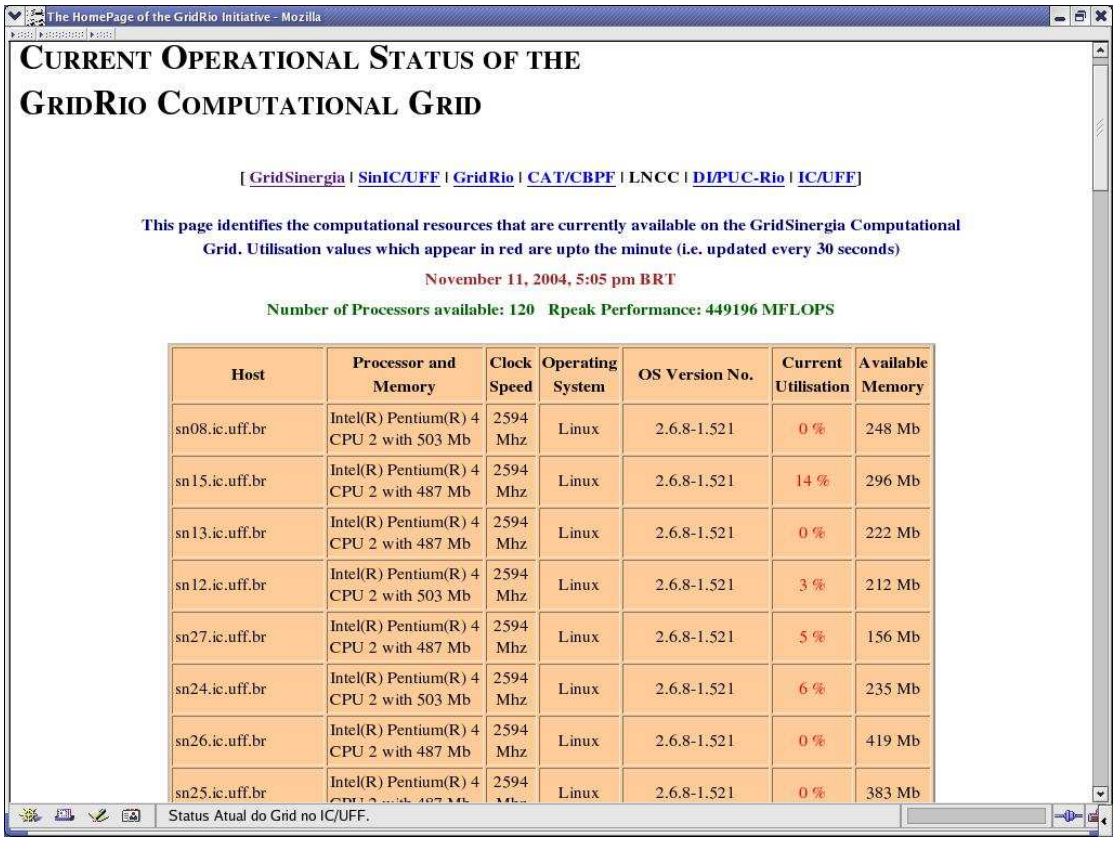


Figura 2.2: Dados fornecidos pelo MDS para o Grid computacional GridRio.

2.3.2 Network Weather Service (NWS)

O NWS [44] é um sistema que fornece informações sobre recursos distribuídos referentes ao percentual da capacidade de processamento disponível nos *hosts* e com relação à capacidade de comunicação entre cada par de *hosts*, principalmente para utilização por escalonadores de aplicações. O NWS implementa um mecanismo de coleta e de previsão futura das informações sobre os recursos. A implementação segue uma estrutura modular (Figura 2.3) onde se tem um módulo sensorial que monitora os recursos coletando informações sobre a performance que os recursos podem disponibilizar para as aplicações, um módulo de previsão que faz a previsão das condições futuras dos recursos e um módulo divulgação que dissemina as informações em vários formatos.

O módulo sensorial é composto por um conjunto de sensores que rodam nos *hosts* do ambiente distribuído e são responsáveis pela coleta das informações sobre os recursos pertencentes ao ambiente. Os sensores são classificados em passivos e ativos. Os passivos coletam informações executando utilitários externos como por exemplo o comando *vmstat* do linux e processando sua saída obtendo as informações como a carga da CPU e a quantidade de memória disponível. Já os sensores ativos medem explicitamente a disponibilidade do(s) recurso(s) que monitora(m). A medida é realizada por experimentos que verificam a performance dos recursos, como por exemplo, a medida da capacidade de comunicação disponível entre dois pares de *hosts* do sistema é desempenhada por um programa que realiza a comunicação entre os *hosts* e obtem as informações.

O módulo de previsão usa vinte e quatro algoritmos para tentar antecipar flutuações na performance dos recursos. Os algoritmos utilizam os dados coletados pelos sensores e, através de processos estatísticos, fornecem um valor estimado para a disponibilidade do recurso no futuro. Alguns dos algoritmos implementados fornecem informações baseadas na média dos valores medidos, no último valor medido, entre outros. A seleção de qual algoritmo é utilizado na determinação de um valor para a disponibilidade do recurso, é feita com base na precisão de cada um dos algoritmos. Para a determinação da precisão dos algoritmos, quando um novo valor é coletado pelo módulo sensorial, seu valor é comparado com os valores obtidos através dos algoritmos. O algoritmo que fornecer um valor mais próximo do valor coletado, mais preciso, é então utilizado nas previsões futuras.

As informações são disponibilizadas via interface na World Wide Web. Uma das preocupações do sistema de informações do NWS é com relação a possíveis problemas de performance por conta de sobrecargas. Os dados são armazenados de forma distribuída em cada *host* que armazena uma cópia do último estado da rede e uma previsão para um passo à frente.

Embora o sistema NWS tenha a preocupação de coletar as informações relativas à rede de comunicação que interliga os *hosts* utilizando mecanismos o mais próximo possível das aplicações, ainda assim não são utilizados os mecanismos im-

plementados pelas bibliotecas de funções de comunicação utilizadas pelas aplicações. O ideal do ponto de vista de escalonamento é que se utilize para a coleta de informações do sistema os mesmos mecanismos qua a aplicação utiliza para comunicação. Uma outra característica do sistema NWS é que apenas informações de latência e *throughput* da rede são coletadas.

2.4 EasyGrid

As caracteristicas dos ambientes Grid tornaram essa área de pesquisa um novo desafio para o desenvolvimento de ferramentas e aplicações. Ferramentas e técnicas utilizadas em ambientes paralelos tradicionais não atendem às novas características impostas, sendo necessário o desenvolvimento de novas ferramentas, técnicas e modelos computacionais que capturem as características dos ambientes Grid.

O sucesso de qualquer sistema depende, principalmente, de sua facilidade de uso, custo satisfatório, segurança e confiabilidade. Acredita-se que para o sucesso dos ambientes Grid, mecanismos que popularizem e tornem fácil o seu uso são necessários, permitindo que o custo de denvolvimento de aplicações que possam usufruir de todo o poder oferecido não se torne proibitivo. Mecanismos que garantam a segurança e a confiabilidade do Grid também são necessários, devendo abstrair dos usuários todas as dificuldades impostas pelas características do Grid.

O projeto EasyGrid [9] propõe a construção de um *framework* de desenvolvimento de aplicações paralelas para ambientes Grid que seja portável (independente de um *middleware* específico), de fácil utilização, eficiente e com baixa intrusão. A principal característica desse *framework* é a capacidade de, a partir de uma aplicação paralela, gerar automaticamente uma aplicação capaz, por si só, de ser executada eficientemente em Grids computacionais, livrando o programador desta tarefa difícil.

As aplicações interessadas não são apenas aquelas perfeitamente paralelas ou com características interessantes aos ambientes Grid como granularidade grossa (pouca comunicação). Aplicações práticas e de cunho geral tanto científicas como

também comerciais são alvos principais do *framework*, aumentando sua abrangência de uso. São consideradas aplicações desenvolvidas tendo como base a plataforma de troca de mensagens e, em particular, programas C/C++ utilizando a interface de troca de mensagens MPI [35], por ser hoje a solução mais comum no desenvolvimento de aplicações científicas.

A facilidade de uso do *framework* advém do fato de que o programador poderá concentrar seus esforços somente na exploração do paralelismo da aplicação. A geração da aplicação e a execução eficiente da mesma utilizando os recursos disponíveis no Grid ficam a cargo do *framework*.

A maioria dos projetos de *middlewares* adotam uma visão centrada nos recursos, Figura 2.4(a), onde um Sistema de Gerenciamento de Recursos (SGR) tem como um de seus objetivos garantir uma utilização eficiente dos recursos baseando-se no monitoramento e análise de informações específicas do sistema. Em ambientes Grid, devido a suas características, uma abordagem onde os ajustes possam ser feitos em cada aplicação, baseando-se em informações específicas de cada uma delas, pode ser mais interessante. O *framework* EasyGrid, adota uma visão de *middleware* centrada na aplicação, Figura 2.4(b), onde o *middleware* oferece serviços e informações orientadas a cada aplicação individualmente. A utilização eficiente dos recursos é obtida de forma distribuída através da concorrência entre as aplicações pelos recursos disponíveis.

O projeto EasyGrid se concentra no estudo e implementação de ferramentas de escalonamento, tolerância a falhas e gerenciamento de aplicações a serem executadas em ambientes Grid, levando em consideração as características particulares desses ambientes. Cada aplicação EasyGrid traz embutido em seu código os serviços de gerenciamento da aplicação necessários para que a execução seja realizada da forma mais eficiente possível.

O problema de escalonamento de tarefas em sistemas paralelos conciste na execução eficiente de uma aplicação, ou seja, executá-la num menor tempo possível. Em um ambiente paralelo baseado em troca de mensagens, além dos custos de computação de cada tarefa, os atrasos decorrentes do envio e recebimento de mensagens

entre cada tarefa também devem ser levados em consideração. Assim, modelos de computação que capturem todas as características do sistema alvo devem ser assumidos para que o escalonamento produzido seja eficiente. O problema de escalonar tarefas em ambientes paralelos é considerado NP-completo, sendo necessária a adoção de heurísticas para que escalonamentos quase ótimos sejam determinados a um tempo satisfatório. O projeto EasyGrid tem como um de seus objetivos o desenvolvimento de heurísticas e ferramentas de escalonamento de aplicações para ambientes Grid, o que é uma tarefa ainda mais complicada. O comportamento de aplicações reais não pode muitas vezes ser determinado estaticamente, gerando a necessidade de mecanismos que possam fazer ajustes na execução da aplicação dinamicamente. Assim, o problema de escalonamento de tarefas no ambiente EasyGrid é tratado de duas maneiras. A Primeira é o mecanismo de escalonamento estático, que visa determinar o melhor escalonamento possível para um conjunto inicial de recursos do Grid. A segunda é o mecanismo de escalonamento dinâmico que, tem como objetivo promover ajustes no escalonamento da aplicação durante a execução, baseando-se em informações iniciais produzidas pelo escalonamento estático e em informações dinâmicas sobre o comportamento da execução da aplicação.

Além das funcionalidades básicas de escalonamento, o *framework* EasyGrid necessita também de um conjunto de outras funcionalidades, como: serviços de modelagem do sistema alvo, monitoramento da execução da aplicação, entre outras. Na Figura 2.5, apresentamos os passos de criação de uma aplicação EasyGrid a partir de uma aplicação do usuário. O usuário pode executar tanto programas baseados em GADs (Gráfico Acíclico Direcionado) de modelos de programas paralelos teóricos como, diamante, forks, joins entre outros, usados para testes de escalonamento; como também de aplicações reais MPI, para as quais a representação GAD da mesma é construída (por enquanto ainda manualmente). No caso de GADs (Gráficos Acíclicos Direcionados) teóricos, o configurador EasyGrid gera um código MPI; já no caso de uma aplicação do usuário, o configurador gera o GAD referente à mesma. Além da representação GAD da aplicação, o configurador EasyGrid gera um modelo da arquitetura do sistema Grid onde a aplicação será executada que é passado junto com o GAD para o escalonador estático. Este escalonador gera um arquivo RSL para

alocação dos processos da aplicação aos recuros do Grid e fornece informações para um pré-compilador que re-estrutura o programa, por exemplo, replicando processos. O ambiente EasyGrid permite também que o usuário execute a aplicação no Grid sem que seja realizado um escalonamento. Neste caso a distribuição das tarefas da aplicação é realizada pelo próprio MPI de forma round-robin sem se preocupar com a eficiência da execução. Para que a aplicação seja executada de maneira mais eficiente, é necessário que se faça um escalonamento da mesma; operação que não pode ser realizada de maneira eficiente sem que se tenha um modelo do sistema alvo.

2.4.1 Portal EasyGrid

A utilização de ambientes Grid não é simples. Tipicamente o usuário deve escolher os recursos apropriados nos quais a aplicação será executada, e realizar a distribuição das tarefas nos mesmos. Para isso, ele deve conhecer os serviços e os comandos disponibilizados pelo ambiente Grid para a realização das etapas necessárias à execução da aplicação. Uma maneira de simplificar para o usuário a utilização de ambientes Grid, é a criação de interfaces gráficas, chamadas portais, através das quais os usuários interagem mais facilmente. Um portal disponibiliza ao usuário os serviços do ambiente Grid, permitindo que os mesmos submetam tarefas, transfiram arquivos e obtenham informações sobre o ambiente.

Como parte deste trabalho, foi implementado um conjunto de *scripts* e programas para facilitar a escolha de recursos e a execução de aplicações MPI em ambientes Grid, baseados no Globus Toolkit. Os *scripts* e programas desenvolvidos são utilizados pelo Portal EasyGrid, Figura 2.6, que é uma ferramenta para a criação de aplicações *System-aware* habilitadas a ambientes Grid. Nesta seção é apresentada uma visão geral desses *scripts* e programas e também do portal EasyGrid.

Inicialmente, tem-se como "ponto de partida" o uso de um arquivo (chamado *GRUniverse*) de configuração que contém as seguintes informações:

- Número de *hosts* do Grid : Número total de *hosts* do Grid;

- Informações sobre os *sites* : *Domain name* padrão do site, número de *hosts* do *site* e se possui sistema de arquivos NFS;
- Informações sobre os *hosts* : *Domain name* do *host*, se tem ou não instalada a versão LAM do MPI, se tem ou não instalada a versão CH-G2 do MPI e por último, se o *host* é ou não servidor de arquivos;

A manutenção do arquivo de configuração do Grid é de responsabilidade do administrador do Grid. Todo novo site que venha a participar do Grid deve informar ao adminstrador os *hosts* que estarão disponíveis, bem como as versões da biblioteca MPI instaladas e se o *host* é ou não servidor de arquivos. É importante lembrar que cada recurso que venha a fazer parte do Grid precisa de um certificado que seja reconhecido pelo Grid. Este certificado é emitido por uma autoridade certificadora pelo administrador do Grid.

Para que o usuário execute uma aplicação no Grid ele deve informar a quais *hosts* ele tem acesso. Para isso, o portal apresenta ao usuário todos os *hosts* do Grid a partir do arquivo *GRUniverse* e aguarda que o usuário determine qual é o conjunto de *hosts* a que tem acesso. A determinação dos *hosts* do usuário pode ser feita através da especificação de um arquivo, *GRMyUniverse*, que contém a lista de *hosts* a que ele tem acesso; ou escolhendo os recursos da lista apresentada. Nesse instante o usuário pode, se quiser, atualizar seu arquivo pessoal de configuração *GRMyUniverse*.

Depois de obter a lista de *hosts* candidatos, escolhida pelo usuário, é necessário determinar quais destes estão ativos no Grid. Para isso, o portal consulta os serviços de diretório do Grid (MDS) coletando informações sobre todos os *hosts* do Grid que estão ativos recentemente (20 minutos na configuração padrão do MDS). Essa consulta pode produzir informações que não são corretas, ou por estarem desatualizadas (em função do tempo de atualização) ou por falhas no sistema de coleta das mesmas, sendo que às vezes um *host* não aparece no MDS mas está ativo. Por esses motivos, para cada *host* da lista de candidatos, a ferramenta verifica se o mesmo responde ao comando *ping* e também verifica se o Globus gatekeeper (processo *daemon* do GRAM que roda em cada *host* para permitir que aplicações externas

possam ser executadas) está ativo. Caso esta última verificação seja negativa, o *host* é eliminado da lista de *hosts* candidatos.

Com as informações sobre os recursos (velocidade da cpu, carga da cpu, memória, memória disponível, versão do sistema operacional), o portal permite ao usuário excluir algum *host* da lista. Os recursos da lista resultante formam o "Grid virtual" a ser modelado.

A modelagem é realizada através de uma aplicação paralela, a ser descrita no Capítulo 4, composta de n processos que são executados em cada um dos n *hosts* pertencentes ao subconjunto de *hosts* do Grid definido pelo usuário. Durante a modelagem, são coletadas informações sobre as características dos recursos do "Grid virtual" que mais influenciam a execução de aplicações paralelas MPI em ambientes Grid. Os valores modelados podem ser utilizados por um escalonador para determinar a alocação inicial dos processos nos recursos disponíveis ou até mesmo realizar ajustes, redistribuindo os processos ao longo da execução da aplicação.

Para execução da aplicação MPI, após a mesma ser compilada, o portal verifica no escalonamento estático inicial para quais *hosts* o código binário da aplicação deverá ser copiado. Para cada *host* determinado, o comando *globus-url-copy* é executado, copiando o código binário da aplicação no destino (*host*). Após a cópia dos códigos binários, o arquivo contendo o código RSL (Resource Specification Language) é utilizado pelo portal, através do comando *mpirun*, para realizar a execução da aplicação.

2.5 Resumo do Capítulo

Este capítulo fez uma abordagem das principais características dos ambientes Grid e do pacote de serviços básicos Globus Toolkit, que fornece alguns serviços essenciais para ambientes Grid. Foram apresentados também, dois mecanismos de coleta de informações sobre recursos computacionais, o NWS e o MDS do Globus Toolkit, destacando-se as características e o mecanismo de funcionamento de cada. O *framework* EasyGrid foi apresentado como uma proposta de um ambiente para o

desenvolvimento de aplicações que tem entre seus objetivos facilitar a execução de aplicações em ambientes Grid. Por fim, apresentamos o Portal EasyGrid que reune e disponibiliza o conjunto de serviços do *framework* EasyGrid aos usuários.

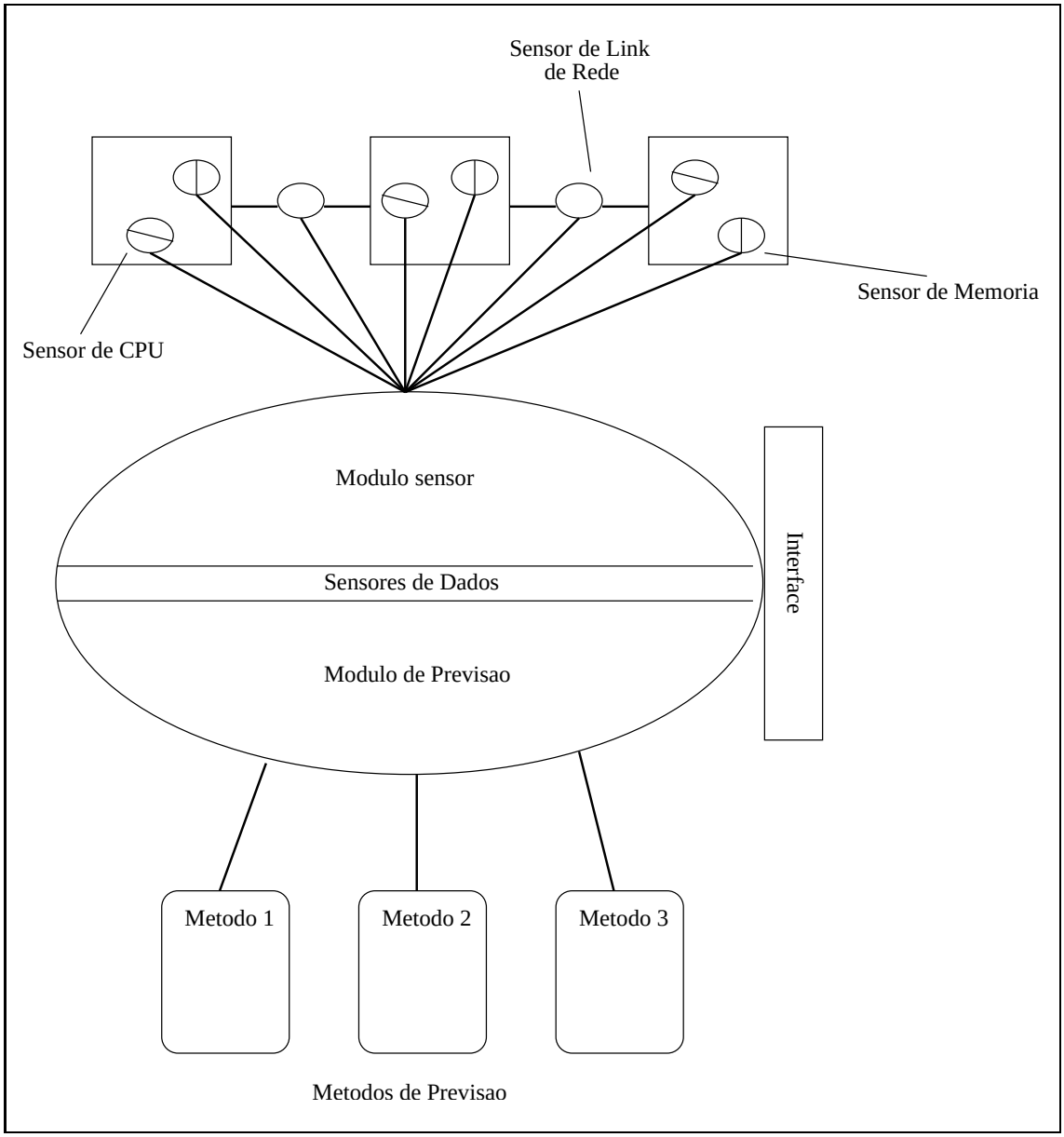


Figura 2.3: Estrutura lógica do NWS.

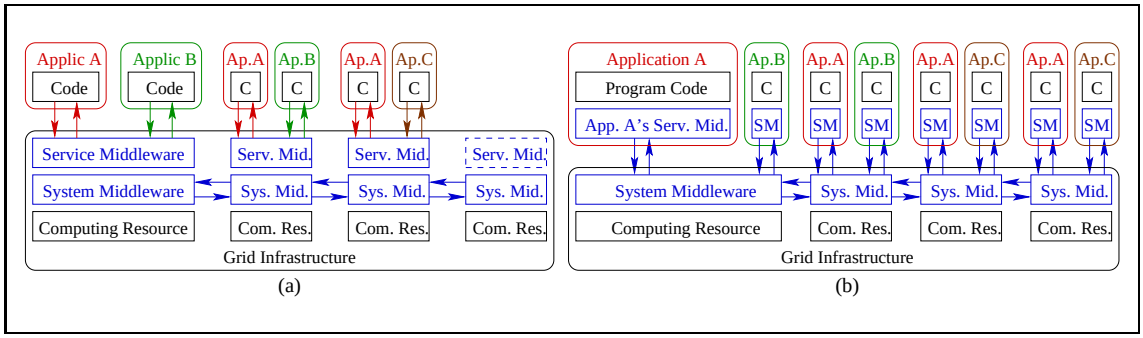


Figura 2.4: (a) Abordagem centrada no sistema e (b) Abordagem centrada na aplicação.

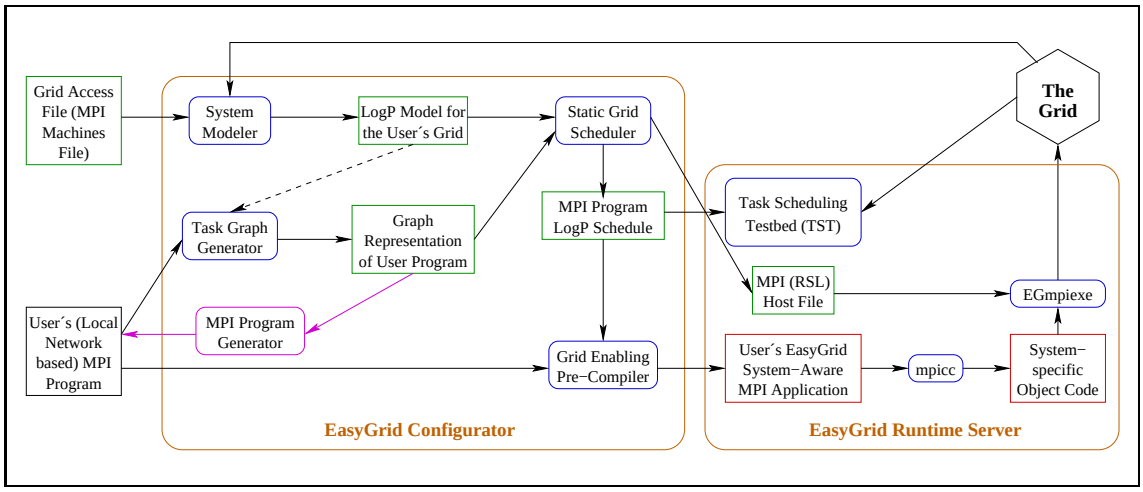


Figura 2.5: Visão geral do ambiente EasyGrid.

Figura 2.6: Tela do Portal EasyGrid, referente à etapa de modelagem da arquitetura do ambiente Grid.

Capítulo 3

Modelos Arquiteturais

Em computação paralela, o uso de modelos para o escalonamento de aplicações é de extrema importância, tendo implicações no desempenho do sistema como um todo (aplicações e recursos computacionais). Este capítulo apresenta um resumo dos principais modelos arquiteturais propostos e apresenta um novo modelo arquitetural voltado para o escalonamento de aplicações paralelas em ambientes Grid.

3.1 Introdução

Uma aplicação paralela consiste de um conjunto de tarefas que podem ser executadas em várias unidades de processamento, trocando informações entre si. Para que uma aplicação paralela obtenha um bom desempenho em sua execução, do ponto de vista do tempo gasto para ser executada, além do algoritmo ter que ser bem desenvolvido, ou seja bem paralelizado, a alocação das tarefas da aplicação nas unidades de processamento disponíveis também deve ser realizada da melhor maneira possível. Para que essa distribuição, chamada escalonamento da aplicação, possa ser realizada, um modelo da arquitetura da máquina alvo deve ser adotado e os parâmetros do modelo referentes à máquina conhecidos, permitindo que a melhor distribuição das tarefas possa ser determinada de forma mais precisa possível.

Um modelo arquitetural identifica quais características do *hardware*, como

por exemplo tempo de acesso à memória das unidades de processamento e latência na rede de interconexão, devem ser consideradas no desenvolvimento de aplicações. Para a execução eficiente de aplicações em ambientes paralelos, é necessário que o desenvolvedor utilize um modelo que abstraia com o máximo de precisão possível as características da arquitetura alvo que influenciarão a execução da aplicação. A utilização de um modelo inadequado por parte do desenvolvedor da aplicação pode levar a resultados indesejáveis quando a aplicação for executada.

Na busca da construção de uma máquina paralela ideal, diversas soluções arquiteturais têm sido adotadas, diferindo umas das outras em aspectos como mecanismos de memória ou tipo de interconexão entre os processadores. Em função dessa diversidade de arquiteturas existentes, vários modelos de computação paralela também têm sido propostos, sendo às vezes específicos para uma determinada arquitetura, considerando particularidades que tornam a portabilidade de aplicações desenvolvidas, tendo como base o modelo, uma tarefa não trivial.

Dentre os modelos propostos, existem os mais teóricos, como o modelo PRAM [20], cujo objetivo é o estudo da paralelização de problemas, permitindo ao projetista abstrair a arquitetura do sistema. Devido à simplicidade desse modelo, aspectos relevantes da maioria dos sistemas paralelos são desconsiderados, tornando inadequada a sua utilização prática para o escalonamento real de aplicações. Na tentativa de representar melhor as características das diversas máquinas paralelas atuais, modelos mais realísticos, que consideram de forma mais detalhada as caracaterísticas arquiteturais, foram propostos [2]. A utilização desses modelos no desenvolvimento de aplicações leva, na maioria das vezes, a melhores resultados de execução das aplicações em máquinas atuais. Embora se obtenha resultados melhores adotando esses modelos, sua maior complexidade leva a um aumento das dificuldades no desenvolvimento de aplicações tanto no que se refere à escrita como no escalonamento na máquina alvo. A seguir, os principais modelos arquiteturais existentes são apresentados de forma resumida, destacando-se as principais características de cada um e também as diferenças existentes de um para o outro.

3.2 Modelo PRAM

O modelo PRAM [20] foi proposto por S. Fortune e J. Wyllie, e tornou-se extremamente conhecido pela sua simplicidade de uso. Seu principal objetivo é ser um modelo simples, facilitando o estudo e o desenvolvimento de algoritmos paralelos permitindo que o projetista da aplicação se concentre na paralelização do problema. Esse modelo considera que o número de processadores é ilimitado e que os mesmos operam de forma sincronizada. O modelo considera que cada unidade de processamento é igual e possui uma memória local, onde residem os dados a serem processados e o código do programa. A comunicação entre os processadores é feita através de uma memória global compartilhada, o que leva a desconsideração do custo de comunicação entre os processadores.

3.3 Modelo de Latência

Com a crescente utilização de máquinas paralelas com memória distribuída, surgiu a necessidade de criação de modelos que representassem de maneira mais precisa a comunicação. Um dos modelos propostos foi o de latência [37] que utiliza um único parâmetro, denominado latência, para representar o custo, ou atraso, da transferência de dados. Neste modelo, o parâmetro é considerado fixo e constante para mensagens enviadas entre quaisquer pares de processadores. O modelo considera também que um processador pode enviar várias mensagens distintas para destinos diferentes simultaneamente, conceito chamado *multicast*.

3.4 Modelo BSP

O modelo Bulk-Synchronous Parallel (BSP) [40, 42] de Valiant, tem como proposta ser um modelo que atenda tanto às questões teóricas quanto as práticas. Esse modelo considera que os algoritmos paralelos são organizados em uma seqüência de superpassos paralelos. Cada superpasso é composto de três fases. Na

primeira, um conjunto de computações locais e independentes são executadas em cada processador. Na segunda fase, todos os processadores que necessitam, realizam comunicação enviando ou recebendo informações para/de outros processadores. A terceira e última fase é uma sincronização entre os processadores para que um novo superpasso seja executado.

3.5 Modelo HBSP

O modelo BSP, descrito anteriormente, considera que todas as unidades de processamento são iguais, ou seja, possuem a mesma capacidade de processamento. O modelo HBSP é uma generalização do modelo BSP proposta por L. Tiffani *et al* [43] que incorpora parâmetros que refletem, de maneira relativa, as diferenças entre as unidades de processamento.

3.6 Modelo LogP

O modelo LogP [14], foi proposto tendo como principal motivação a adoção crescente de sistemas de computação paralela baseados em computadores completos (processador, memória e periféricos) interconectados por uma rede de comunicação. Esse modelo considera que as unidades de processamento são independentes e se comunicam através de troca de mensagens. A comunicação neste caso acontece de forma não coordenada, diferentemente do modelo BSP; ou seja, cada unidade de processamento realiza sua comunicação independentemente das outras. Os parâmetros do modelo são os seguintes:

Latency: Um limite superior da latência ou atraso no envio de uma mensagem contendo um número pequeno de bytes.

overhead: Representa o tempo que o processador leva para enviar e receber uma mensagem, sendo que durante este tempo o mesmo não pode realizar outra tarefa.

gap: É o intervalo de tempo mínimo entre envios ou recebimentos consecutivos.

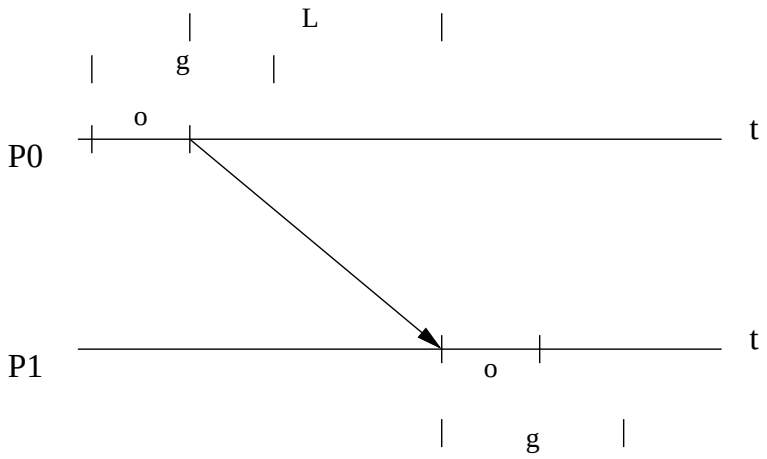


Figura 3.1: Comunicação segundo modelo LogP.

Processors: O número de processadores disponíveis.

Os parâmetros **L, o** e **g** do modelo especificam as características de performance da rede de interconexão sem se ater às características estruturais da rede como por exemplo: a topologia, o meio de propagação ou os protocolos, entre outros. A Figura 3.1 mostra os parâmetros de comunicação do modelo considerando a comunicação do processo *P0* para o *P1*. O processo *P0* inicia a preparação da mensagem a ser enviada no tempo 0. Após um tempo **o** a mensagem é enviada para *P1*, sendo que o primeiro byte da mensagem chega no tempo $o + L$, e a mensagem fica disponível para o processo *P1* no tempo $o+L+o$. O processo *P0* fica disponível para o envio de uma próxima mensagem somente após o tempo **g**.

O modelo LogP representa bem sistemas paralelos formados por uma coleção de computadores conectados por redes de comunicação, como visto por Culler *et al* [14], que são bastante utilizados hoje em dia.

3.7 Modelo LogGP

O modelo LogP modela, com boa precisão, somente comunicações nas quais o tamanho de mensagens seja pequeno, inferiores a um valor predeterminado (ω). Neste caso, para mensagens de tamanho maior do que ω, a mesma tem que ser

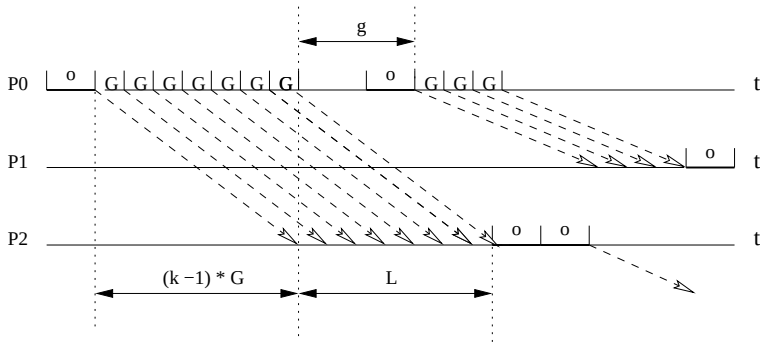


Figura 3.2: Comunicação segundo modelo LogGP.

subdivida em mensagens de tamanho no máximo igual a ω. O modelo LogGP [2] é uma extensão simples do modelo LogP onde o tamanho da mensagem não precisa ser pequeno, tanto mensagens de tamanhos menores ou iguais a ω, como mensagens maiores são tratadas pelo modelo. Para mensagens de tamanho menor do que ω, o modelo LogGP faz o mesmo tratamento feito pelo modelo LogP, ou seja, o envio de uma mensagem entre dois processos leva $o + L + o$. Para mensagens maiores do que ω, o modelo considera um outro parâmetro denominado **G** que representa o tempo necessário para o envio de cada byte da mensagem. Como visto na Figura 3.2, o tempo necessário para o envio de uma mensagem grande (maior que ω), segundo o modelo LogGP é $o + (k-1) * G + L + o$, onde k é o número de bytes da mensagem.

3.8 Modelo LogGPS

Embora o modelo LogGP trate mensagens de tamanhos longos (maiores que ω), Al-Tawil and Moritz [1] observaram que um processo de sincronização ocorre antes do envio de mensagens longas, quando se utiliza bibliotecas de comunicação de alto nível. O modelo proposto por Fumihiko Ino *et al*, denominado LogGPS [27], tem como objetivo tratar a sincronização entre os processos envolvidos na transmissão de mensagens grandes. O modelo considera dois parâmetros extra (**s** e **S**) que se referem respectivamente ao comprimento a partir do qual a mensagem é quebrada em pacotes e ao tamanho de mensagem a partir do qual a sincronização ocorre.

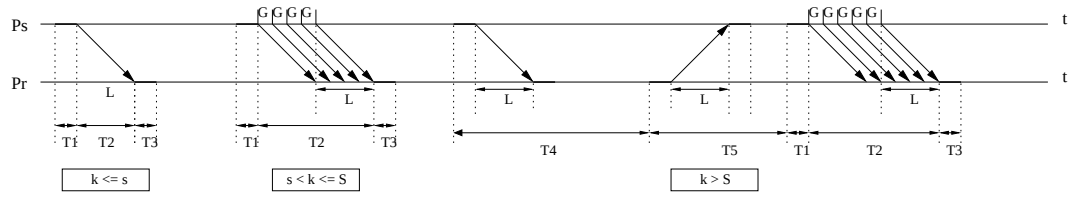


Figura 3.3: Comunicação segundo modelo LogGPS.

Na Figura 3.3, podemos verificar que o modelo LogGPS trata a comunicação para mensagens de tamanho $k \leq s$ como no modelo LogP. Para mensagens de tamanho $s < k \leq S$ a comunicação é tratada como no modelo LogGP. E, para mensagens de tamanho $k > S$, o modelo considera que ocorre uma sincronização entre o processo que envia e o processo que recebe a mensagem, antes do envio. Assim, o tempo de comunicação neste caso é dado por: $T1 + T2 + T3 + T4 + T5$, onde $T4 + T5$ representam o tempo gasto pela sincronização.

3.9 Modelo realístico de comunicação para computação em cluster

Este modelo, proposto por A. Tam and C. Wang [41], trata os parâmetros de comunicação como funções do tamanho da mensagem, considerando três fases no processo de comunicação: fase de envio, fase de transferência e fase de recebimento. Os parâmetros considerados pelo modelo são os seguintes:

p: Se refere ao número de processadores disponíveis;

$\boldsymbol{O_s}$(Fase de envio): Este é um evento síncrono iniciado pelo processo que envia a mensagem, onde é considerado como o tempo para preparo da mensagem mais o tempo de interação com a interface de rede. Esta fase é modelada pela função linear $\boldsymbol{O_s(m) = k_s + \tau_s m}$, onde $\boldsymbol{k_s}$ é o custo de inicialização, $\boldsymbol{m}$ é o tamanho da mensagem e $\boldsymbol{\tau_s}$ é a taxa de transferência de dados da memória.

$\boldsymbol{g_s}$(Gap de envio): Representa o intervalo de tempo necessário para que o dis-

positivo de rede possa enviar outra mensagem após ter terminado um envio. Este parâmetro é definido como: $g_s(m) = g_1 + \tau_1 m$, onde g_1 é o custo de inicialização da transferência dos dados e τ_1 é a largura de banda disponível.

L(Fase de transferência): Este parâmetro representa o tempo gasto pela rede para transportar os dados da memória do processo fonte para a memória do processo destino. Ele encapsula diferentes aspectos dos componentes da rede como: largura de banda, topologia da rede e o protocolo de comunicação. A fase de transferência é modelada pela função $L(m,p) = l(p) + \tau_1(m,p)m$, sob condição de o link de comunicação estar livre de congestionamento.

g_r(Gap de recebimento): Representa o intervalo de tempo mínimo entre dois recebimentos consecutivos. A modelagem deste parâmetro é feita pela função $g_r(m) = g_2 + \tau_2 m$, onde g_2 é o custo de inicialização do recebimento dos dados e τ_2 pode ser considerado o mesmo que τ_1.

O_r(Fase de recebimento): Este parâmetro trata da sobrecarga do *software* no tratamento de mensagens recebidas. O modelo considera que essa fase é manipulada por um processo do *kernel* do sistema não envolvendo o processo da aplicação que recebe a mensagem. O modelo expressa o parâmetro pela função $O_r(m) = k_r + \tau_r m$, onde k_r representa o custo mínimo da operação e τ_r representa o custo de movimentação de dados na memória.

Este modelo trata os parâmetros de comunicação como funções do tamanho da mensagem mas não considera a heterogeneidade das unidades de processamento, não sendo suficiente para modelar de maneira eficiente aplicações em ambientes Grid.

3.10 Modelo HLogP para escalonamento em Grids

Dentre todos os modelos apresentados, o modelo LogP [14] e seus variantes, LogGP [2], LogGPS [27], e o modelo realístico [41], são os que melhor representam as características das arquiteturas paralelas de memória distribuída em uso hoje.

Embora os ambientes Grid possuam características de sistemas com memória distribuída e a comunicação ocorra através de troca de mensagens, outras características adicionais como, por exemplo, a heterogeneidade das unidades de processamento e o compartilhamento da rede de interconexão entre essas unidades, tornam a utilização dos modelos apresentados não muito apropriada para esses ambientes. Os modelos LogP, LogGP e LogGPS, consideram iguais todas as unidades de processamento, a latência, a sobrecarga de envio e a sobrecarga de recebimento para a comunicação entre quaisquer pares de unidades de processamento. Já o modelo proposto por Wang, não considera a heterogeneidade dos parâmetros para unidades de processamento diferentes. As características desses modelos os tornam mais adaptados a sistemas homogêneos, o que não é o caso de sistemas Grid.

O modelo HLogP proposto neste trabalho baseia-se no modelo LogP e no modelo de Wang, considerando as características heterogêneas dos ambientes Grid. O objetivo do modelo é capturar as características dos recursos de sistemas distribuídos heterogêneos e não detalhes da arquitetura tais como topologias de redes de interconexão, características específicas dos *hosts*, entre outras. Os parâmetros são os mesmos propostos no modelo LogP, sendo considerada apenas a possibilidade de variação dos mesmos em função do tamanho das mensagens e de quais unidades de processamento se comunicam, como visto a seguir:

P: refere-se às unidades de processamento, sendo considerado o número de unidades presentes. $\mathbf{H}(\boldsymbol{p_i})$ é um fator que representa o grau de diferença da capacidade de processamento das unidades, denominado Fator de Heterogeneidade;

$\mathbf{L}(\boldsymbol{m}, \boldsymbol{p_i}, \boldsymbol{p_j})$: refere-se ao atraso no envio de uma mensagem de tamanho m de uma unidade de processamento $\boldsymbol{p_i}$ a outra $\boldsymbol{p_j}$. A latência nesse modelo é definida entre cada par de unidade de processamento;

$\mathbf{Os}(\boldsymbol{m}, \boldsymbol{p_i})$: representa a quantidade de tempo que a unidade de processamento $\boldsymbol{p_i}$ gasta para preparar e enviar uma mensagem de tamanho m, sendo que durante esse tempo a unidade não pode realizar nenhum outro processamento;

$\mathbf{Or}(\boldsymbol{m}, \boldsymbol{p_j})$: representa a quantidade de tempo que a unidade de processamento

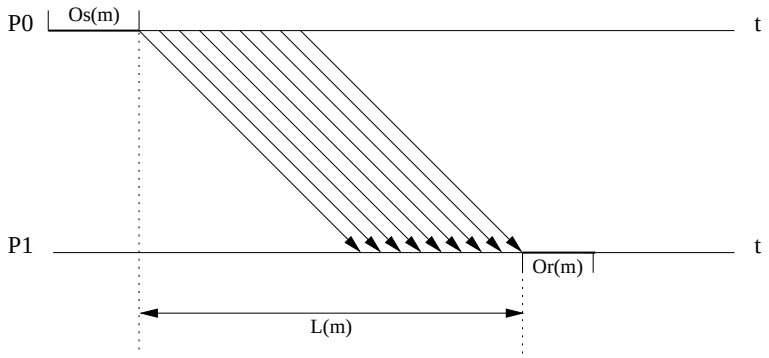


Figura 3.4: Comunicação segundo modelo HLogP.

p_j gasta para receber uma mensagem de tamanho m. Durante esse tempo também a unidade não pode realizar nenhum outro processamento.

Os parâmetros de comunicação do modelo HLogP são considerados funções do tamanho da mensagem e dos processadores envolvidos. O tempo de envio de uma mensagem de tamanho m entre dois processos nos processadores p_i e p_j é definido como $Os(m, p_i) + L(m, p_i, p_j) + Or(m, p_j)$, como pode ser visto na Figura 3.4

O modelo HlogP proposto não tem como objetivo servir de modelo para análise de desempenho e sim servir de modelo de escalonamento para aplicações MPI em ambientes heterogêneos, como é o caso dos ambientes Grid.

3.11 Resumo do Capítulo

Neste capítulo foram apresentados vários modelos propostos para ambientes paralelos, sendo que os mesmos não possuem características que tornem seus usos eficientes no que se refere ao escalonamento de aplicações MPI em ambientes Grid. O modelo HLogP foi proposto para tal fim, sendo mostrado, nos capítulos seguintes, sua validação e uso no escalonamento de aplicações MPI para ambientes Grid.

Capítulo 4

Calibragem do modelo HLogP

Este capítulo apresenta três mecanismos de calibragem dos parâmetros do modelo HLogP proposto no Capítulo 3. Estes mecanismos, denominados modeladores, capturam os valores referentes aos parâmetros do modelo HlogP, disponibilizando-os a sistemas escalonadores de tarefas voltados para o ambiente Grid escolhido. As duas primeiras versões são aplicações distribuídas, implementadas utilizando-se a biblioteca de troca de mensagens MPI, para medir os valores dos parâmetros do modelo. A terceira, menos intrusiva, baseia-se em métodos analíticos para a determinação dos valores.

4.1 Introdução

Informações referentes ao desempenho de componentes de sistemas computacionais, como a velocidade de processamento e as características da rede de interconexão, são especificadas pelo próprio fabricante dos equipamentos. Em sistemas onde os recursos são compartilhados entre várias aplicações (ou usuários), geralmente essas informações sobre o desempenho não representam as características reais dos sistemas por serem definidas como sendo os valores de desempenho sob condições ideais. Dessa forma, as informações referentes ao desempenho devem ser obtidas através de processos de modelagem, que podem ser realizados de forma ana-

lítica, onde equações são utilizadas para expressar uma determinada característica, ou através de *benchmarks*, onde os dados são medidos no próprio sistema. Os processos analíticos são, na maioria das vezes, mais complexos e de difícil definição. Já os métodos de *benchmarks* são mais simples e levam a uma melhor precisão; porém são mais intrusivos, podendo interferir na performance do sistema quando aplicados.

Existem mecanismos como o NWS[44, 45] e o do MDS do Globus Toolkit[16], já mencionados no Capítulo 2, para coleta de informações sobre a capacidade de processamento disponível em recursos computacionais e sobre a latência de comunicação entre os mesmos. Porém, esses mecanismos de coleta não atendem por completo aos requisitos do modelo HLogP, pois não informam todos os parâmetros que o modelo considera. O NWS, por exemplo, não determina parâmetros como as sobrecargas de envio e de recebimento. Já o MDS, em sua configuração padrão de instalação, não disponibiliza nenhuma informação referente à comunicação. Outro fato importante é que mesmo informações como a latência de comunicação disponibilizada pelo NWS não são obtidas a partir dos mecanismos de comunicação utilizados pelas aplicações. O NWS coleta informações de latência e largura de banda utilizando pacotes no nível 3 do modelo OSI [30], podendo resultar em informações imprecisas do ponto de vista da aplicação e do processo de escalonamento. Neste caso, o ideal é que os valores dos parâmetros sejam obtidos através de mecanismos de comunicação o mais próximo possível dos mecanismos utilizados pela aplicação, que normalmente utilizam pacotes em níveis iguais ou superiores ao nível 4.

O mecanismo de modelagem proposto nesta dissertação tem como objetivo a coleta dos parâmetros do modelo HLogP, para o escalonamento de aplicações MPI em ambientes Grid. Neste trabalho foram implementadas três versões do modelador. Cada versão possui melhorias no que se refere à qualidade dos valores coletados ou à intrusão gerada pelo modelador no sistema Grid onde o mesmo é executado. A primeira versão foi desenvolvida de forma a se obter os parâmetros do modelo para um tamanho de mensagem fornecido, sem preocupação com o desempenho do modelador. Na segunda versão foi realizada uma melhoria no processo de coleta de informações sobre a comunicação, sendo implementado um método mais simples de modelagem da comunicação entre dois pares de *hosts*. Também foi criada uma nova

estrutura de modelagem onde o processo de coleta das informações não é aplicado para todos os pares de *hosts* do Grid. Somente um subcojunto dos *hosts* é utilizado, diminuindo o tempo de execução do modelador e também a intrusão gerada pelo mesmo sobre a rede de comunicação entre os *hosts*.

Na tentativa de diminuir o número de execuções do modelador (com tamanhos de mensagens variados), a terceira versão propõe que os valores dos parâmetros do modelo, para qualquer tamanho de mensagem, sejam determinados por um processo analítico. O objetivo é propor um processo de modelagem que seja bastante preciso e com uma baixa intrusão. O processo analítico toma como base valores dos parâmetros obtidos pela modelagem realizada utilizando-se uma das versões anteriormente definidas. Com base nesses valores, equações propostas são utilizadas para realizar a previsão dos valores atuais dos parâmetros do modelo.

4.1.1 Ambiente de modelagem

Para todas as versões do modelador, o sistema recebe como entrada um arquivo contendo informações sobre os *hosts* ativos aos quais o usuário tem acesso. Este arquivo, como no exemplo da Figura 4.1, contém em cada linha o nome do *host* e, caso seja necessário, o caminho onde se encontram os arquivos executáveis do modelador (se no *host* o executável não fica no mesmo caminho em que fica no *host* onde o modelador vai ser iniciado). Ao final do processo de modelagem os resultados são apresentados em um arquivo de saída, como é ilustrado na Figura 4.2, onde temos na primeira linha o número de *hosts* modelados, em seguida uma linha para cada *host* contendo: o fator de heterogeneidade **H** do *host*, o nome do *host*, a sobrecarga de envio, a sobrecarga de recebimento, o sistema operacional do *host*, e por último o caminho onde os arquivos executáveis do modelador estão armazenados. Após as linhas dos *hosts*, uma matriz contendo as latências de comunicação entre cada par de hosts modelados é apresentada.

A seguir são apresentadas as versões do modelador em detalhes e os resultados dos testes obtidos da aplicação dos mesmos em um ambiente Grid. O ambiente

```
siriguela.ic.uff.br
melancia.ic.uff.br
n01.par.inf.puc-rio.br /home/local/gridriouser/helder
caju.ic.uff.br
```

Figura 4.1: Exemplo de um arquivo de entrada do modelador.

```
4
298 caju.ic.uff.br 28 3 linux
323 siriguela.ic.uff.br 27 5 linux
426 melancia.ic.uff.br 45 6 linux
250 n01.par.inf.puc-rio.br 25 4 linux /home/local/gridriouser/helder
 0        234      238      23,567
 246      0        226      25,432
 230      246      0        24,126
 27,036   26,223   23,677   0
```

Figura 4.2: Exemplo de um arquivo de saída do modelador.

utilizado para realização dos testes foi o GridRio; um Grid computacional para fins de pesquisa na área, envolvendo três organizações de pesquisa localizados nas cidades de Niterói e Rio de Janeiro: Instituto de Computação da Universidade Federal Fluminense (UFF), Departamento de Informática da Pontífica Universidade Católica do Rio de Janeiro (PUC-Rio) e o Centro Brasileiro de Pesquisas Físicas (CBPF). A Figura 2.2, apresentada no Capítulo 2, mostra a página do *site* oficial do GridRio onde temos uma lista parcial dos *hosts* ativos no Grid durante uma consulta. Os dados apresentados são obtidos acessando-se o MDS do GridRio. Devido ao uso do MPI, os testes foram realizados apenas entre máquinas da UFF e da PUC pois as máquinas do CBPF utilizam endereços IP privados.

Host	*site*	Processador	Velocidade	Memória	Kernel do Linux
sn00.ic.uff.br	UFF	Intel Pentium IV	2.6 GHz	512 Mb	2.6.8-1.521
sn01.ic.uff.br	UFF	Intel Pentium IV	2.6 GHz	512 Mb	2.6.8-1.521
sn02.ic.uff.br	UFF	Intel Pentium IV	2.6 GHz	512 Mb	2.6.8-1.521
sn04.ic.uff.br	UFF	Intel Pentium IV	2.6 GHz	512 Mb	2.6.8-1.521
jenipapo.ic.uff.br	UFF	AMD Athlon	1.4 GHz	256 Mb	2.4.21-0.16mm-mdk
caju.ic.uff.br	UFF	AMD Athlon	1.3 GHz	256 Mb	2.6.3-15mdk
sn16.ic.uff.br	Sinergia	Intel Pentium IV	2.6 GHz	512 Mb	2.6.8-1.521
sn17.ic.uff.br	Sinergia	Intel Pentium IV	2.6 GHz	512 Mb	2.6.8-1.521
n19.par.inf.puc-rio.br	PUC	Intel Pentium IV	1.7 GHz	256 Mb	2.4.18-19.7.x
n20.par.inf.puc-rio.br	PUC	Intel Pentium IV	1.7 GHz	256 Mb	2.4.18-19.7.x
n21.par.inf.puc-rio.br	PUC	Intel Pentium IV	1.7 GHz	256 Mb	2.4.18-19.7.x
n22.par.inf.puc-rio.br	PUC	Intel Pentium IV	1.7 GHz	256 Mb	2.4.18-19.7.x

Tabela 4.1: Hosts Modelados.

A Tabela 4.1 apresenta o nome, o fabricante e o modelo do processador, a velocidade do relógio e a quantidade de memória de cada *host* (Dados tambem disponibilizados pelo GIIS do MDS do Grid Computacional GridRio).

4.2 Versão I do modelador

A primeira versão do modelador foi implementada utilizando-se mecanismos de coleta das informações já propostos em outros trabalhos relacionados, [29], uma vez que o objetivo era o estudo da aplicabilidade da técnica de coleta de parâmetros do modelo LogP utilizando a biblioteca de troca de mensagens MPI. Foram feitos apenas ajustes nas implementações para adaptação ao ambiente Grid e à forma de coleta pretendida.

A modelagem nessa versão é realizada por uma aplicação paralela onde um processo é executado em cada *host* do Grid. Um dos processos em execução em um dos *hosts* do *site* do usuário, é eleito controlador da modelagem. Suas funções são: leitura dos arquivos de entrada, controle do processo de modelagem, determinando qual par de *hosts* fará a modelagem a cada instante, recebimento dos parâmetros coletados pelos outros processos, e formatação do arquivo de saída. Cada um dos outros processos é responsável pela modelagem da capacidade de processamento disponível no seu respectivo *host* e pela modelagem da comunicação com cada um dos outros *hosts* do Grid. Durante a modelagem, o processo coordenador envia uma mensagem ao par de processos coordenados dizendo que os mesmos farão a medida dos parâmetros. Um dos processos do par recebe uma mensagem dizendo que ele iniciará a medida dos parâmetros e a informação de qual é o identificador (rank) do processo que responderá à medida. O outro processo recebe uma mensagem informando que ele fará o espelhamento (devolução das mensagens recebidas) do processo de medida realizado pelo primeiro. Cada par de processos realiza a modelagem dos parâmetros, trocando informações entre si. Os valores medidos são enviados ao processo coordenador que recebe os parâmetros coletados; essa etapa da modelagem será melhor detalhada mais a diante na descrição da modelagem da comunicação.

Cada processo da aplicação, consequentemente cada *host* do Grid, realiza a medida dos parâmetros $n-1$ vezes, considerando que ele não faz a medida e nem o espelhamento com ele mesmo. Dessa forma, como são n processos, são necessárias um total de n^2-n operações de modelagem entre pares de processos. O processo coordenador permite que a modelagem seja realizada entre apenas um par de *hosts* a cada instante, de forma a garantir que os parâmetros coletados não sofram influência da modelagem entre outros pares, o que poderia ocorrer se o sistema permitisse a modelagem entre mais de um par de *hosts* ao mesmo tempo. Ao final do processo de medida entre todos os pares de *hosts*, os parâmetros coletados são gravados em um arquivo de saída, como mostrado na Figura 4.2. A Figura 4.3 mostra um exemplo da comunicação gerada pelo modelador.

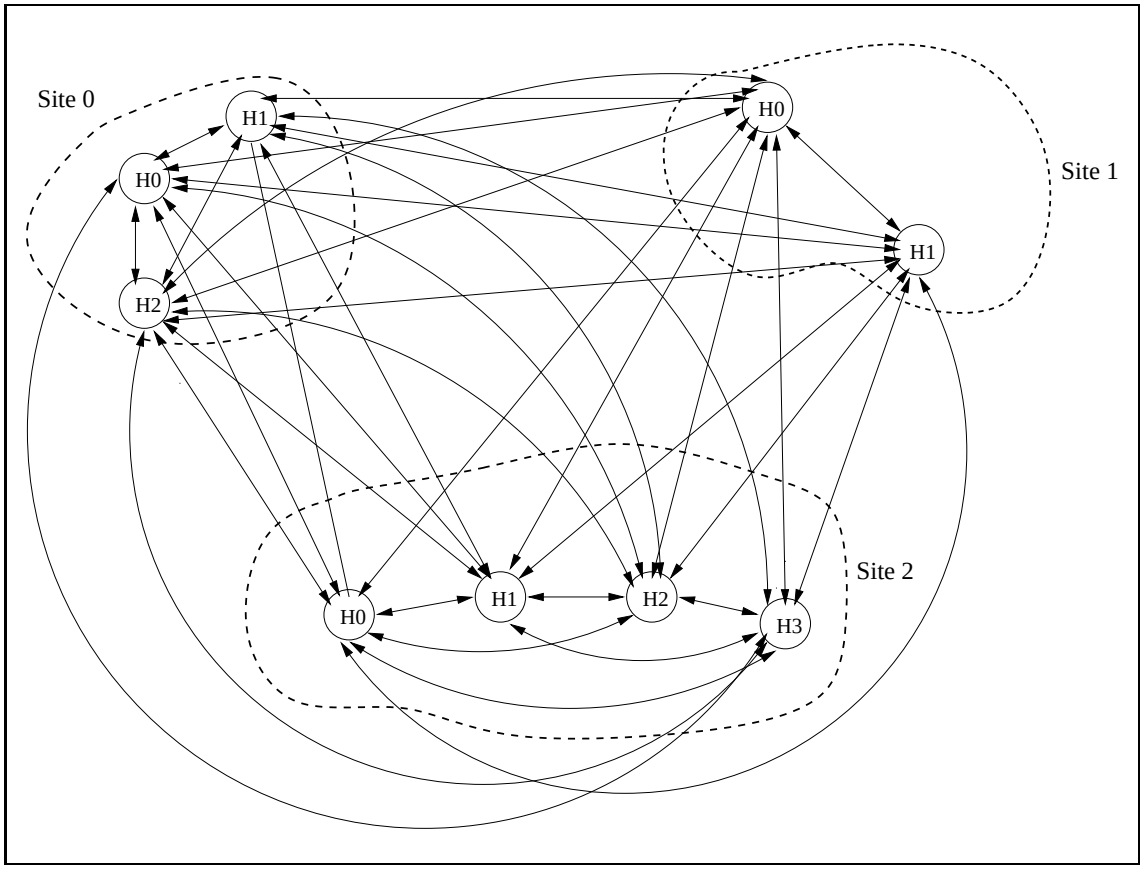


Figura 4.3: Modelagem da comunicação entre todos os hosts.

4.2.1 Modelagem da capacidade de processamento disponível

A modelagem da capacidade de processamento disponível num *host* é realizada através da medida do tempo gasto pelo *host* para executar um trecho de código. O trecho foi escolhido segundo o critério de simplicidade, onde o que se buscou foi um trecho de código que produzisse uma carga no processador do *host* e que fosse o mais simples possível. O trecho escolhido inicialmente, realiza uma série de operações matemáticas de incremento de uma variável do tipo inteiro simples.

O tempo obtido da execução do trecho de código é então normalizado em função do número de repetições realizadas. O fator obtido, denominado fator de heterogeneidade (**H**) dos *hosts*, expressa tanto as características estáticas (velocidade da CPU, velocidade de acesso a memória, etc) como as características de carga (número de processos ativos, utilização da CPU, etc) dos *hosts*. Dessa forma, um *host* que tenha um processador mais lento ou que esteja com uma carga mais alta terá um fator maior do que um *host* com processador mais rápido ou com carga menor.

Por motivos de validação, o modelador captura, também, imediatamente antes de determinar o fator de heterogeneidade de cada *host*, algumas informações do sistema via sistema operacional do *host*: número de processos ativos, carga e percentual livre do processador. Esses parâmetros são obtidos através de leituras de arquivos do sistema linux, como /proc/loadavg e da saída do comando vmstat.

4.2.2 Resultado da modelagem da capacidade de processamento disponível

Os testes para verificação da precisão do mecanismo de modelagem do fator de heterogeneidade foram, inicialmente, realizados em um ambiente controlado, mantendo os *hosts* com carga zero durante a modelagem. Os fatores obtidos, apresentados na Tabela 4.2, retratam as diferenças entre os *hosts* do ponto de vista de suas características estáticas (velocidade da cpu, quantidade de memória, etc).

Além do fato de que o fator consegue refletir o desempenho com boa precisão, é também robusto às variações típicas do tempo de execução em uma máquina.

Host	Carga	Tam. Fila	Perc. CPU Livre	Fator H
sn00	0	1	100	23
sn01	0	1	100	23
sn02	0	1	100	23
sn04	0	1	100	23
jenipapo	0	1	100	40
caju	0	1	100	42
sn16	0	1	100	23
sn17	0	1	100	23
n19	0	1	100	36
n20	0	1	100	36
n21	0	1	100	36
n22	0	1	100	36

Tabela 4.2: Fatores Modelados com carga 0.

Foram feitos, também, testes com carga nos *hosts*, sendo que alguns tinham carga gerada por outros usuários executando outros programas e outros tiveram uma carga fixa gerada de maneira controlada. Nos *hosts* com carga controlada, a variação foi obtida executando-se em cada *host* um número de instâncias de um processo que consome tempo da CPU, denominado *processo consumidor*. Em alguns dos *hosts* onde não foram executadas instâncias do *processo consumidor*, outros processos de usuários estavam sendo executados no momento da modelagem, como por exemplo os *hosts* n19, n20, n21 e n22. Os fatores obtidos, apresentados na Tabela 4.3, mostram a sensibilidade do sistema de modelagem também às características dinâmicas dos *hosts* (número de processos, carga, etc), onde podemos verificar o aumento dos fatores de heterogeneidade em função da carga.

Host	Carga	Num. Processos Consumidores	Tam. Fila	Perc. CPU Livre	Fator H
sn00	3,31	2	3	0	66
sn01	4,59	4	5	0	111
sn02	10,42	8	9	0	200
sn04	12,25	10	11	0	250
jenipapo	2,20	0	3	0	94
caju	1,01	0	2	0	100
sn16	0	0	1	100	23
sn17	0	0	1	100	23
n19	2,29	0	3	0	120
n20	2,07	0	4	0	121
n21	2,04	0	4	0	122
n22	2,00	0	4	0	122

Tabela 4.3: Fatores Modelados com carga diferente de 0.

Na Tabela 4.3, para os *hosts* com carga controlada (processos consumidores), podemos observar que os valores referentes ao tamanho da fila de processos

ativos sempre apresenta um processo a mais do que o número de processos consumidores. Esse outro processo ativo é o processo referente ao modelador, uma vez que os valores apresentados foram coletados através do modelador. No caso da Tabela 4.2, também verificamos que o tamanho da fila é igual a um, ou seja, apenas o processo do modelador se encontrava em execução.

Podemos observar também, na Tabela 4.3, que os valores do fator de heterogeneidade aumentam para cada *host* proporcionalmente ao número de processos ativos concorrendo pelo processador. Por exemplo, no caso do *host* sn02, o número de processos em execução, considerando o processo modelador, é igual a 9, o que resulta em um fator de heterogeneidade igual a 207. Este valor, calculado multiplicando-se o número de processos ativos (9) pelo fator de heterogeneidade modelado sem carga (23), é bem próximo do valor real obtido que foi de 200. Este assunto será novamente abordado na subseção 4.4.1.

4.2.3 Modelagem da comunicação

Para a modelagem da comunicação nesta primeira versão do modelador, foi utilizado um algoritmo baseado no sistema de modelagem de parâmetros do modelo LogP proposto por Thilo Kielmann *et al* [29], com algumas alterações introduzidas para que o mesmo possa ser utilizado para modelar qualquer par de processos da aplicação modeladora (originalmente somente eram modelados dois processos fixos). O algoritmo proposto por Kielmann *et al.* tem como objetivo otimizar a medida do parâmetro **g** do modelo LogP para tamanhos de mensagens variados. Em outros métodos [15, 25], a medida de **g**, para tamanhos diferentes de mensagens, é feita saturando-se o link enviando um conjunto de mensagens; o que não é conveniente, pois as mensagens enviadas durante o processo de medida podem prejudicar demais a comunicação de outros processos do sistema. O método proposto por Kielmann *et al.*, satura o link apenas para mensagens de tamanho 0. Para mensagens de tamanhos maiores, o valor de **g** é obtido com base no valor desse mesmo parâmetro obtido para mensagens de tamanho 0.

Método de Kielmann *et al.* [29]

A implementação do método *Fast Parameter Measurement* utiliza dois processos, um denominado *measure* e outro *mirror*. Inicialmente, o método determina o valor do tempo de ida e volta para uma mensagem de tamanho 0 (**RTT(0)-Round Trip Time**) e o valor do intervalo de tempo entre envios consecutivos de mensagens (**g(0)**), que serão utilizados na determinação dos parâmetros para tamanhos de mensagens diferentes de 0. Para determinação do **RTT(0)**, o processo *measure* envia um número pré-determinado n de mensagens de tamanho **0** para o processo *mirror* que retorna uma mensagem de tamanho 0 para cada mensagem recebida, sendo calculado o tempo gasto para cada mensagem. O valor do **RTT(0)** é computado a partir da média das medidas realizadas. Na determinação do **g(0)**, o processo *measure* envia conjuntos de n mensagens de tamanho m para o processo *mirror* que retorna, para cada mensagem recebida, uma mensagem de tamanho 0. O número de mensagens n de cada conjunto começa em 10 e vai sendo dobrado a cada passo até que o link seja saturado, o que é determinado quando o gap **g** por mensagem varia somente em 1%.

Para as mensagens de tamanhos diferente de 0, a medida dos parâmetros é realizada em duas fases, sendo que em cada fase conjuntos de n mensagens são trocados entre os processos. O tamanho do conjunto de mensagens trocadas, em cada fase, inicia-se com 3 e é incrementado de três em três até que o gap entre mensagens mude apenas por um fator de erro igual a 1%, ou que um tamanho de mensagem máximo, pré-definido (18), seja atingido. Na primeira fase, como visto na Figura 4.4, o processo *mirror* envia uma mensagem de tamamho 0 (a) para o processo *measure* dizendo que está pronto para responder. Em seguida, o processo *measure* começa a enviar (b) o conjunto de mensagens de tamanho m para o processo *mirror* que retorna uma mensagem (c) de tamanho 0, garantindo ao *measure* que a mensagem foi recebida. Após a conclusão da primeira fase, os valores da sobrecarga de envio (**Os**) e do tempo de ida e volta da mensagem de tamanho m (**RTT(m)**), são calculados pelo processo *measure*. Na segunda fase, o processo *mirror* novamente envia uma mensagem de sincronização (d) ao processo *measure* que então começa a

enviar (e) os conjuntos de mensagens de tamanho 0 ao processo *mirror*. Para cada mensagem de tamanho 0 recebida pelo processo *mirror* uma mensagem de tamanho m (f) é retornada ao processo *measure*. Ao término da segunda fase o valor da sobrecarga de recebimento (**Or**) é calculada pelo processo *measure*.

O cálculo de $\boldsymbol{g(m)}$, $\boldsymbol{RTT(m)}$ e a latência $\boldsymbol{L(m)}$ para mensagens de tamanho m maior que *0* é feito segundo as seguintes equações:

$$\boldsymbol{RTT(m) = L + g(m) + L + g(0)}$$

$$\boldsymbol{g(m) = RTT(m) - RTT(0) + g(0)}$$

e $\boldsymbol{L = (RTT(0) - 2 * g(0))/2}$.

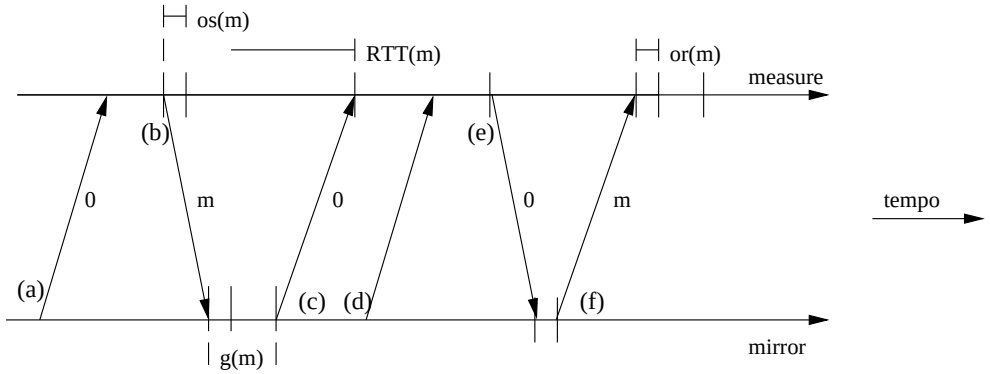


Figura 4.4: Processo de modelagem da comunicação segundo o método de Kielmman.

4.2.4 Resultados da modelagem da comunicação

Os resultados das modelagens para a latência, sobrecarga de envio e sobrecarga de recebimento, utilizando-se a versão I do modelador, podem ser observados nas Tabelas 4.4, 4.5 e 4.7 respectivamente, onde os tempos são apresentados em microsegundos. Os testes foram realizados utilizando-se *hosts* da UFF, sendo que quatro *hosts* são conectados por um switch fast ethernet e dois interligados por um switch gigabit, e os switchs interligados por um switch ethernet de 10 megabits. Na PUC, foram utilizados quatro *hosts* interligados por um switch fast ethernet, sendo que a ligação entre a UFF e a PUC é estabelecida pela infraestrutura internet da

RedeRio. A versão simplificada desta configuração pode se visualizada na Figura 4.5. Durante os testes os hosts estavam em modo exclusivo, sendo executados apenas processos do sistema e da aplicação modeladora, e o tamanho de mensagem fixo utilizado foi de 64 Kbytes. Podemos observar na Tabela 4.4 que os valores da latência são de mesma ordem para a comunicação interna de cada *site* (*hosts* interligados por um mesmo switch). Com relação à comunicação *inter-site* (entre *hosts* da UFF e PUC), observamos que os valores obtidos não são muito homogêneos, provavelmente devido a ocilações no tráfego da internet.

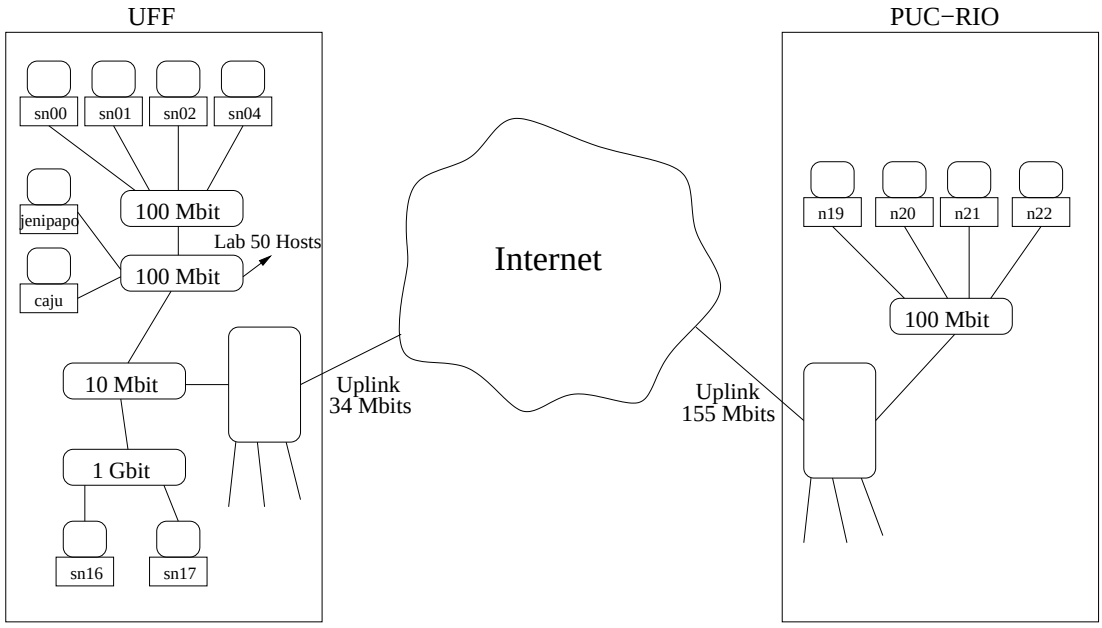


Figura 4.5: Estrutura da rede de interligação entre os hosts do GridRio utilizados nos testes.

Ori./Dest.	sn02	sn04	Jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		161	157	161	503	478	7.083	5.501	5.813	5.307
sn04	160		157	161	441	300	5.729	5.810	4.192	4.390
jenipapo	163	207		103	411	408	7.246	6.476	4.560	4.867
caju	162	162	99		457	464	4.980	4.291	5.189	4.408
sn16	358	318	165	232		86	5.730	5.991	5.013	4.825
sn17	336	460	203	226	102		4.791	4.426	4.244	4.522
n19	4.692	4.892	4.823	4.224	4.311	4.269		102	102	129
n20	4.451	6.040	5.754	5.562	4.905	4.597	99		99	132
n21	5.297	4.713	5.965	4.359	5.848	4.300	102	101		127
n22	4.274	5.552	4.182	4.285	4.322	4.329	130	129	129	

Tabela 4.4: Latências modeladas pela versão I do Modelador.

Analisando os valores apresentados na Tabela 4.5, verificamos que a sobrecarga de envio sofre pouca variação para mensagens enviadas com destino a *hosts*

de um mesmo *site*. Para comunicação envolvendo um *host* de origem no *site* UFF ou no *site* sinergia e um *host* de destino no *site* da PUC, os valores são da ordem de 100 vezes maiores. Isso se deve à utilização do comando MPI_Send (comando bloqueante) na versão I do modelador. O *host* de origem fica aguardando um retorno (*ack*) do *host* de destino que o envia somente após o recebimento de toda a mensagem. Como a velocidade da comunicação entre os *sites* UFF-PUC e sinergia-PUC é menor do que internamente nos dois *sites* (UFF e sinergia), a sobrecarga de envio tem um valor maior.

Ori./Dest.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		142	178	185	159	170	128.737	103.523	97.432	101.258
sn04	161		183	185	139	169	79.526	92.178	87.740	106.311
jenipapo	491	465		567	501	496	62.455	61.726	105.002	64.472
caju	211	214	228		231	333	62.243	107.244	91.488	104.693
sn16	186	182	174	166		145	116	75.919	58.072	62.754
sn17	171	162	169	168	136		112.116	146	67.471	71.480
n19	273	269	280	335	277	269		305	311	322
n20	294	271	287	275	242	258	302		297	306
n21	248	262	269	265	263	268	296	296		311
n22	286	274	299	335	263	270	339	332	329	

Tabela 4.5: Sobrecargas de envio modeladas pela versão I do Modelador.

Era esperado que a comunicação entre PUC-UFF (PUC-sinergia) e UFF-PUC (sinergia-PUC) fossem similares, por ser o mesmo link de comunicação. Entretanto, os resultados da Tabela 4.5 mostram que o valor da sobrecarga de envio para mensagens da UFF para sinergia e da PUC para UFF sinergia possuem valores inferiores aos valores para a comunicação entre UFF e PUC. Porem, com testes realizados considerando-se um tamanho maior de mensagem (1 Mbytes), esse fato não ocorreu, os valores da sobrecarga de envio entre PUC-UFF e PUC-sinergia foram maiores do que os valores para comunicação interna dos *sites*. Os resultados desse experimento podem ser visualizados na Tabela 4.6, sendo que por motivos de espaço os tempos foram apresentados em milisegundos.

Com relação à sobrecarga de recebimento, a Tabela 4.7 mostra que seus valores se comportam como a sobrecarga de envio, variando em função da capacidade de processamento do *host*. Por exemplo, o *host* sn02 possui valores menores do que o *host* n19, que possui capacidade de processamento inferior ao primeiro. Observamos também que os valores para mensagens recebidas de todos os *hosts* por um mesmo *host* são da mesma ordem, não sendo influenciado pelas características do *host* de

Ori./Dest.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		83	83	83	1.376	1.505	1.494	1.891	1.708	1.701
sn04	83		83	82	1.499	1.250	1.779	1.848	1.704	1.691
jenipapo	83	83		83	1.254	1.269	1.829	1.956	1.855	1.747
caju	83	124	83		1.200	1.236	1.736	1.890	1.855	1.719
sn16	1.267	1.066	1.280	1.214		8	1.761	1.717	1.611	1.758
sn17	1.106	1.082	1.234	1.207	9		1.872	1.661	2.074	1.883
n19	1.508	3.264	1.326	1.339	1.478	1.464		83	83	83
n20	1.444	1.397	1.332	1.405	1.377	1.423	83		83	83
n21	1.325	1.421	1.385	1.435	1.362	1.390	83	83		82
n22	1.356	1.333	1.406	1.285	1.390	1.421	83	83	83	

Tabela 4.6: Sobrecargas de envio para mensagens de 1 Mbyte (tempos em milisegundos).

origem.

Dest./Ori.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		53	52	53	58	60	66	59	57	60
sn04	56		58	57	54	54	62	57	63	55
jenipapo	258	261		261	267	266	265	250	157	273
caju	72	70	72		82	73	159	139	186	165
sn16	55	52	59	53		55	58	57	65	54
sn17	52	66	51	59	51		58	54	56	55
n19	133	144	140	149	140	135		131	132	140
n20	120	120	127	124	126	124	117		121	123
n21	107	111	107	116	109	118	97	96		99
n22	132	135	131	134	137	144	126	136	128	

Tabela 4.7: Sobrecargas de recebimento modeladas pela versão I do Modelador.

Foram feitos testes da modelagem da comunicação utilizando-se a versão I do modelador para tamanhos variados de mensagens. Os resultados, apresentados nas Tabelas 4.8, 4.9, 4.10 e nos gráficos da Figura 4.6, mostram que a sobrecarga de envio e a sobrecarga de recebimento são influenciados também pelo tamanho da mensagem enviada, como foi verificado também por Fumihiko Ino *et al.* [27]. Para mensagens inferiores a 64 Kbytes os valores são relativamente constantes ou têm pouca variação. Para tamanhos maiores que 64 Kbytes, os valores apresentam um crescimento linear. Essa mudança de comportamento para mensagens de tamanhos superiores a 64 Kbytes ocorre por uma mudança no protocolo de comunicação. Neste caso, a comunicação é síncrona e o processo que envia a mensagem aguarda um retorno (ACK) do processo destino quando o recebimento é finalizado. Para mensagens de tamanho inferior, a comunicação é assíncrona e o processo que envia não aguarda um retorno do processo destino. Por outro lado, a latência, que é tratada como o atraso de enviar o primeiro bit, efetivamente parece constante, independente do tamanho da mensagem.

Tam.Mensagem (Bytes)	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)
64	103.925	40	4
128	103.920	44	4
256	103.925	39	5
512	103.930	40	4
1.024	103.910	44	5
2.048	103.910	56	6
4.096	103.925	42	9
8.192	103.925	44	6
16.384	103.915	61	8
32.768	103.960	71	19
65.536	103.925	141	51
131.078	103.915	198	133
262.144	103.920	203.997	376
524.288	103.945	407.972	856
1.048.576	103.910	794.604	1.669
2.097.152	103.915	1.631.815	3.307
4.194.304	103.935	3.330.298	6.541
8.388.608	103.915	6.602.152	13.148
16.777.216	103.900	13.266.141	26.250
33.554.432	104.915	26.585.246	52.867
67.108.864	103.911	53.179.419	138.894

Tabela 4.8: Modelagem da comunicação entre processos num mesmo host, para tamanhos de mensagens diferentes, usando a versão I do Modelador.

Um fato interessante pode ser observado, comparando-se os resultados das Tabelas 4.8 e 4.9. Os valores obtidos para a latência e a sobrecarga de envio para a comunicação entre processos localizados num mesmo *host* é frequentemente maior em comparação aos valores dos mesmos parâmetros para a comunicação entre processos em *hosts* diferentes. Essa característica é exclusiva da versão da biblioteca MPICH-G2 utilizada nos testes e é ocasionada pelo fato de que o processo que aguarda o envio da mensagem faz um *polling* enquanto a mensagem não chega. Neste caso, como os processos origem e destino são executados no mesmo *host*, a troca de contexto entre os processos ocasiona um aumento nos valores dos parâmetros. Mais adiante no Capítulo 5, será mostrado o resultado dos mesmos testes utilizando-se a biblioteca MPI-LAM, sendo o comportamento diferente neste caso.

Os gráficos, apresentados na Figura 4.6, mostram o crescimento linear das sobrecargas de envio e recebimento, para tamanhos de mensagens superiores a 64 KBytes. No caso da latência, ocorre uma variação que está dentro do que será visto nos experimentos mais detalhados na Seção 4.4.3. Os testes para a geração dos gráficos foram realizados utilizando-se dois *host* da UFF (gráficos da esquerda) e um *host* da UFF e um da PUC (gráficos da direita). Podemos observar também que o comportamento da comunicação entre *hosts* de um mesmo *site* e entre *hosts*

Tam.Mensagem (Bytes)	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)
64	187	34	3
128	188	33	2
256	188	32	2
512	187	32	3
1.024	187	34	3
2.048	187	36	3
4.096	187	43	3
8.192	188	47	5
16.384	189	64	7
32.768	187	69	14
65.536	156	188	39
131.078	154	3.472	157
262.144	154	16.274	381
524.288	152	38.667	843
1.048.576	154	83.724	1.680
2.097.152	187	170.516	3.364
4.194.304	224	350.421	6.851
8.388.608	195	734.368	13.432
16.777.216	151	1.419.170	60.935
33.554.432	161	2.903.808	53.368
67.108.864	161	5.814.361	106.593

Tabela 4.9: Modelagem da comunicação entre dois hosts da UFF, para tamanhos de mensagens diferentes, usando a versão I do Modelador.

de *sites* diferentes é muito similar, visto que as curvas são equivalentes.

4.2.5 Verificação da versão I do modelador

Uma verificação inicial dos parâmetros coletados pela versão I do modelador foi feita utilizando-se uma pequena aplicação distribuída, representada pelo grafo da Figura 4.7, onde três processos se comunicam em seqüência, tipo anel, com o último enviando uma mensagem de volta ao primeiro. O processamento realizado em cada processo é o mesmo realizado pelo modelador. Assim, como as mensagens enviadas também possuem o mesmo tamanho das utilizadas pelo modelador. Dessa forma, o relógio da máquina onde o processo T0 foi executado foi anotado antes do início do processamento do trecho de código e novamente tomado logo após o recebimento da mensagem enviada pelo processo T2. Antes da execução da aplicação, o modelador foi executado e os valores dos parâmetros de comunicação (sobrecarga de envio, sobrecarga de recebimento e latência) foram coletados para cada par de *hosts* modelados. As Tabelas da Figura 4.7 mostram os resultados da modelagem de duas configurações: a primeira com máquinas de um cluster e a segunda com máquinas de um Grid. Considerando os valores coletados, verificamos na Tabela 4.12 que é

Tam.Mensagem (Bytes)	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)
64	4.984	33	3
128	4.122	33	3
256	4.969	34	3
512	4.059	34	3
1.024	4.062	34	3
2.048	5.633	36	3
4.096	4.334	38	4
8.192	5.951	48	5
16.384	5.257	53	8
32.768	4.189	73	20
65.536	5.871	116	42
131.078	5.530	113.749	174
262.144	7.973	415.512	395
524.288	4.780	771.512	828
1.048.576	5.014	1.842.948	1.643
2.097.152	6.727	3.654.303	3.261
4.194.304	6.522	6.860.275	6.543
8.388.608	7.263	14.380.402	13.181
16.777.216	5.283	28.210.924	26.341
33.554.432	4.382	57.150.012	52.809
67.108.864	5.512	113.850.253	105.002

Tabela 4.10: Modelagem da comunicação entre hosts da UFF e PUC, para tamanhos de mensagens diferentes, usando a versão I do Modelador.

possível estimar o tempo de execução da aplicação, a partir dos valores modelados.

Etapa	Tempo (μs) UFF-UFF	Tempo (μs) UFF-PUC
Início T0	0	0
Proc(T0)	4.544.512	4.544.870
L(T0,T1)	156	5.871
g(65.535)	6.112	192.828
Proc(T1)	4.544.870	6.923.074
L(T1,T2)	157	5.871
g(65.535)	6.112	177.890
Proc(T2)	4.547.739	4.544.512
L(T2,T0)	156	156
g(65.535)	6.112	6.112
Fim de T0	13.650.426	16.401.184

Tabela 4.11: Evolução do tempo estimado para a aplicação da Figura 4.7, com comunicação entre processos da UFF e entre processos da UFF e PUC, com base nos parâmetros coletados pela versão I do Modelador.

4.3 Versão II do modelador

A versão II do modelador foi desenvolvida com o intuito de otimizar a modelagem da comunicação entre os *hosts* do Grid. A versão I realiza a modelagem da comunicação entre cada par de *hosts* do Grid, gerando uma sobrecarga muito alta no sistema. Uma outra característica indesejável é a má escalabilidade do

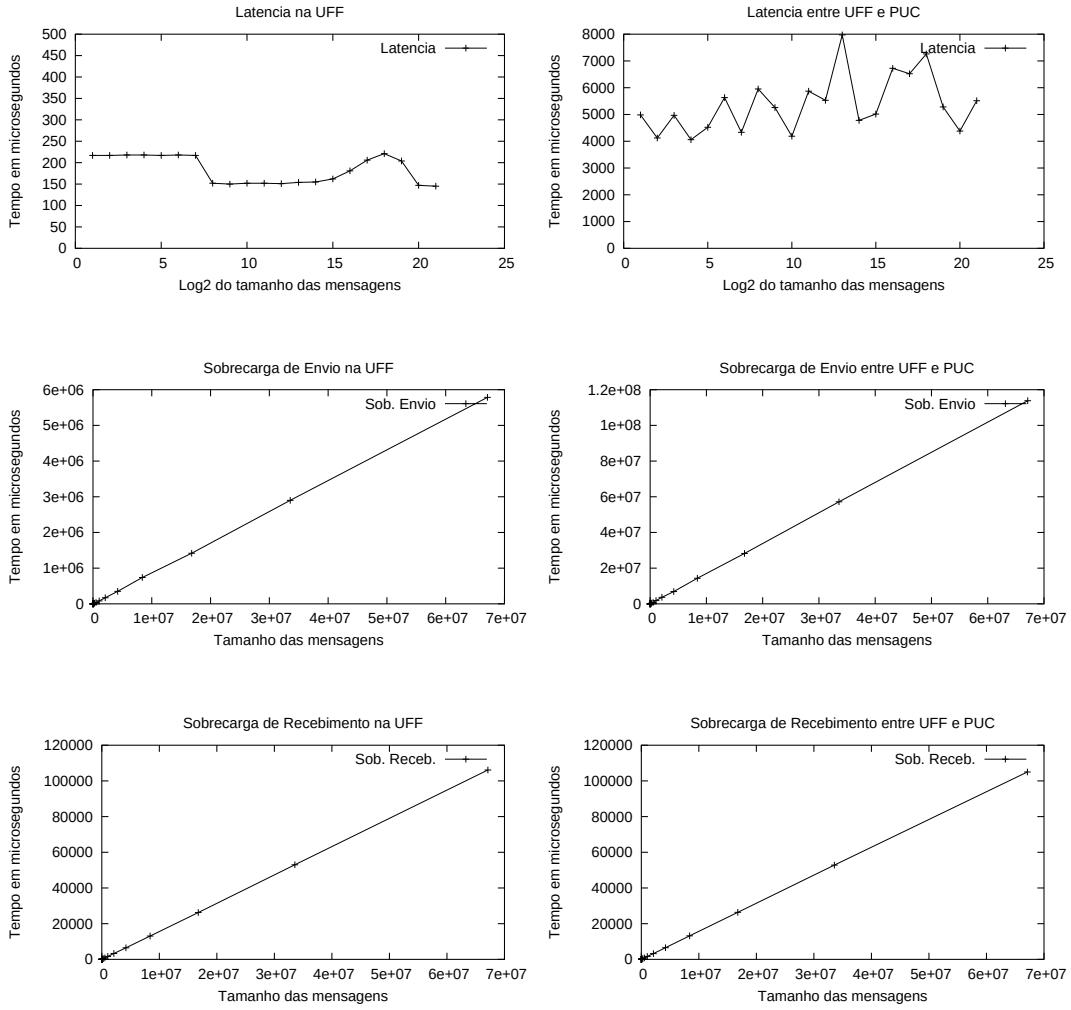
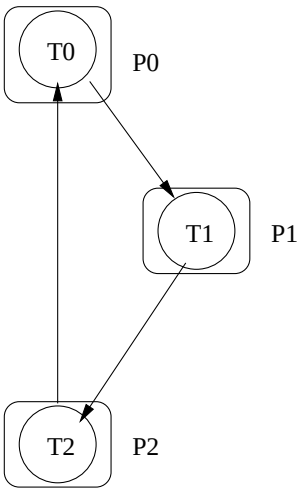


Figura 4.6: Gráficos de Comunicação Versão I do Modelador.

modelador pois, para cada novo *host* acrescentado, $\mathbf{2 * (n - 1)}$ medidas a mais entre *hosts* deverão ser executadas, onde $\boldsymbol{n}$ é o número de *hosts*.

Uma vez que, para um mesmo *site* é comum se ter uma certa homogeneidade da rede de interconexão entre os *hosts*, essa versão do modelador modela a comunicação entre um número menor de *hosts* de um mesmo *site*, por exemplo só entre dois de cada *site*, como ilustrado na Figura 4.8. Para os outros *hosts* que não participam da modelagem de comunicação, os parâmetros de comunicação são considerados iguais aos parâmetros de comunicação modelado entre os dois *hosts* participantes. A comunicação entre os *sites* é modelada escolhendo-se um *host* entre os dois *hosts* de cada *site*. Para a escolha dos *hosts*, por exemplo, pode-se basear



UFF

Processo	Host	Tempo Proc.
P0	sn00	4.544.512
P1	sn01	4.544.870
P2	sn02	4.547.739

	L	g(65536)
p0→ p1	156	6.112
p1→ p2	156	6.112
p2→ p0	156	6.112

UFF−PUC

Processo	Host	Tempo Proc.
P0	sn00	4.544.870
P1	n20	6.923.074
P2	sn01	4.544.512

	L	g(65536)
p0→ p1	5.871	192.828
p1→ p2	5.871	177.890
p2→ p0	156	6.112

Figura 4.7: Grafo da aplicação utilizada na verificação da modelagem e a estimativa dos valores dos parâmetros do modelo obtidos.

	Tempo Execução (μs)	Tempo Estimado (μs)	Erro (%)
UFF-UFF	13.782.189	13.650.426	**0, 97**
UFF-PUC	17.171.094	16.401.184	**4, 69**

Tabela 4.12: Comparação entre tempo de execução da aplicação e o tempo a partir dos parâmetros modelados, utilizando a versão I do Modelador.

em informações obtidas do MDS.

A modelagem da capacidade de processamento dos *hosts*, nesta versão, continua sendo realizada através da execução de um trecho de código em cada um dos *hosts* e o valor de **H** obtido da normalização do tempo gasto para a execução do trecho de código. Nesta versão do modelador, foram testados novos trechos, como será visto na seção seguinte.

Essa nova versão, também utiliza uma aplicação paralela para a modelagem, constituida de tantos processos quantos forem os *hosts* a serem modelados. Como no modelador I, um processo que é executado em um dos *hosts* do *site* do usuário é escolhido como coordenador do processo de modelagem e é responsável pela lei-

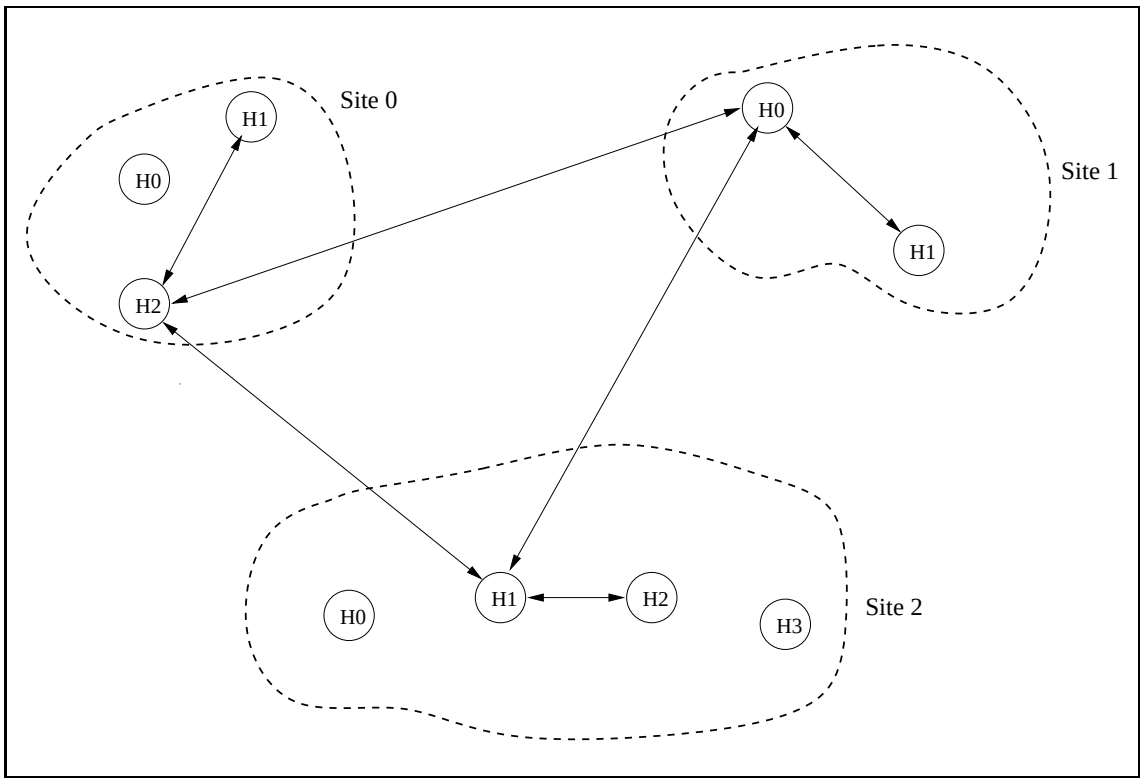


Figura 4.8: Modelagem da comunicação entre alguns hosts.

tura dos arquivos de entrada, pelo recebimento dos parâmetros modelados e pela geração do arquivo de saída. Todos os processos têm a função de modelar o fator de heterogeneidade dos *hosts* onde são executados e somente alguns dos processos são responsáveis pela modelagem da comunicação. Ao final de cada modelagem de comunicação entre um par de *hosts* do mesmo *site* ou de um par de *hosts* de *sites* diferentes os parâmetros são enviados ao processo coordenador.

4.3.1 Modelagem da capacidade de processamento disponível

Nessa versão, outros trechos de código, com estruturas mais complexas do que o código utilizado na versão I do modelador, foram testados para a modelagem do fator de heterogeneidade dos *hosts*. A primeira versão baseava-se em um laço contado que realizava a cada iteração uma operação simples de incremento de uma variável. Foi observado que, para um aumento do número de iterações realizadas o fator de heterogeneidade aumentava proporcionalmente. Porém, para um código

de aplicação diferente do código usado para a modelagem, considerando outras operações além da operação de incremento, variações no fator de heterogeneidade não eram proporcionais a variações no código da aplicação.

A segunda versão de trecho de código utilizada considerava a multiplicação de matrizes de tamanho fixo. Neste caso, foi observado que o fator de heterogeneidade não era sensível a um aumento no número de operações realizadas aumentando-se o tamanho das matrizes, muito provavelmente por otimizações do compilador.

O terceiro trecho de código utilizado implementa uma versão do algoritmo de eliminação gaussiana. Foi observado que o fator de heterogeneidade é sensível a variações no volume de operações realizadas e que variações no fator de heterogeneidade obtido eram bastante proporcionais a variações no tamanho do código de outras aplicações.

A versão II do modelador utiliza uma combinação dos três trechos de código verificados. Essa decisão foi tomada pelo fato de que os três trechos de código possuem características distintas, representando melhor situações reais de códigos de aplicações científicas.

4.3.2 Resultado da modelagem da capacidade de processamento disponível

Os testes para verificação da precisão do mecanismo de modelagem do fator de heterogeneidade foram realizados em um ambiente controlado, mantendo os *hosts* com carga zero durante a modelagem, e também variando-se a carga dos *hosts*, executando em cada um deles instâncias de um processo que consome 100% da CPU. Os fatores obtidos, vistos na Tabela 4.13, retratam as diferenças entre os *hosts* do ponto de vista de suas características estáticas, e os fatores obtidos, vistos na Tabela 4.14, mostram a sensibilidade do sistema de modelagem também às características dinâmicas dos *hosts*, onde podemos verificar o aumento dos fatores de heterogeneidade em função da carga gerada pelos processos em execução em cada *host*. O

número de processos consumidores que é executado em cada *host* para simular a carga é o mesmo usado para a versão I do modelador, ou seja, *host* sn00 executa duas instâncias do *processo consumidor*, o *host* sn01 três instâncias, o *host* sn02 seis instâncias, o *host* caju oito instâncias, o *host* sn04 dez instâncias e o o *host* jenipapo doze instâncias.

Host	Carga	Tam. Fila	Perc. CPU Livre	Fator H
sn00	0	1	100	26
sn01	0	1	100	26
sn02	0	1	100	26
sn04	0	1	100	26
jenipapo	0	1	100	40
caju	0	1	100	42
sn16	0	1	100	26
sn17	0	1	100	26
n19	0	1	100	40
n20	0	1	100	40
n21	0	1	100	40
n22	0	1	100	40

Tabela 4.13: Fatores Modelados com carga 0.

Na Tabela 4.13 os valores dos fatores obtidos para os *hosts* modelados é maior do que os modelados pela versão I do modelador em função da diferença do código utilizado pelo modelador.

Host	Carga	Num. Processos Consumidores	Tam. Fila	Perc. CPU Livre	Fator H (%)
sn00	1,97	2	3	0	50
sn01	4,02	4	5	0	102
sn02	7,38	8	9	0	200
sn04	11,02	10	11	0	251
jenipapo	2,20	0	3	0	94
caju	1,03	0	2	0	100
sn16	0,00	0	1	100	26
sn17	0,00	0	1	100	26
n19	2,20	0	3	0	120
n20	2,10	0	4	0	121
n21	2,08	0	4	0	122
n22	1,97	0	4	0	122

Tabela 4.14: Fatores Modelados com carga diferente de 0.

4.3.3 Modelagem da Comunicação

Além da mudança no que se refere ao número de medidas realizadas entre os *hosts*, uma outra mudança com relação ao primeiro método foi a definição de um novo algoritmo para a modelagem da comunicação entre pares de *hosts*. Esse novo

algoritmo é mais simples, baseando-se no conceito de ping-pong, onde uma mensagem é enviada ao destino que imediatamente responde à origem, como ilustrado na Figura 4.9. Esse algoritmo simplifica a abordagem de Kielmann *et al.* utilizada na versão I do modelador. A principal diferença é que a determinação do parâmetro g é eliminanda. O gap tem pouca influência no custo de comunicação quando se utiliza rotinas de comunicação baseadas em software como, por exemplo, MPI [27] (como foi observado também na versão I do modelador). Neste método, os parâmetros são medidos em uma única fase onde um conjunto de $\boldsymbol{n}$ mensagens de tamanho $\boldsymbol{m}$ são enviadas pelo processo *measure* e retornadas pelo processo *mirror*. A cada mensagem enviada e recebida pelo processo *measure* os valores medidos são acumulados e ao final o valor de cada parâmetro é calculado pela média.

Método Ping-pong

O método tradicional ping-pong, é formado por duas partes. A primeira chamada *measure* é responsável pelo início e coordenação do processo de medida. A segunda, denominada *mirror*, é responsável por responder à primeira. Durante cada medida o processo *measure* envia uma requisição, (a) na Figura 4.9, para o processo *mirror* que retorna, quando pronto, uma mensagem ACK (b) para o processo *measure* e em seguida executa o comando MPI-Probe para aguardar o início do processo de medida dos parâmetros. Quando o processo *measure* recebe o ACK do processo mirror ele envia uma mensagem (c) de tamanho $\boldsymbol{m}$ para o *mirror*, computa a sobrecarga de envio e em seguida executa o comando MPI-Probe para aguardar a mensagem de retorno. Quando a mensagem (d) chega ao processo *mirror*, este a recebe computando a sobrecarga de recebimento e em seguida envia a mensagem de volta ao processo *measure*, computando a sobrecarga de envio. O processo *measure* ao receber a mensagem de volta computa a sobrecarga de recebimento e o *Round Trip Time* (RTT). Após a computação das sobrecargas e do RTT o processo *measure* recebe do processo *mirror* suas sobrecargas computadas e então a latência pode ser calculada pela expressão $[\boldsymbol{L(m, measure, mirror)} = (\boldsymbol{(RTT(m, measure, mirror)} - \boldsymbol{Os(m, measure, mirror)} - \boldsymbol{Or(m, mirror, measure)} - \boldsymbol{Os(m, mirror,}$

$measure) - Or(m, measure, mirror))/2)]$.

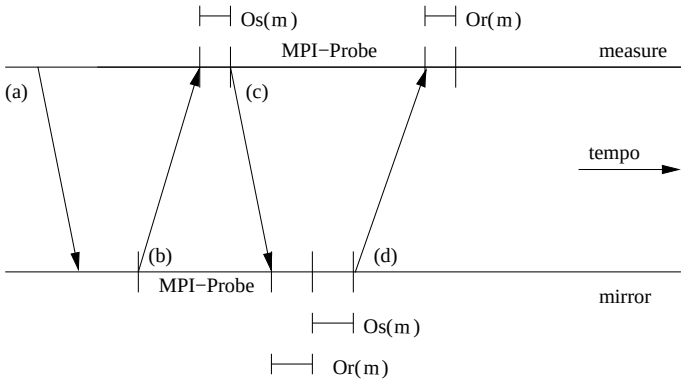


Figura 4.9: Método ping-pong.

4.3.4 Resultados da modelagem da comunicação

O método de Kielman utiliza, para o envio das mensagens durante o processo de modelagem da comunicação, o comando MPI_Send(). Este comando é dito bloqueante, pois o processo que envia a mensagem aguarda uma mensagem de confirmação de recebimento por parte do destinatário, antes de liberar o processador. Na versão II do modelador foi considerada também a utilização do comando de envio de mensagens de forma não bloqueante MPI_Isend(). Este comando não aguarda o recebimento da mensagem pelo destinatário, mas simplesmente delega a um outro processo a função de enviar, liberando o processo que realizou a chamada para continuar executando. As Tabelas 4.15, 4.16 e 4.17 apresentam os valores obtidos para a latência, a sobrecarga de envio e a sobrecarga de recebimento, para vários tamanhos de mensagem, utilizando-se tanto o comando MPI_Send() como o comando MPI_Isend(). A Tabela 4.15 apresenta os valores para a comunicação realizada entre um mesmo *host*, a Tabela 4.16 apresenta os valores para a comunicação realizada entre dois *hosts* da UFF e a Tabela 4.17 apresenta os valores para a comunicação realizada entre um host da UFF e outro da PUC-Rio.

Comparando os dados das Tabelas 4.15, 4.16 e 4.17 , referentes ao comando MPI_Send() modelados com a versão II do modelador, com os dados das Tabelas

Tam. da Mensagem (Bytes)	Usando Send()			Usando Isend()		
	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)
64	9.960	24	2	99.983	12	3
128	9.966	23	3	99.970	11	3
256	16.632	24	3	99.979	14	3
512	19.966	25	3	99.971	13	4
1.024	63.288	33	3	99.980	15	4
2.048	150.052	34	4	99.970	15	4
4.096	99.852	38	6	99.970	15	6
8.192	99.947	33	8	116.629	20	12
16.384	99.936	43	12	114.961	11	16
32.768	99.906	61	24	99.951	13	26
65.536	99.850	99	57	99.935	13	60
131.078	99.707	177	133	99.860	23	128
262.144	99.998	199.665	332	299.612	32	353
524.288	60.224	469.167	741	499.280	39	744
1.048.576	83.421	999.014	1.453	1.112.116	39	1.461
2.097.152	99.990	2.228.035	2.872	2.327.556	37	2.872
4.194.304	99.992	4.595.119	5.709	4.782.963	47	5.720
8.388.608	99.985	9.336.981	11.439	9.464.368	50	11.427
16.777.216	60.227	18.717.065	22.898	18.807.181	57	22.868
33.554.432	63.507	37.774.983	45.670	37.831.473	51	46.543
67.108.864	93.375	75.560.559	124.418	75.714.793	65	150.020

Tabela 4.15: Modelagem da comunicação pela versão II do Modelador usando Send() e Isend() no mesmo host.

4.8, 4.9 e 4.10, respectivamente, modelados com a versão I do modelador, observamos que os valores para a sobrecarga de envio **Os** e a sobrecarga de recebimento **Or** são mais ou menos equivalentes, mostrando que o mecanismo de modelagem para esses parâmetros na versão II do modelador possui a mesma precisão em relação à versão I do modelador. A diferença entre os valores da latência **L**, pode ser explicada em função da diferença sutil na sua definição (e então nos mecanismos de determinação). No método de Kielman, utilizado na versão I do modelador, a latência é o tempo entre o início de envio até a chegada do primeiro bit no destino (sendo calculada em função do *gap* g). No método proposto na versão II do modelador a latência é o tempo entre o fim da sobrecarga de envio e o início da sobrecarga de recebimento (sendo calculada em função do Round Trip Time (RTT) e das sobrecargas de envio e de recebimento). Neste último caso, como o comando de envio é bloqueante, as sobrecargas aumentam de forma linear com o aumento do tamanho da mensagem, mantendo a latência praticamente constante.

Observando os dados das Tabelas 4.16 e 4.17, referentes a testes de comunicação entre hosts diferentes utilizando o comando MPI_Send, verificamos que os valores para a sobrecarga de envio **Os** e a sobrecarga de recebimento **Or** têm

Tam. da Mensagem (Bytes)	Usando Send()			Usando Isend()		
	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)
64	247	18	1	197	6	1
128	251	19	2	199	7	1
256	253	19	2	197	7	1
512	219	19	3	241	6	1
1.024	477	19	2	312	6	3
2.048	488	22	2	456	7	2
4.096	732	26	4	626	6	4
8.192	1.004	31	4	958	8	5
16.384	1.557	361	7	1.654	7	9
32.768	2.948	492	17	3.032	8	17
65.536	3.620	2.588	48	5.836	12	49
131.078	4.720	6.790	156	11.430	18	150
262.144	4.097	18.662	364	22.547	27	349
524.288	4.381	40.356	795	44.855	34	750
1.048.576	4.450	84.577	1.533	89.383	35	1.469
2.097.152	4.663	174.473	2.960	178.534	36	2.940
4.194.304	4.343	352.803	5.944	356.819	37	5.868
8.388.608	4.246	708.995	11.967	713.909	42	11.619
16.777.216	4.163	1.478.664	23.764	1.454.667	46	23.164
33.554.432	4.279	2.930.064	48.160	2.906.328	47	46.954
67.108.864	4.615	5.865.085	94.206	5.813.260	48	141.518

Tabela 4.16: Modelagem da comunicação pela versão II do Modelador usando Send() e Isend() entre hosts diferentes.

o mesmo comportamento dos valores valores obtidos para medidas entre processos sendo executados num mesmo *host*, visto na Tabela 4.15.

Observando em cada uma das Tabelas 4.16 e 4.17, verificamos que os valores da latência **L**, para tamanhos de mensagem inferiores a 64 Kbytes, considerando o comando MPI_Send, tem um valor da mesma ordem dos valores obtidos utilizando o comando MPI_Isend. Para tamanhos de mensagem maiores do que 64 KBytes, os valores modelados utilizando o comando MPI_Send são diferentes dos valores modelados pelo comando MPI_Isend pois, o comando MPI_Send somente é concluído após o recebimento de uma mensagen de retorno do *host* destino confirmando o recebimento, ocasionando assim uma sobreposição dos processos de envio e de comunicação. Como o valor da latência é calculado como a metade da diferença entre o Round Trip Time (RTT) e as sobrecargas de envio e de recebimento, visto na Subseção 4.3.3; e o valor das sobrecargas aumentam proporcionalmente em relação ao tamanho da mensagem, o valor da latência fica constante para tamanhos de mensagens superiores a 64 KBytes. Para as sobrecargas de envio **Os**, observamos que o valor para o comando MPI_Send é da mesma ordem do valor obtido através do comando MPI_Isend para: mensagens de tamanhos inferiores a 64 KBytes,

Tam. da Mensagem (Bytes)	Usando Send()			Usando Isend()		
	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)	Latência (μs)	Sob.Env. (μs)	Sob.Receb. (μs)
64	4.966	18	4	4.365	7	4
128	4.554	18	4	4.556	7	4
256	4.888	18	5	4.641	6	4
512	5.965	18	5	7.280	6	5
1.024	8.301	18	6	7.645	6	5
2.048	11.874	23	6	9.878	7	6
4.096	12.938	25	14	12.101	7	15
8.192	18.341	30	17	18.064	8	18
16.384	24.316	6.366	30	84.621	10	39
32.768	41.586	8.377	80	69.276	13	71
65.536	74.155	24.745	183	101.230	16	167
131.078	68.931	145.245	422	216.046	22	406
262.144	73.653	306.597	844	389.221	31	854
524.288	82.263	707.151	1.729	769.653	35	1.688
1.048.576	87.785	1.553.781	3.452	1.582.665	38	3.400
2.097.152	72.425	3.200.162	6.861	3.218.569	39	6.827
4.194.304	141.103	6.611.145	13.598	6.159.065	36	13.623
8.388.608	86.251	13.099.198	27.188	12.566.757	43	27.210
16.777.216	121.014	26.949.215	54.372	25.357.812	43	54.460
33.554.432	77.147	54.022.399	108.394	49.709.811	48	108.124
67.108.864	64.249	109.097.179	217.607	101.605.148	49	232.972

Tabela 4.17: Modelagem da comunicação pela versão II do Modelador usando Send() e Isend() entre hosts pertencentes a *sites* diferentes (UFF e PUC).

no caso de mensagens enviadas entre processos de um mesmo *host* (Tabela 4.15) e mensagens de tamanhos inferiores a 8 KBytes, no caso de mensagens enviadas entre processos sendo executados em *hosts* diferentes. Para mensagens maiores do que os tamanhos citados, os valores obtidos para o comando MPI_Send são próximos dos valores obtidos para o comando MPI_Isend.

Com relação ao valor das sobrecargas de recebimento **Or** associada ao comando MPI_Receive, observamos que os valores obtidos com MPI_Send são basicamente da mesma ordem aos obtidos utilizando MPI_Isend, podendo ser considerados os mesmos.

Or./Des.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		**6.126**	6.180	6.250	**79.796**	78.092	**124.914**	121.555	148.364	131.634
sn04	5.948		6.073	6.088	79.948	78.869	142.190	124.986	108.932	133.560
jenipapo	6.044	6.203		5.869	77.023	82.454	122.008	149.139	175.285	202.375
caju	5.969	5.959	5.862		81.106	79.183	150.393	112.633	171.447	114.180
sn16	**75.580**	83.717	76.888	82.525		**1.076**	**108.860**	155.724	151.584	138.501
sn17	77.333	77.043	78.772	82.895	1.023		121.290	125.458	130.242	107.133
n19	**131.723**	149.797	137.502	110.589	**121.770**	162.608		**5.868**	5.870	5.866
n20	139.973	118.607	130.764	124.060	127.528	122.964	5.852		5.875	5.871
n21	116.662	124.117	132.676	123.051	134.249	118.213	5.858	5.863		5.878
n22	114.495	137.422	135.882	130.054	111.265	108.938	5.854	5.859	5.863	

Tabela 4.18: Latências modeladas pela versão II do Modelador.

As Tabelas 4.18, 4.19 e 4.20 mostram os resultados das modelagens para a la-

Ori./Dest.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		**20**	18	18	**19**	18	**18**	22	20	20
sn04	21		19	19	20	19	22	20	22	21
jenipapo	71	69		71	71	72	74	75	76	77
caju	35	32	32		42	42	36	45	39	40
sn16	**19**	18	17	18		**18**	**18**	20	20	19
sn17	24	20	20	21	19		20	21	20	19
n19	**53**	52	51	51	**46**	44		**47**	45	46
n20	49	45	44	45	45	46	44		47	45
n21	49	50	49	53	52	48	46	48		47
n22	52	51	48	49	49	49	51	49	50	

Tabela 4.19: Sobrecargas de envio modeladas pela versão II do Modelador.

Ori./Dest.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		**60**	51	53	**57**	61	**58**	57	60	61
sn04	94		50	56	51	56	54	56	54	59
jenipapo	281	289		286	281	288	293	274	273	281
caju	173	176	172		224	231	204	258	213	223
sn16	**58**	58	58	58		**50**	**60**	63	62	60
sn17	63	62	58	62	58		63	65	61	62
n19	**147**	148	146	155	**162**	165		**159**	157	154
n20	154	154	145	144	146	145	142		135	134
n21	144	145	143	143	148	146	140	140		136
n22	138	134	133	135	136	137	133	132	131	

Tabela 4.20: Sobrecargas de recebimento modeladas pela versão II do Modelador.

tência, sobrecarga de envio e sobrecarga de recebimento, respectivamente, utilizando-se a versão II do modelador. Neste caso, os valores considerados são os obtidos usando-se o comando MPI_Isend() do MPI. Nas tabelas, observamos que os valores dos parâmetros referentes à comunicação entre *hosts* de um mesmo *site* ou entre *hosts* de *sites* diferentes têm valores semelhantes. Com base nessa observação, foi feita uma simplificação na versão II do modelador, denominada versão II(b), onde somente parte dos *hosts* do Grid a ser modelado participam da modelagem. Neste caso, são considerados dois *hosts* de cada *site*, que são responsáveis pela medida dos valores dos parâmetros para comunicações *intra-site*. Para medida dos valores para a comunicação *inter-sites*, o primeiro *host* determinado para cada *site* é escolhido, sendo responsável pela medida dos valores dos parâmetros para a comunicação entre seu *site* e os demais *sites*. Os valores destacados em negrito nas Tabelas 4.18, 4.19 e 4.20, mostram os valores que seriam considerados como sendo os valores dos parâmetros de comunicação para cada *site* e entre cada *site*.

Essa consideração acarreta em erros para os parâmetros de comunicação envolvendo os demais *hosts* do Grid, como pode ser visto nas Tabelas 4.21, 4.22 e 4.23. Para a latência, Tabela 4.21, os erros obtidos pela simplificação são pequenos

quando a comunicação envolve *hosts* de um mesmo *site*. Já, para a comunicação entre *hosts* de *sites* diferentes, os erros apresentam uma variação maior, em função de variações no tráfego através dos links que interligam os *sites*

Para o caso das sobrecargas de envio e de recebimento, Tabelas 4.22 e 4.23, observamos que os erros obtidos para valores referentes a comunicações entre *hosts* de um mesmo *site*, onde os *hosts* são heterogêneos (identificados através das informações do MDS), são muito altos. Neste caso, são necessárias medidas envolvendo todos os hosts diferentes de um mesmo *site* para que os valores dos parâmetros sejam corretos.

Ori./Dest.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		0,00	-0,88	-2,02	0,00	2,14	-0,08	2,61	-18,87	-5,46
sn04	2,32		0,87	0,62	-0,19	1,16	-13,92	-0,14	12,72	-7,01
jenipapo	1,34	-1,26		4,20	3,48	-3,33	2,25	-19,49	-40,44	-62,14
caju	2,56	2,73	4,31		-1,64	0,77	-20,49	9,76	-37,36	8,52
sn16	0,00	-10,77	-1,73	-9,19		0,00	0,00	-43,05	-39,25	-27,23
sn17	-2,32	-1,94	-4,22	-9,68	4,93	-11,42	-15,25	-19,64	1,59	
n19	0,00	-13,72	-4,39	16,04	0.00	-33,54		0,00	-0,03	0,03
n20	-6,26	9,96	0,73	5,82	-4,73	-0,98	0,27		-0,12	-0,05
n21	11,43	5,77	-0,72	6,58	-10,25	2,92	0,17	0,09		-0,17
n22	13,08	-4,33	-3,16	1,27	8,63	10,54	0,24	0,15	0,09	

Tabela 4.21: Porcentagem de erro nas medidas de latência ao se utilizar a versão II(b) do Modelador.

Ori./Dest.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn02		0,00	10,00	10,00	0,00	5,26	0,00	-22,22	-11,11	-11,11
sn04	-5,00		5,00	5,00	99,97	0,00	-22,22	-11,11	-22,22	-16,67
jenipapo	-255,00	-245,00		-255,00	-273,68	-278,95	-311,11	-316,67	-322,22	-327,78
caju	-75,00	-60,00	-60,00		-121,05	-121,05	-100,00	-150,00	-116,67	-122,22
sn16	0,00	5,26	10,53	5,26		0,00	0,00	-11,11	-11,11	-5,56
sn17	-26,32	-5,26	-5,26	-10,53	-5,56		-11,11	-16,67	-11,11	-5,56
n19	0,00	1,89	3,77	3,77	0,00	4,35		0,00	4,26	2,13
n20	7,55	15,09	16,98	15,09	2,17	0,00	6,38		0,00	4,26
n21	7,55	5,66	7,55	0,00	-13,04	-4,35	2,13	-2,13		0,00
n22	1,89	3,77	9,43	7,55	-6,52	99,96	-8,51	-4,26	-6,38	

Tabela 4.22: Porcentagem de erro nas medidas de sobrecargas de envio ao se utilizar a versão II(b) do Modelador.

A versão II(b) do modelador gera uma intrusão menor no Grid, pois o número de mensagens trocadas durante a fase de modelagem é proporcional ao número de *sites* e não ao número de *hosts*, como nas versões I e II(a). Outra característica importante da versão II(b) é a maior escalabilidade; somente quando um novo *site* fizer parte do Grid novas mensagens deverão ser trocadas para a modelagem. Como o modelador II(b) executa um número menor de operações para a

Ori./Dest.	sn02	sn04	jenipapo	caju	sn16	sn17	n19	n20	n21	n22
sn16		0,00	15,00	11,67	0,00	-7,02	0,00	1,72	-3,45	-5,17
sn17	-56,67		16,67	6,67	99,94	1,75	6,90	3,45	6,90	-1,72
jenipapo	-368,33	-381,67		-376,67	-392,98	-405,26	-405,17	-372,41	-370,69	-384,48
caju	-188,33	-193,33	-186,67		-292,98	-305,26	-251,72	-344,83	-267,24	-284,48
sn16	0,00	0,00	0,00	0,00		0,00	0,00	-5,00	-3,33	0,00
sn17	-8,62	-6,90	0,00	-6,90	-16,00		-5,00	-8,33	-1,67	-3,33
n19	0,00	-0,68	0,68	-5,44	0,00	-1,85		0,00	1,26	3,14
n20	-4,76	-4,76	1,36	2,04	9,88	10,49	10,69		15,09	15,72
n21	2,04	1,36	2,72	2,72	8,64	9,88	11,95	11,95		14,47
n22	6,12	8,84	9,52	8,16	16,05	99,89	16,35	16,98	17,61	

Tabela 4.23: Porcentagem de erro nas medidas de sobrecargas de recebimento ao se utilizar a versão II(b) do Modelador.

modelagem, seu tempo de execução é inferior ao das versões II(a) e I. Para o ambiente utilizado nos testes, o tempo de modelagem de cada versão foi de **5.555.277 μs** para a versão II(b), **150.086.088 μs** para a versão II(a) e **343.040.417 μs** para a versão I. A versão II(b), é cerca de 27 vezes mais rápida do que a versão II(a) e 61 vezes mais rápida do que a versão I.

Os gráficos da Figura 4.10, são referentes aos parâmetros de comunicação obtidos através da versão II(b) do modelador, utilizando o comando MPI_Isend. Comparando-os com os obtidos pela versão I do modelador, Figura 4.6, observamos que, por causa do uso do comando MPI_Isend, o parâmetro de sobrecarga de envio tem um comportamento diferente do obtido pela versão I modelador, seu valor cresce até um certo tamanho de mensagem e partir desse ponto permanece aproximadamente constante para tamanhos de mensagens superiores.

4.3.5 Verificação da versão II do Modelador

A verificação dos parâmetros modelados foi feita utilizando-se a mesma aplicação distribuída utilizada para a verificação da versão I do modelador. Os valores dos parâmetros foram coletados através do modelador e o cálculo da previsão do tempo de execução da aplicação, baseados nesses valores, são apresentados na Tabela 4.24. Comparando-se o tempo estimado com o tempo real de execução da aplicação, podemos verificar, nas Tabelas 4.25 e 4.26, que a simplificação do mecanismo de modelagem não prejudica a precisão dos parâmetros modelados.

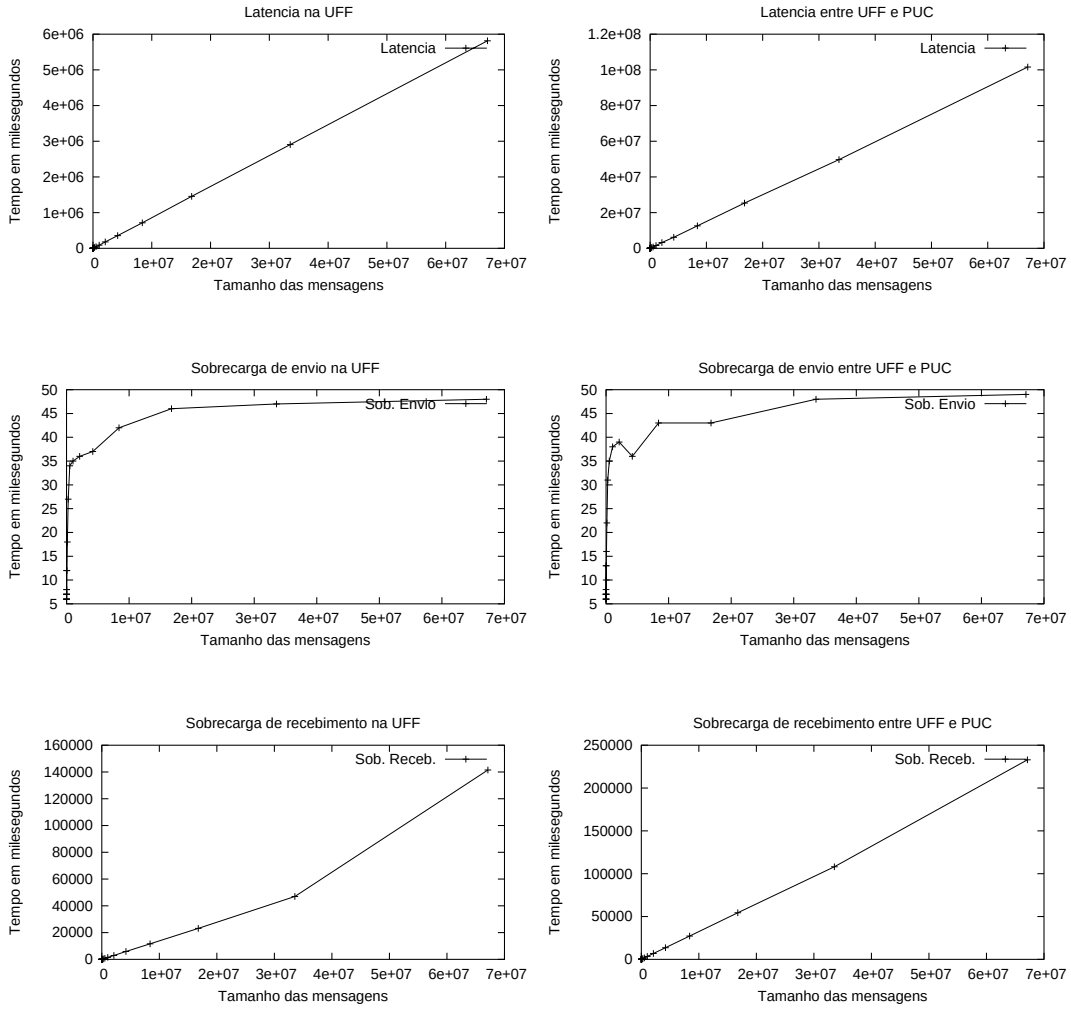


Figura 4.10: Gráficos de Comunicação Versão II do Modelador.

4.4 Modelador Analítico

As características de um ambiente Grid podem variar entre uma execução e outra de uma aplicação, devido ao compartilhamento dos recursos entre várias aplicações e também devido a variações de tráfego nos links da rede. Dessa forma, para se obter uma melhor eficiência em cada execução de uma aplicação, deve-se realizar o escalonamento da aplicação antes de cada execução. Como para a realização do escalonamento são necessárias informações atualizadas sobre os recursos, uma nova execução do modelador tem que ser feita, gerando uma maior intrusão no ambiente. Nessa seção apresentamos uma versão do modelador que se baseia em um modelo

Etapa	Versão II(a)		Versão II(b)	
	Tempo (μs) UFF-UFF	Tempo (μs) UFF-PUC	Tempo (μs) UFF-UFF	Tempo (μs) UFF-PUC
Início T0	0	0	0	0
Proc(T0)	5.133.257	5.133.257	5.133.257	5.133.257
Os(T0,T1)	20	18	20	18
L(T0,T1)	6.126	124.914	6.126	124.914
Or(T1,T0)	60	58	60	58
Proc(T1)	5.079.766	8.161.396	5.079.766	8.161.396
Os(T1,T2)	19	52	20	53
L(T1,T2)	6.073	149.797	6.126	131.723
Or(T2,T1)	51	148	60	147
Proc(T2)	5.057.399	5.057.399	5.057.399	5.057.399
Os(T2,T0)	71	21	20	20
L(T2,T0)	6.044	5.948	6.126	6.126
Or(T0,T2)	281	94	60	60
Fim T0	15.289.167	18.633.102	15.289.040	18.615.171

Tabela 4.24: Evolução do tempo estimado para a aplicação da Figura 4.7, com comunicação entre processos da UFF e entre processos da UFF e PUC, com base nos parâmetros coletados pela versão II(a) e II(b) do Modelador.

	Tempo Execução (μs)	Tempo Estimado (μs)	Erro (%)
UFF-UFF	15.320.844	15.289.167	**0,21**
UFF-PUC	19.021.120	18.633.102	**2,08**

Tabela 4.25: Comparação entre tempo de execução da aplicação e o tempo a partir dos parâmetros modelados, utilizando a versão II(a) do Modelador.

analítico para determinação dos valores dos parâmetros do modelo HLogP (latência, sobrecarga de envio e sobrecarga de recebimento).

Os valores dos parâmetros do modelo são estimados com base em equações propostas nas seções seguintes. A estimativa é feita com base em valores, denominados *parâmetros calibrados*, que são determinados uma única vez para cada *host* do Grid, utilizando a versão II(a) do modelador. As equações utilizam informações obtidas do sistema operacional, como por exemplo a carga e o tamanho da fila, entre outros. A versão II(b) do modelador é utilizada a cada escalonamento, para fazer ajustes nos *parâmetros calibrados* e determinar o valor aproximado para os parâmetros. O valor obtido representa as características atuais do recurso.

	Tempo Execução (μs)	Tempo Estimado (μs)	Erro (%)
UFF-UFF	15.320.844	15.289.040	**0,21**
UFF-PUC	19.021.120	18.615.171	**2,18**

Tabela 4.26: Comparação entre tempo de execução da aplicação e o tempo a partir dos parâmetros modelados, utilizando a versão II(b) do Modelador.

4.4.1 Modelagem da capacidade de processamento disponível

Nessa versão do modelador, o valor referente à capacidade de processamento de cada *host* é determinado utilizando uma das equações apresentadas abaixo (4.1, 4.2 e 4.3). Essas equações têm o objetivo de estimar o poder computacional disponível ($\boldsymbol{H_{Estimado}}$) em cada *host* do Grid, ajustando o fator de heterogeneidade calibrado ($\boldsymbol{H_{Calibrado}}$). O ajuste é feito com base em características coletadas do próprio sistema operacional do *host*. Três equações diferentes que analisam mudanças nas características dos *hosts* foram propostas neste trabalho:

$$\boldsymbol{H_{Estimado} = H_{Calibrado} * (\frac{TamFilaAtual}{TamFilaPadrao})} \tag{4.1}$$

$$\boldsymbol{H_{Estimado} = H_{Calibrado} * CargaAtual} \tag{4.2}$$

$$\boldsymbol{H_{Estimado} = H_{Calibrado} * (\frac{TempoParede}{TempoCPU})} \tag{4.3}$$

- *Equação 4.1 : TamFilaPadrão* é a quantidade de processos que estavam em execução quando o $\boldsymbol{H_{Calibrado}}$ do *host* foi determinado; e o *TamFilaAtual* é a quantidade atual de processos em execução no *host*.
- *Equação 4.2 : CargaAtual* é o valor atual coletado para a carga do *host*.
- *Equação 4.3 : TempoParede* é o tempo total que leva para executar um trecho de código no *host*; e o *TempoCPU* é o tempo de CPU utilizado para executar o trecho de código.

4.4.2 Resultado da modelagem da capacidade de processamento disponível

As Tabelas 4.27, 4.28 e 4.29 mostram os resultados obtidos através da modelagem utilizando as Equações 4.1, 4.2 e 4.3, respectivamente. Podemos verificar na Tabela 4.27, como visto por [45], que a modelagem analítica pode não apresentar boa precisão em alguns casos. Os valores calculados (Fator Estimado) para os hosts sn00, sn01 e sn16 são bastante diferentes dos valores obtidos através da modelagem utilizando a versão II do modelador (Fator Medido).

Host	Fila Padrao	Fator Calibrado	Fila Atual	Fator Estimado	Fator Medido	Erro (%)
sn00	1	26	3	78	54	44,44
sn01	1	26	2	52	40	30,00
sn02	1	26	1	26	25	4,00
sn04	1	26	1	26	25	4,00
jenipapo	1	40	1	40	40	0,00
caju	1	42	1	42	42	0,00
sn16	1	26	2	52	39	33,33
sn17	1	26	1	26	25	4,00
n19	1	42	6	252	221	14,02
sn20	1	42	6	252	222	13,51
sn21	1	42	6	252	224	12,50
sn22	1	42	6	252	223	13,00

Tabela 4.27: Fatores Modelados Analiticamente com a Equação 4.1.

Host	Carga Padrão	Fator Calibrado	Carga Atual	Fator Estimado	Fator Medido	Erro (%)
sn00	0,00	26	2,11	55	54	1,81
sn01	0,00	26	1,05	27	40	-48,14
sn02	0,00	26	0,05	26	25	4,00
sn04	0,00	26	0,07	26	25	4,00
jenipapo	0,00	40	0,06	40	40	0,00
caju	0,00	42	0,13	42	42	0,00
sn16	0,00	26	1,10	29	39	-34,48
sn17	0,00	26	0,06	26	25	4,00
n19	0,00	42	5,08	213	221	-3,75
n20	0,00	42	5,12	215	222	-3,25
n21	0,00	42	5,05	212	224	-4,24
n22	0,00	42	5,06	212	223	-5,18

Tabela 4.28: Fatores Modelados Analiticamente com a Equação 4.2.

A utilzação da Equação 4.1 que se baseia no tamanho da fila pode, como é o caso dos *hosts* sn00 e sn01, levar a uma previsão bastante errada. Isso acontece pelo fato de que às vezes, processos em execução, que não consomem demasiadamente o processador do *host*, podem elevar o número de processos ativos sem que ocorra um aumento do uso do processador.

Host	Fator Calibrado	Tempo Parede (μs)	Tempo CPU (μs)	Fator Calculado	Fator Real	Erro (%)
sn00	26	4.818.706	2.790.000	45	54	-19,90
sn01	26	3.644.149	2.800.000	34	40	-17,64
sn02	26	2.790.625	2.790.000	26	26	0,00
sn04	26	2.791.409	2.790.000	26	26	0,00
jenipapo	40	3.506.312	3.450.000	41	40	2,50
caju	42	4.026.773	3.810.000	44	42	4,76
sn16	26	3.933.088	2.800.000	36	39	-8,33
sn17	26	2.795.377	2.800.000	26	25	4,00
n19	42	24.105.993	4.490.000	225	221	1,81
n20	42	23.664.450	4.520.000	220	222	-1,00
n21	42	23.906.317	4.510.000	223	224	-0,50
n22	42	22.690.068	4.440.000	215	223	-3,72

Tabela 4.29: Fatores Modelados Analiticamente com a Equação 4.3.

A Tabela 4.28 mostra que a utilização da Equação 4.2 que se baseia no valor da carga, retornado pelo sistema operacional do *host*, que é o mesmo disponibilizado pelo MDS do Globus Toolkit, produz previsões não muito precisas em alguns casos. Isso pode ser explicado pelo fato de que o valor da carga ser medido em função do tamanho médio da lista de processos ativos do sistema. Podemos verificar, no caso do *host* sn04, que os valores do erro foram próximos para a previsão feita através das Equações 4.1, 4.2.

Os resultados apresentados nas Tabelas 4.27, 4.28 e 4.29 mostram que a utilização da equação 4.3 produz, na maioria das vezes, resultados com erros menores (14,4%, 9,4% e 5,3% de erro do valor absoluto da média, respectivamente).

4.4.3 Modelagem da Comunicação

A modelagem dos parâmetros de comunicação é feita utilizando-se funções que determinam os valores de cada parâmetro de comunicação do modelo HLogP. Dependendo do *site* onde se localizam os processadores que estão executando os processos que se comunicam, conjuntos diferentes de equações são utilizados. São determinados conjuntos de equações para comunicação em um mesmo *site* e para comunicação entre *sites* diferentes. Por exemplo, no caso do ambiente de testes onde existem três *sites* distintos, são definidos seis conjuntos de equações.

Cada função pode ter como parâmetro o tamanho da mensagem ($\boldsymbol{m}$), e/ou o processador de origem ($\boldsymbol{p_i}$) e/ou o processador de destino($\boldsymbol{p_j}$). O cál-

culo do valor atual de um parâmetro de comunicação pelo modelador analítico é feito determinando-se inicialmente qual conjunto de equações deve ser utilizado, em função da localização dos processadores envolvidos na comunicação. Em seguida, utilizando-se o tipo de parâmetro de comunicação desejado e o tamanho da mensagem, determina-se qual equação deverá ser utilizada e então calcula-se o valor do parâmetro.

As equações que modelam as funções de cálculo dos parâmetros de comunicação foram elaboradas a partir da análise dos gráficos referentes aos valores dos parâmetros, obtidos através da utilização da versão II(b) do modelador. Foram realizadas comunicações com tamanhos de mensagem variando de 100 em 100 bytes até um tamanho máximo igual a 2.000.000 bytes. A seguir são apresentadas as equações para o cálculo dos parâmetros de comunicação do modelo HLogP, para cada *site* pertencente ao Grid experimental onde foram realizados os testes.

Funções para o *site* UFF

A função para determinação da latência, quando a comunicação ocorre entre dois processos situados em *hosts* pertencentes ao *site* UFF, foi determinada com base na análise do gráfico da Figura 4.11, onde podemos observar um comportamento linear cujo gradiente da reta é igual a 0,0865. Assim, a latência para qualquer tamanho de mensagem pode ser determinada pela equação $\boldsymbol{latencia(m, pi, pj) = 0,0865 * (m - 64) + 314}$. As outras constantes da equação são obtidas através de medidas, onde o valor 314 se refere à latência para tamanho de mensagem igual a 64 bytes, obtido utilizando-se a versão II(b) do modelador.

A função para cálculo da sobrecarga de envio foi determinada com base na análise do gráfico da Figura 4.12. Neste caso, observamos que o gráfico tem um comportamento linear e crescente para tamanhos de mensagens inferiores a 128 kbytes, e para tamanhos de mensagens superiores, tem um comportamento linear e constante. Assim, a função para a sobrecarga de envio é modelada por equações, cuja utilização de uma ou outra depende do tamanho da mensagem. As equações são:

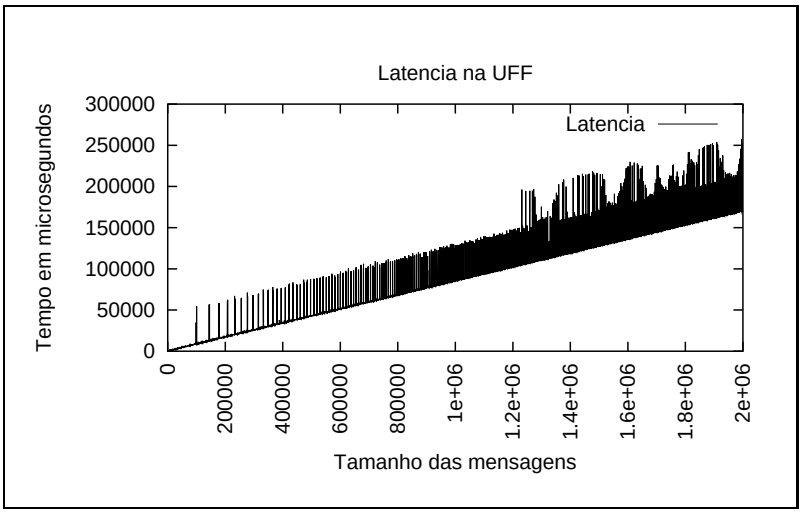


Figura 4.11: Latências para tamanho de mensagens variados obtidas da comunicação entre *hosts* do *site* 1 do GridRio.

$$Os(m, pi) = \begin{cases} 0,0001 * (m - 64) + 14 & \text{se} \quad m < 131.072 \\ 37 & \text{se} \quad m >= 131.072 \end{cases}$$

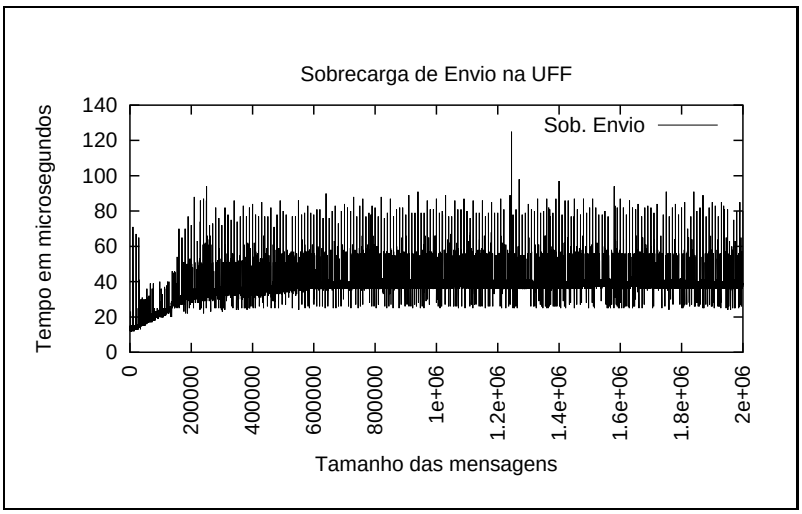


Figura 4.12: Equações para cálculo do valor da sobrecarga de envio no *site* UFF.

Para a sobrecarga de recebimento a função foi determinada com base na análise dos gráficos da Figura 4.13. Neste caso, observamos que o gráfico tem duas faixas com comportamentos diferentes. Para mensagens de tamanho inferior a 64 Kbytes, os gráficos têm um comportamento quadrático e para mensagens de ta-

manhos maiores que 64 Kbytes tem comportamento linear e crescente. Assim, as equações para sobrecarga de recebimento são:

$$Or(m, pj) = \begin{cases} 4,637 * 10^{-9} * m^2 + 1,023 * 10^{-3} * m + 1,9829 & \text{se} \quad m < 65536 \\ 0,00165 * (m - 65536) + 86 & \text{se} \quad m >= 65536 \end{cases}$$

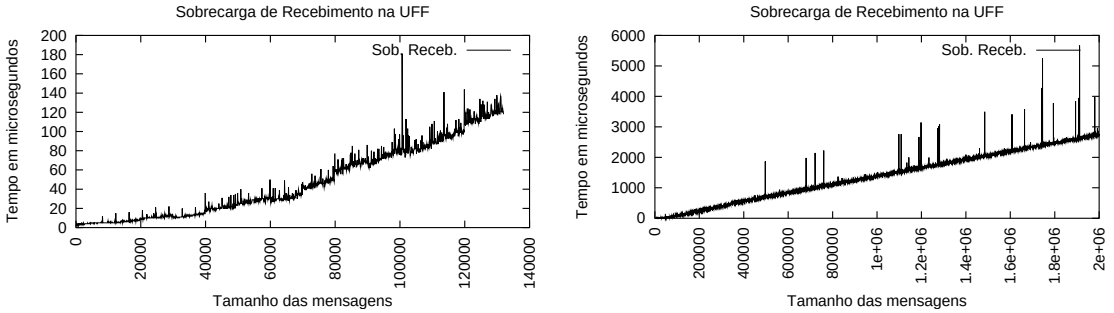


Figura 4.13: Equações para cálculo da sobrecarga de recebimento no site UFF.

Funções para os demais *sites*

Analisando os gráficos obtidos nos testes de comunicação para os demais *sites*, relacionados no Apêndice A, observamos que o comportamento é o mesmo em relação ao *site* da UFF. Neste caso, somente os valores das constantes são alterados em função de diferenças nas velocidades das redes. As Tabelas 4.30, 4.31 e 4.32 mostram as equações para os demais *sites* pertencentes ao GridRio.

site	Equação
site PUC	$latencia(m, pi, pj) = 0,0860 * (m - 64) + 192$
site Sinergia	$latencia(m, pi, pj) = 0,0116 * (m - 64) + 230$
entre UFF e Sinergia	$latencia(m, pi, pj) = 1,1337 * (m - 64) + 1123$
entre UFF e PUC	$latencia(m, pi, pj) = 1,6275 * (m - 64) + 4680$
entre Sinergia e PUC	$latencia(m, pi, pj) = 1,6275 * (m - 64) + 4680$

Tabela 4.30: Equações para o cálculo da latência para os demais *sites* do GridRio.

site	Equação
site PUC	$Os(m, pi) = 0,0003 * (m - 64) + 17$ m < 131.072 $Os(m, pi) = 56$ m $>= 131.072$
site Sinergia	$Os(m, pi) = 0,00005 * (m - 64) + 11$ m < 131.072 $Os(m, pi) = 32$ m $>= 131.072$
entre UFF e Sinergia	$Os(m, pi) = 0,0001 * (m - 64) + 13$ m < 131.072 $Os(m, pi) = 31$ m $>= 131.072$
entre UFF e PUC	$Os(m, pi) = 0,000046 * (m - 64) + 15$ m < 131.072 $Os(m, pi) = 33$ m $>= 131.072$
entre Sinergia e PUC	$Os(m, pi) = 0,000046 * (m - 64) + 15$ m < 131.072 $Os(m, pi) = 33$ m $>= 131.072$

Tabela 4.31: Equações para o cálculo da Sobrecarga de Envio para os demais *sites* do GridRio.

site	Equação
site PUC	$Or(m, pj) = 1,0434 * 10^{-8} * m^2 + 1,023 * 10^{-3} * m + 3,9829$ m < 65536 $Or(m, pj) = 0,00293 * (m - 65536) + 152$ m $>= 65536$
site Sinergia	$Or(m, pj) = 4,637 * 10^{-9} * m^2 + 1,023 * 10^{-3} * m + 2,9829$ m < 65536 $Or(m, pj) = 0,00185 * (m - 65536) + 93$ m $>= 65536$
entre UFF e Sinergia	$Or(m, pj) = 1,159 * 10^{-8} * m^2 + 1,023 * 10^{-3} * m + 2,9829$ m < 65536 $Or(m, pj) = 0,00200 * (m - 65536) + 111$ m $>= 65536$
entre UFF e PUC	$Or(m, pj) = 1,739 * 10^{-8} * m^2 + 1,023 * 10^{-3} * m + 4,9829$ m < 65536 $Or(m, pj) = 0,00304 * (m - 65536) + 131$ m $>= 65536$
entre Sinergia e PUC	$Or(m, pj) = 1,739 * 10^{-8} * m^2 + 1,023 * 10^{-3} * m + 4,9829$ m < 65536 $Or(m, pj) = 0,00304 * (m - 65536) + 131$ m $>= 65536$

Tabela 4.32: Equações para o cálculo da Sobrecarga de Recebimento para os demais *sites* do GridRio.

4.4.4 Resultado da modelagem da comunicação

As Tabelas 4.33, 4.34 e 4.35 mostram os resultados obtidos através da modelagem analítica, utilizando as funções apresentadas na subseção anterior, para o *site* UFF. Os valores dos parâmetros calculados, encontrados na coluna 3 das tabelas, foram coletados utilizando a versão II(b) do modelador. Podemos verificar também, como visto com NWS [45], que a precisão das informações calculadas a partir da modelagem analítica (na coluna 3), nem sempre é muito boa.

Tam.Mensagem (Bytes)	Latência (μs)	Latência Calculada (μs)	Erro (%)
64	314	314	0,13
128	286	320	-11,57
256	339	331	2,42
512	418	353	15,61
1.024	453	397	12,31
2.048	571	486	14,98
4.096	706	663	6,15
8.192	1.185	1.017	14,17
16.384	1.847	1.726	6,59
32.768	3.163	3.143	0,62
65.536	5.965	5.977	-0,21
131.072	11.594	11.646	-0,45
262.144	22.652	22.984	-1,47
524.288	44.981	45.659	-1,51
1.048.576	89.527	91.010	-1,66
2.097.152	178.651	181.712	-1,71
4.194.304	356.786	363.116	-1,77
8.388.608	713.286	725.923	-1,77
16.777.216	1.426.448	1.451.538	-1,76
33.554.432	2.940.215	2.902.767	1,27
67.108.864	5.792.056	5.805.225	-0,23

Tabela 4.33: Modelagem analítica da latência.

4.4.5 Verificação da versão analítica do modelador

A verificação dos parâmetros modelados foi feita de maneira igual à verificação das versões I e II do modelador, utilizando a mesma aplicação. Cada processo executa uma carga de trabalho duas vezes maior do que a utilizada pelo modelador para determinar o fator de heterogeneidade e o tamanho das mensagens enviadas é igual a 2.097.152 bytes. Os valores dos parâmetros foram obtidos utilizando as equações apresentadas, como pode ser visto na Tabela 4.36. A partir dos valores calculados, podemos estimar o tempo de execução da aplicação, como pode ser visto na Tabela 4.37, para o caso de execução da aplicação em *hosts* da UFF e em *hosts* da UFF e da PUC.

A comparação entre o tempo previsto e o tempo de execução real da aplicação, apresentados na Tabela 4.38, mostra que podemos chegar, a partir dos valores modelados, a um valor bastante próximo ao tempo de execução. A simplificação do mecanismo de modelagem não reduz substancialmente a precisão dos parâmetros modelados.

Tam.Mensagem (Bytes)	Sob. Envio Efetiva (μs)	Sob. Envio Calculada (μs)	Erro (%)
64	14	14	0,0
128	13	14	-7,7
256	17	14	15,6
512	15	14	7,7
1.024	13	14	-8,2
2.048	13	14	-12,1
4.096	13	14	-11,4
8.128	14	15	-5,2
16.384	20	15	25,5
32.768	16	16	-0,6
65.536	18	18	0,0
131.072	21	22	-4,7
262.144	34	37	-10,1
524.288	35	37	-6,3
1.048.576	37	37	0,5
2.097.152	38	37	1,6
4.194.304	37	37	0,0
8.388.608	36	37	-2,2
16.777.216	36	37	-3,4
33.554.432	44	37	15,5
67.108.864	39	37	4,1

Tabela 4.34: Modelagem analítica da Sobrecarga de envio.

4.4.6 Verificação da influência da carga na máquina sobre os parâmetros de comunicação do modelo HLogP

Analisando os resultados dos testes das versões I e II do modelador, apresentados nas seções anteriores, verificamos que os parâmetros de comunicação (latência, sobrecarga de envio e sobrecarga de recebimento) são sensíveis ao tamanho da mensagem e também do *host* que envia, no caso da sobrecarga de envio; do *host* que recebe, no caso da sobrecarga de recebimento; e do *host* que envia e do *host* que recebe, no caso da latência. Foram realizados alguns testes para verificar qual influência a carga gerada pela execução de outros processos nos *hosts* exerce sobre os valores dos parâmetros. Para a realização dos testes foram utilizados dois *hosts* da UFF de mesmas características, sendo que dois processos foram executados em cada *host* produzindo uma carga controlada. Em seguida, o programa modelador de comunicação para tamanhos variados de mensagens foi executado dez vezes. As Tabelas 4.39, 4.40 e 4.41 apresentam os valores médios dos parâmetros obtidos comparados com os valores obtidos pelo mesmo teste sendo executado sem que nenhum processo extra estivesse sendo executado nos *hosts*.

Na Tabela 4.39, verificamos que os valores da sobrecarga de envio para mensagens de tamanhos variados não sofre variação significativa quando os *hosts* envol-

Tam.Mensagem (Bytes)	S.R. Efetiva (μs)	Sob. Receb. Calculada (μs)	Erro (%)
64	3	4	-33,33
128	3	4	-33,33
256	3	4	-33,33
512	3	4	-33,33
1.024	4	4	0,00
2.048	4	4	0,00
4.096	11	12	-11,69
8.192	16	17	-7,5
16.384	19	17	7,53
32.768	39	43	-9,14
65.536	86	86	0,00
131.072	211	194	8,17
262.144	451	410	9,00
524.288	921	843	8,46
1.048.576	1.823	1.708	6,32
2.097.152	3.609	3.438	4,73
4.194.304	7.150	6.898	3,52
8.388.608	14.220	13.819	2,82
16.777.216	28.290	27.660	2,23
33.554.432	56.428	55.343	1,92
67.108.864	112.497	110.707	1,59

Tabela 4.35: Modelagem analítica da Sobrecarga de recebimento.

Funçao	UFF	UFF-PUC
Os(T0,T1)	37	33
L(T0,T1)	0,08650 * (2.097.152 - 64) + 314	1,62750 * (2.097.152 - 64) + 4.680
Or(T2,T1)	0,00165 * (2.097.152 - 65.536) + 86	0,00200 * (2.097.152 - 65.536) + 111
Os(T1,T2)	37	33
L(T1,T2)	0,08650 * (2.097.152 - 64) + 314	1,62750 * (2.097.152 - 64) + 4.680
Or(T1,T0)	0,00165 * (2.097.152 - 65.536) + 86	0,00200 * (2.097.152 - 65.536) + 111
Os(T2,T1)	37	37
L(T2,T1)	0,08650 * (2.097.152 - 64) + 314	1,08650 * (2.097.152 - 64) + 314
Or(T2,T1)	0,00165 * (2.097.152 - 65.536) + 86	0,00304 * (2.097.152 - 65.536) + 131
Proc T0	5.090.000 * (10.371.949/5.110.000)	5.090.000 * (10.371.949/5.110.000)
Proc T1	5.090.000 * (10.371.949/5.110.000)	5.090.000 * (10.371.949/5.110.000)
Proc T2	5.100.000 * (10.371.949/5.110.000)	5.090.000 * (10.371.949/5.110.000)

Tabela 4.36: Cálculo dos valores dos parâmetros do modelo HLogP pela versão analítica do modelador.

vidos no processo de comunicação (MPI_Isend) se encontram com outros processos em execução. As variações observadas ocorrem por serem os valores da sobrecarga de envio muito pequenos onde, neste caso, qualquer pequena variação durante a execução do comando de envio da mensagem se reflete no valor obtido.

Os valores da sobrecarga de recebimento praticamente sofrem pouca ou nenhuma influência da carga gerada nos *hosts* envolvidos no processo de comunicação, como pode ser observado na Tabela 4.40. Variações mais significativas foram observadas para mensagens de tamanhos maiores como pode ser observado para mensagem de tamanho 32 Mbytes e 64 Mbytes.

Etapa	Tempo (μs) UFF-UFF	Tempo (μs) UFF-PUC
Start time T0	0	0
Proc(T0)	10.331.554	10.331.554
Os(T0,T1)	37	35
L(T0,T1)	181.712	3.417.691
Or(T1,T0)	3.438	3.438
Proc(T1)	10.331.554	10.331.554
Os(T1,T2)	37	35
L(T1,T2)	181.712	3.417.691
Or(T2,T1)	3.438	6.307
Proc(T2)	10.351.651	10.331.554
Os(T2,T0)	37	35
L(T2,T0)	181.712	181.712
Or(T0,T2)	3.438	6.307
Finish time T0	31.549.625	38.027.918

Tabela 4.37: Evolução do tempo de execução estimado, com comunicação entre processos da UFF e entre processos da UFF e PUC.

	Tempo Execução (μs)	Tempo Estimado (μs)	Erro (%)
UFF-UFF	31.408.734	31.549.625	**0.45**
UFF-PUC	37.814.766	38.027.918	**0.56**

Tabela 4.38: Comparação entre tempo de execução da aplicação e o tempo a partir dos parâmetros modelados, utilizando a versão Analítica do Modelador.

Os testes mostraram que a latência, da maneira que foi calculada, sofre consideráve influência da carga nos *hosts* que estão se comunicando. Não foi possível neste trabalho identificar as causas dessa influência, sendo necessária uma investigação mais aprofundada.

4.5 Resumo do Capítulo

Neste capítulo foram apresentadas três versões para um mecanismo de coleta dos parâmetros do modelo LogP. As versões apresentadas diferem umas das outras em função da intrusão gerada pela execução das mesmas em um ambiente Grid (um fator importante em ambientes compartilhados com outras aplicações e usuários) e pelos mecanismos de coleta adotados em cada uma. A obtenção dos parâmetros do modelo, de forma eficiente, é de extrema importância para que escalonadores possam realizar o escalonamento de aplicações. O próximo capítulo aborda aspectos importantes sobre diferenças encontradas na implementação MPI-LAM da biblioteca

Tam.Mensagem (Bytes)	Sob. Envio sem carga (μs)	Sob. Envio com carga (μs)
64	13	13
128	12	15
256	13	13
512	12	13
1.024	12	17
2.048	11	15
4.096	14	13
8.128	13	15
16.384	14	15
32.768	15	16
65.536	17	17
131.072	21	24
262.144	29	30
524.288	38	32
1.048.576	56	37
2.097.152	39	29
4.194.304	56	32
8.388.608	41	29
16.777.216	41	37
33.554.432	27	39
67.108.864	53	49

Tabela 4.39: Influência de carga de processamento na sobrecarga de envio.

MPI com relação à implementação MPICH-G2. Em função dessas diferenças um modelo analítico também é proposto para a versão MPI-LAM.

Tam.Mensagem (Bytes)	Sob. Receb. sem carga (μs)	Sob. Receb. com carga (μs)
64	3	4
128	3	4
256	3	4
512	4	4
1.024	4	4
2.048	5	4
4.096	12	14
8.192	11	12
16.384	17	19
32.768	36	38
65.536	97	83
131.072	217	207
262.144	450	464
524.288	936	940
1.048.576	1.820	1.838
2.097.152	3.557	3.582
4.194.304	7.065	7.208
8.388.608	14.780	14.236
16.777.216	29.570	28.426
33.554.432	98.155	57.328
67.108.864	120.143	220.897

Tabela 4.40: Influência de carga de processamento na sobrecarga de recebimento.

Tam.Mensagem (Bytes)	Latência sem carga (μs)	Latência com carga (μs)
64	202	31.235
128	201	32.556
256	201	9.656
512	267	279
1.024	371	9.741
2.048	446	441
4.096	643	20.971
8.192	1.021	6.786
16.384	1.659	22.088
32.768	3.059	12.230
65.536	5.813	53.335
131.072	11.460	59.879
262.144	22.571	140.952
524.288	44.832	328.681
1.048.576	134.226	479.160
2.097.152	178.526	1.194.691
4.194.304	356.816	2.262.652
8.388.608	713.087	4.362.913
16.777.216	1.468.851	10.160.690
33.554.432	2.939.479	20.893.019
67.108.864	5.837.850	35.597.321

Tabela 4.41: Influência de carga de processamento na latência.

Capítulo 5

Modelo analítico para a implementação LAM do MPI

A partir da versão 7.0 em diante da implementação LAM do MPI (atualmente sendo desenvolvido na Universidade de Indiana, EUA), foi incluído o suporte para execução de aplicações paralelas MPI num Grid computacional que utiliza o Globus Toolkit. Este capítulo apresenta um modelo analítico do MPI-LAM para parte do ambiente Grid de teste utilizado (somente os nós dos *sites* UFF e Sinergia possuem essa versão instalada). As equações foram obtidas a partir da análise dos resultados das comunicações entre *hosts* dos *sites* utilizando a versão 7.0.4 do MPI-LAM.

5.1 MPICH-G2 versus MPI-LAM : O inesperado

No desenvolvimento da ferramenta de modelagem, sempre foi tomado o cuidado para que nenhuma decisão restringisse a ferramenta a uma implementação específica de MPI. Assim, o modelador poderia ser compilado para qualquer implementação de MPI disponível em um Grid, por exemplo, tanto para a versão MPICH-G2 como para a versão MPI-LAM. Na realização dos testes foi observada uma diferença importante entre as duas implementações em situações onde mais

de uma tarefa da mesma aplicação é executada numa mesma unidade de processamento. Para mensagens enviadas entre tarefas executadas na mesma unidade, a tarefa destino fica aguardando a mensagem enviada pela tarefa de origem, definindo uma ordem de execução entre ambas. Considerando a implementação MPICH-G2, enquanto a mensagem enviada não chega à tarefa destino, esta precisa, de tempos em tempos, verificar se a mensagem já está disponível no buffer de recebimento. Essa verificaçao influencia no tempo de execução das tarefas que estão sendo executadas na mesma unidade de processamento da tarefa destino, uma vez que compete pelos recursos do hardware como se fosse uma outra aplicação rodando no *host*. Este fato não foi observado na implementação MPI-LAM, onde a tarefa em espera não influencia no tempo de execução da tarefa que envia a mensagem.

Uma verificação do exposto acima foi feita através da execução do código de aplicações teste, representadas na Figura 5.1, cujos resultados do tempo de computação (sem considerar comunicação) de cada tarefa podem ser verificados na Tabela 5.1. Nos dois primeiros casos, as tarefas T0 e T2 foram executadas tanto em unidades de processamento distintas, Figura 5.1(a), como numa mesma unidade, Figura 5.1(b).

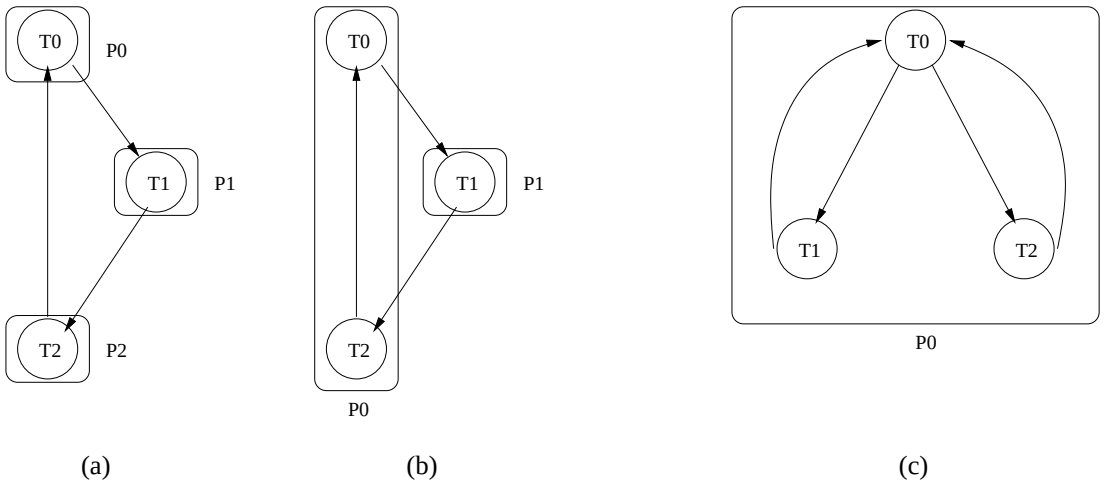


Figura 5.1: Aplicação para testar as implementações MPI.

Os resultados apresentados na Tabela 5.1, mostram que, para o caso da execução compilada para a versão MPICH-G2, o tempo de execução das tarefas

	MPICH-G2		MPI-LAM	
Tarefa	Fig. 5.1(a)	Fig. 5.1(b)	Fig. 5.1(a)	Fig. 5.1(b)
T0	695.600	1.296.882	693.768	689446
T1	696.192	694.694	691.104	689415
T2	691.385	996.237	689.600	690290

Tabela 5.1: Tempos de execução (em μs) dos processos da aplicação da Figura 5.1.

T0 e T2, quando executadas em um mesmo processador, são elevados com relação aos tempos das mesmas tarefas quando executadas em processadores distintos. Isso mostra que uma tarefa influencia no tempo de execução de outra quando executadas em um mesmo processador, mesmo que uma das tarefas esteja apenas aguardando mensagem. Já com relação à execução da mesma aplicação compilada para a versão MPI-LAM, os tempos de execução das tarefas T0 e T2 não se alteram quando são executados em um mesmo *host* ou em *hosts* diferentes, ou seja, a tarefa que aguarda mensagem não influencia o tempo de execução da tarefa que realiza processamento. A Figura 5.1(c) mostra 3 processos, cujos tempos de computação são em torno de **5** segundos, sendo executados num mesmo *host*. No MPI-LAM, o tempo total de execução foi de **15** segundos e no MPICH-G2 **31** segundos, ou seja, os três processos concorreram pela *cpu* do *host*.

Este resultado tem um impacto profundo no escalonamento de processos MPI. Quando a versão MPICH-G2 (o padrão em Grids computacionais) é utilizada, não deveria ser alocado mais de um processo por recurso (na prática difícil de evitar); ou, os escalonadores devem considerar esta concorrência (aumentando a complexidade dos mesmos significativamente).

5.2 Modelagem da comunicação na versão MPI-LAM

Foram realizados testes de calibragem dos parâmetros do modelo HLogP também utilizando a versão II(b) do modelador compilada com a versão MPI-LAM. Os testes foram executados apenas envolvendo *hosts* dos *sites* UFF e Sinergia, devido ao fato de que a versão MPI-LAM (6.5.7) nos *hosts* da PUC-Rio não têm suporte para o Globus Toolkit. Nos resultados, apresentados nas Tabelas 5.2 e 5.4,

observamos diferenças com relação aos resultados obtidos com a versão compilada utilizando o MPICH-G2

Tam. da	Usando Send()			Usando Isend()		
Mensagem	Latência	Sob.Env.	Sob.Receb.	Latência	Sob.Env.	Sob.Receb.
64	24	10	1	4	15	2
128	24	11	3	4	15	3
256	23	11	3	4	23	2
512	27	12	1	4	15	1
1.024	30	10	2	4	15	3
2.048	27	11	3	4	28	4
4.096	36	17	14	2	22	10
8.192	38	17	11	2	25	8
16.384	63	36	20	10	36	13
32.768	92	51	42	12	56	30
65.536	162	85	73	8	101	68
131.078	-382	466	468	-190	21	429
262.144	-742	756	757	-739	23	790
524.288	-1.436	1.394	1.390	-792	27	1.582
1.048.576	-2.785	2.633	2.632	-1.537	26	2.844
2.097.152	-5.545	5.630	5.581	-3.106	29	5.442
4.194.304	-10.733	10.993	10.945	-6.255	28	10.766
8.388.608	-21.079	21.512	21.461	-14.053	29	21.672
16.777.216	-41.760	43.174	43.125	-25.937	29	49.333
33.554.432	-83.958	85.376	85.323	-56.526	29	97.806
67.108.864	-169.118	172.013	171.962	-116.219	29	199.849

Tabela 5.2: Modelagem da comunicação, pela versão II do Modelador compilada com MPI-LAM, no mesmo host.

Na Tabela 5.2 observamos, para o caso do comando MPI_Send e mensagens de tamanho superior a 64 Kbytes, que a sobrecarga de envio é praticamente igual à sobrecarga de recebimento. Isso acontece pelo fato de que o protocolo utilizado pelo MPI-LAM para mensagens superiores a 64 Kbytes é bloqueante, levando a uma sobreposição quase completa entre os processos de envio e de recebimento. Isso explica tambem, os valores negativos obtidos para a latência que, neste caso, é calculada como sendo a metade do tempo de Round Trip Time (RTT) menos as sobrecargas de envio e de recebimento.

Considerando que o objetivo é servir ao processo de escalonamento de aplicações, a situação apresentada não se encaixa na definição do modelo, não podendo ser tratada pelos escalonadores. Do ponto de vista de escalonamento, não é possível que em um mesmo *host* duas ações (envio e recebimento) aconteçam ao mesmo tempo. O processador do *host*, hora atendende ao processo da tarefa de envio, hora atende ao processo da tarefa de recebimento. Para contornar esse problema, o cálculo da sobrecarga de envio, da latência e da sobrecarga de recebimento devem ser realizados de forma diferente com relação à versão II do modelador. A latência é

Tam. da	Usando Send()		
Mensagem	Latência	Sob.Env.	Sob.Receb.
64	24	10	1
128	24	11	3
256	23	11	3
512	27	12	1
1.024	30	10	2
2.048	27	11	3
4.096	36	17	14
8.192	38	17	11
16.384	63	36	20
32.768	92	51	42
65.536	162	85	73
131.078	0	467	467
262.144	0	757	757
524.288	0	1.392	1.392
1.048.576	0	2.633	2.633
2.097.152	0	5.605	5.605
4.194.304	0	10.974	10.974
8.388.608	0	21.493	21.493
16.777.216	0	43.151	43.151
33.554.432	0	85.353	85.353
67.108.864	0	171.996	171.996

Tabela 5.3: Modelagem da comunicação pela versão II do Modelador compilada com MPI-LAM entre hosts diferentes.

considerada 0 para tamanhos de mensagem superiores a 64 KBytes e as sobrecargas de envio e de recebimento são calculadas dividindo-se o RTT por quatro, como pode ser visto nos esquemas da Figura 5.2. Assim, os valores dos parâmetros para este modelo são os relacionados na Tabela 5.3.

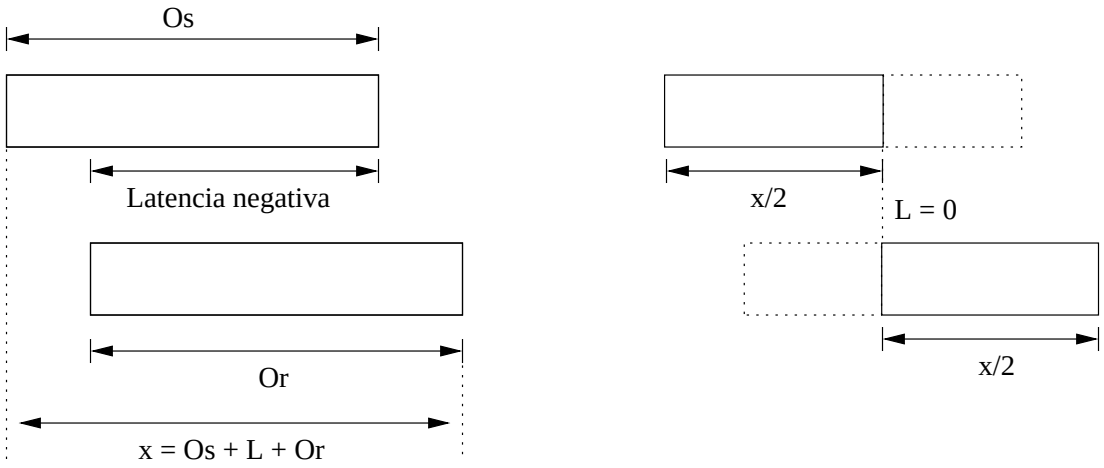


Figura 5.2: Ajuste no modelo para comunicação num mesmo *host*.

Comparando as Tabelas 5.2, 5.4 e 5.5, podemos verificar que o *comportamento* dos comandos de troca de mensagens da versão MPI-LAM é praticamente o mesmo para comunicações entre processos num mesmo *host*, processos em *hosts*

Tam. da	Usando Send()			Usando Isend()		
Mensagem	Latência	Sob.Env.	Sob.Receb.	Latência	Sob.Env.	Sob.Receb.
64	105	10	3	163	14	3
128	115	10	3	170	12	3
256	168	10	3	171	12	3
512	233	10	2	170	12	3
1.024	290	9	2	300	14	3
2.048	411	12	3	428	15	4
4.096	586	25	13	539	23	10
8.192	953	25	10	921	27	10
16.384	1.575	48	21	1.546	43	18
32.768	2.972	87	33	2.909	82	35
65.536	5.635	138	87	5.638	138	77
131.078	-3.727	3.731	11.498	121	21	11.464
262.144	-16.381	16.530	22.619	123	21	22.659
524.288	-38.494	38.752	44.855	150	25	44.921
1.048.576	-83.150	83.163	89.443	57	29	89.481
2.097.152	-172.028	171.673	178.772	88	34	178.564
4.194.304	-350.929	351.304	357.019	96	31	356.858
8.388.608	-706.024	705.820	713.142	175	36	713.045
16.777.216	-1.419.305	1.418.290	1.426.718	185	36	1.426.286
33.554.432	-2.845.311	2.846.431	2.852.703	169	36	2.852.143
67.108.864	-5.696.680	5.696.749	5.702.823	99	38	5.703.770

Tabela 5.4: Modelagem da comunicação, pela versão II do Modelador compilada com MPI-LAM, entre hosts diferentes.

diferentes e processos em *sites* diferentes. O mesmo não foi observado para a versão MPICH-G2, como pode ser visto nas Tabelas 4.15 e 4.16, 4.17.

Os resultados da modelagem da versão compilada para MPI-LAM, utilizando o comando MPI_Send, são diferentes dos obtidos utilizando a versão MPICH-G2, como pode ser observado pela comparação entre as Tabelas 4.15 e 5.2, 4.16 e 5.4, 4.17 e 5.5. Diante dessas diferenças, foi feita também uma modelagem analítica da versão compilada para MPI-LAM, utilizando o comando MPI_Send para comunicação. Neste caso, foram utilizados os *sites* UFF e Sinergia, pois somente nestes existe instalada a versão do MPI-LAM com suporte para o Globus. Nas seções seguintes são apresentadas as equações do modelo analítico obtidas pela análise dos resultados dos testes de comunicação realizados. Os dados dos gráficos foram obtidos executando-se o modelador para tamanhos variados de mensagens, sendo que o tamanho inicial foi de 64 bytes, o tamanho final de 2 MBytes e o intervalo do tamanho de mensagens de 100 bytes.

Tam. da	Usando Send()			Usando Isend()		
Mensagem	Latência	Sob.Env.	Sob.Receb.	Latência	Sob.Env.	Sob.Receb.
64	489	10	2	441	13	1
128	491	10	2	507	12	3
256	739	10	2	767	13	2
512	1.228	10	2	1.224	13	3
1.024	3.306	9	3	2.135	13	1
2.048	3.969	13	4	3.357	15	4
4.096	5.095	21	10	5.100	24	10
8.192	8.902	27	12	8.987	30	10
16.384	16.249	45	20	16.623	58	20
32.768	39.392	90	41	33.538	95	41
65.536	74.196	116	81	80.958	124	76
131.078	-85.428	74.939	131.047	1.044	22	13.1546
262.144	-283.774	277.466	318.685	5.931	21	330.173
524.288	-553.065	611.907	628.266	10.060	25	623.789
1.048.576	-1.339.373	1.474.802	1.596.420	3.068	37	1.759.344
2.097.152	-2.694.748	3.109.364	3.175.906	7.306	30	2.938.215
4.194.304	-5.490.174	5.609.431	5.866.209	20.255	28	6.017.091
8.388.608	-11.717.213	12.757.598	12.794.943	426	28	11.727.850
16.777.216	-22.313.911	23.882.397	23.909.425	3.888	30	26.079.555
33.554.432	-48.451.379	51.324.597	51.377.402	268	31	47.237.303
67.108.864	-100.594.107	115.538.468	115.603.214	4.141	34	101.350.440

Tabela 5.5: Modelagem da comunicação, pela versão compilada para MPI-LAM, entre hosts pertencentes a *sites* diferentes (UFF-Sinergia).

Funções para o *site* UFF

Devido ao comportamento diferente da latência para a versão MPI-LAM, observadas no gráfico da Figura 5.3, são necessárias duas equações. A primeira equação modela a comunicação para mensagens de tamanho inferior a 64 Kbytes, onde a latência cresce de forma linear com relação ao tamanho da mensagem. Para mensagens de tamanho superior a 64 Kbytes, a latência também tem um comportamento linear. Porém, o gradiente da reta tem valor negativo, e os valores das latências medidas são negativos. Esse valor negativo se deve ao fato de ocorrer uma sobreposição da sobrecarga de envio com a sobrecarga de recebimento e a latência ser calculada pela mesma equação utilizada na versão II do modelador vista no Capítulo 4. Como as sobrecargas se sobrepõem, o valor da soma das mesmas é maior do que o valor do RTT e, quando se faz a diferença entre o RTT e as sobrecargas o valor restante fica negativo. Como a latência é a metade desse valor a mesma também é negativa. As equações para determinação da latência para o caso do comando MPI_Send e a versão MPI-LAM da bilioteca de troca de mensagens MPI são:

$$L(m, pi, pj) = \begin{cases} 0,0940 * (m - 64) + 115 & \text{se} \quad m < 65.536 \\ -0,085 * (m - 65536) - 410 & \text{se} \quad m >= 65.536 \end{cases}$$

Onde p_i e p_j são processadores do *site* UFF.

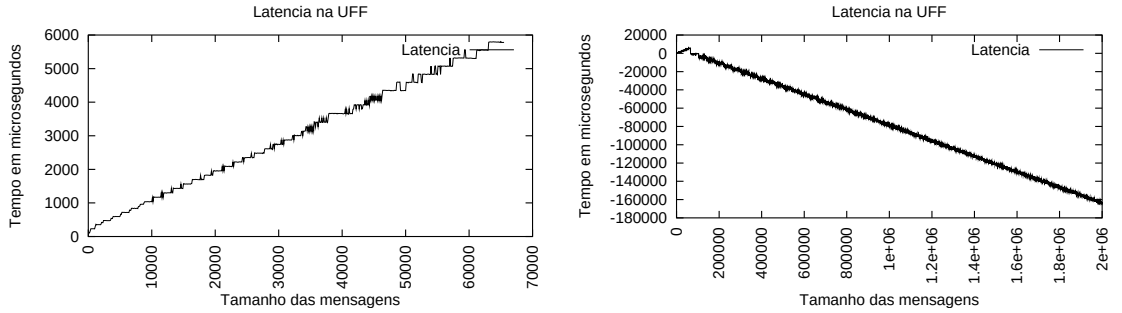


Figura 5.3: Latências medidas para tamanhos de mensagem variados, utilizando a versão II(b) do modelador compilada com a biblioteca de troca de mensagens MPI-LAM, para o *site* UFF.

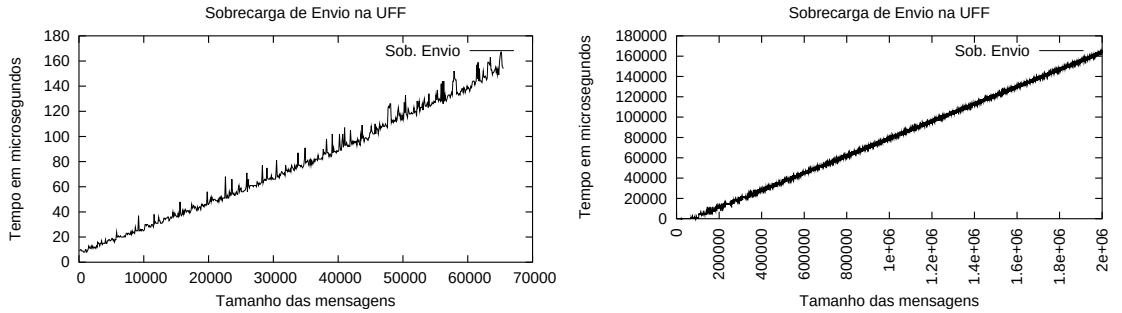


Figura 5.4: Sobrecargas de envio medidas para tamanhos de mensagem variados, utilizando a versão II(b) do modelador compilada com a biblioteca de troca de mensagens MPI-LAM, para o *site* UFF.

A sobrecarga de envio também tem comportamento diferente para mensagens de tamanhos inferior ou superior a 64 KBytes, conforme visto no gráfico da Figura 5.4. Neste caso, também são necessárias duas equações para o cálculo da sobrecarga de envio. A diferença das equações obtidas, com relação às equações do modelo analítico apresentado no Capítulo 4, são em decorrência da utilização do comando MPI_Send. Este comando só é finalizado quando o processo destino recebe toda a mensagem enviada. Com isso, seu valor varia também com relação ao tamanho da mensagem, sendo que para mensagens de tamanho inferior a 64 KBytes

o crescimento é quadrático e para mensagens de tamanho superior o crescimento é linear. As equações para determinação da sobrecarga de envio são:

$$Os(m, pi) = \begin{cases} 1,159 * 10^{-8} * m^2 + 1,705 * 10^{-3} * m + 9 & \text{se} \quad m < 65.536 \\ 1,508 * (m - 65536) + 97 & \text{se} \quad m >= 65.536 \end{cases}$$

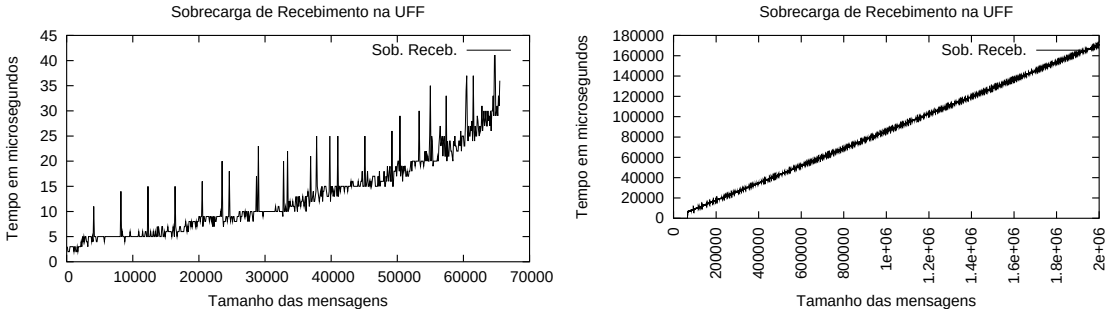


Figura 5.5: Sobrecargas de recebimento medidas para tamanhos de mensagem variados, utilizando a versão II(b) do modelador compilada com a biblioteca de troca de mensagens MPI-LAM, para o *site* UFF.

Analisando os gráficos da Figura 5.5, podemos verificar que o comportamento da sobrecarga de recebimento é o mesmo com relação ao modelo analítico apresentado no Capítulo 4. Assim, as equações para cálculo da sobrecarga de recebimento são as seguintes:

$$Or(m, pj) = \begin{cases} 5,217 * 10^{-9} * m^2 + 1,023 * 10^{-3} * m + 2 & \text{se} \quad m < 65.536 \\ 0,0849 * (m - 65536) + 6006 & \text{se} \quad m >= 65.536 \end{cases}$$

Equações de comunicação entre *sites* e entre processos de um mesmo *host*

Igualmente a versão do modelo analítico apresentada no Capítulo 4, as equações para cálculo dos parâmetros para os demais *sites*, são obtidas pela análise dos gráficos apresentados no Apendice B. Com excessão da comunicação entre processos executados em um mesmo *host*, observamos que o comportamento para os demais

sites é o mesmo em relação ao *site* da UFF. Neste caso, somente os valores das constantes são alterados em função de diferenças nas velocidades das redes. As Tabelas 5.6, 5.7 e 5.8 mostram as equações para os demais *sites*.

Site	Equação
mesmo host	$L(m,pi,pj) = 0,0022 * (m - 64) + 24$ se m < 65.536 0 se m $>= 65.536$
site Sinergia	$L(m,pi,pj) = 0,0110 * (m - 64) + 100$ se m < 65.536 $L(m,pi,pj) = 0,0110 * (m - 65.536) - 605$ se m $>= 65.536$
entre UFF e Sinergia	$L(m,pi,pj) = 1,0844 * (m - 64) + 100$ se m < 65.536 $L(m,pi,pj) = -1,369 * (m - 65.536) - 664$ se m $>= 65.536$

Tabela 5.6: Equações para o cálculo da latência para MPI-LAM.

Site	Equação
mesmo host	$Os(m,pj) = 4,6373 * 10^{-8} * m^2 + 1,364 * 10^{-3} * m + 9$ m < 65.536 $Os(m,pi) = 0,0850 * (m - 65.536) - 174$ m $>= 65.536$
site Sinergia	$Os(m,pj) = 5,8 * 10^{-10} * m^2 + 1,705 * 10^{-3} * m + 7$ m < 65.536 $Os(m,pi) = 0,0096 * (m - 65.536) - 123$ m $>= 65.536$
entre UFF e Sinergia	$Os(m,pj) = 1,1593 * 10^{-8} * m^2 + 1,705 * 10^{-3} * m + 9$ m < 65.536 $Os(m,pi) = 1,508 * (m - 65.536) - 97$ m $>= 65.536$

Tabela 5.7: Equações para o cálculo da sobrecarga de envio para MPI_Send do MPI-LAM.

Site	Equação
mesmo *site*	$Or(m,pj) = 1,4492 * 10^{-8} * m^2 + 6,479 * 10^{-4} * m + 2$ m < 65.536 $Or(m,pj) = 0,00255 * (m - 65536) + 300$ m $>= 65.536$
site Sinergia	$Or(m,pj) = 5,217 * 10^{-9} * m^2 + 1,006 * 10^{-3} * m + 2$ m < 65.536 $Or(m,pj) = 0,00961 * (m - 65536) + 1000$ m $>= 65.536$
entre UFF e Sinergia	$Or(m,pj) = 1,159 * 10^{-8} * m^2 + 6,479 * 10^{-4} * m + 2,$ m < 65.536 $Or(m,pj) = 1,508 * (m - 65536) + 73415$ m $>= 65.536$

Tabela 5.8: Equações para o cálculo da sobrecarga de recebimento para os demais *sites* do GridRio.

5.2.1 Verificação da versão analítica do modelador para o MPI-LAM

A verificação dos parâmetros modelados foi feita de maneira igual à verificação da versão analítica apresentada no Capítulo 4. Cada processo executa uma carga

de trabalho duas vezes maior do que a utilizada pelo modelador para determinar o fator de heterogeneidade e, o tamanho das mensagens enviadas é igual a 2.097.152 bytes. Os testes foram realizados considerando processos sendo executados no *site* UFF e também considerando processos nos *sites* UFF e Sinergia. Os valores dos parâmetros foram obtidos utilizando as equações apresentadas como pode ser visto na Tabela 5.9.

Funçao	UFF	UFF-Sinergia
Os(T0,T1)	0,0847 * (2.097.152 - 65.536) + 138	1,5080 * (2.097.152 - 65.536) + 97
L(T0,T1)	-0,0850 * (2.097.152 - 65.536) - 410	-1,369 * (2.097.152 - 65.536) - 664
Or(T1,T0)	0,0849 * (2.097.152 - 65.536) + 6006	1.50795 * (2.097.152 - 65.536) + 73.415
Os(T1,T2)	0,0847 * (2.097.152 - 65.536) + 138	1,5080 * (2.097.152 - 65.536) + 97
L(T1,T2)	-0,0850 * (2.097.152 - 65.536) - 410	-1,369 * (2.097.152 - 65.536) - 664
Or(T2,T1)	0,0849 * (2.097.152 - 65.536) + 6006	1.50795 * (2.097.152 - 65.536) + 73.415
Os(T2,T0)	0,0847 * (2.097.152 - 65.536) + 138	* (2.097.152 -) +
L(T2,T0)	-0,850 * (2.097.152 - 65.536) - 410	-0,850 * (2.097.152 - 65.536) - 410
Or(T0,T2)	0,0849 * (2.097.152 - 65.536) + 6006	0,0849 * (2.097.152 - 65.536) + 6006
Proc T0	5.090.000 * (10.128.626/5.110.000)	5.090.000 * (10.128.626/5.110.000)
Proc T1	5.090.000 * (10.128.626/5.110.000)	5.090.000 * (10.128.626/5.110.000)
Proc T2	5.090.000 * (10.128.626/5.110.000)	5.090.000 * (10.128.626/5.110.000)

Tabela 5.9: Cálculo dos valores dos parâmetros do modelo HLogP pela versão analítica do modelador para MPI-LAM.

A partir dos valores calculados, podemos estimar o tempo de execução da aplicação, como pode ser visto na Tabela 5.10, para o caso de execução da aplicação em *hosts* da UFF e em *hosts* da UFF e Sinergia. A comparação entre o tempo previsto e o tempo de execução real da aplicação, apresentados na Tabela 5.11, mostra que podemos chegar, a partir dos valores modelados, a um valor bastante próximo ao tempo de execução.

5.3 Resumo do Capítulo

Neste capítulo foram apresentadas diferenças com relação ao comportamento da comunicação em aplicações implementadas utilizando a versão MPI-LAM e MPICH-G2. A principal diferença ocorre em situações onde os processos referentes às tarefas de envio e de recebimento se encontram em um mesmo host. Em função dessas diferenças e com o objetivo de modelar o comando MPI_Send, uma novo

Etapa	Tempo (μs) UFF-UFF	Tempo (μs) UFF-Sinergia
Start time T0	0	0
Proc(T0)	10.088.984	10.088.984
Os(T0,T1)	172.133	3.063.740
L(T0,T1)	-173.024	-2.781.950
Or(T1,T0)	178.497	178.497
Proc(T1)	10.088.984	10.088.984
Os(T1,T2)	172.133	3.063.740
L(T1,T2)	-173.024	-2.781.950
Or(T2,T1)	178.497	3.136.989
Proc(T2)	10.088.984	10.088.984
Os(T2,T0)	172.133	172.133
L(T2,T0)	-173.024	-173.024
Or(T0,T2)	178.497	3.136.989
Finish time T0	30.799.770	37.282.116

Tabela 5.10: Evolução do tempo de execução estimado, com comunicação entre processos da UFF e entre processos da UFF e Sinergia.

	Tempo Execução (μs)	Tempo Estimado (μs)	Erro (%)
UFF-UFF	30.093.816	30.799.770	**2,29**
UFF-PUC	36.424.200	37.282.116	**2,30**

Tabela 5.11: Comparação entre tempo de execução da aplicação e o tempo a partir dos parâmetros modelados, utilizando a versão analítica do modelador para MPI-LAM.

modelo analítico foi proposto. Foram apresentados também, os resultados dos testes de validação do novo modelo. No próximo capítulo será investigada a qualidade do modelo proposto como um modelo de escalonamento.

Capítulo 6

Validação do Modelo

Neste capítulo são apresentados os testes e os resultados da validação do modelo de escalonamento **HLogP** proposto no Capítulo 3. Os testes realizados tiveram como o objetivo a comparação do tempo de execução de uma aplicação paralela escalonada com base no modelo proposto, com o tempo de execução da mesma aplicação escalonada com base em outros modelos de escalonamento: *round-robin* feito pelo MPI (estratégia mais utilizada por usuários/programadores que executam aplicações nesses ambientes) e um escalonamento baseado no modelo latência. O foco deste capítulo é no uso do MPI-LAM, devido ao mal comportamento do MPICH-G2 quando processos dependentes são executados no mesmo processador. Inicialmente foi verificada a influência das sobrecargas de envio e de recebimento no escalonamento de tarefas num mesmo host.

6.1 Influência das sobrecargas de envio e recebimento

O envio e o recebimento de mensagens, utilizando comandos de bibliotecas de comunicação como o MPI, têm um custo que pode influenciar sobremaneira o processo de escalonamento de uma aplicação. Em situações onde, por exemplo, uma tarefa envia várias mensagens e o escalonador determina que uma outra tarefa seja

executada no mesmo processador onde se encontra a tarefa de envio, o tempo de início da segunda tarefa é atrasado. Um outro caso obesrvado é quando a tarefa recebe várias mensagens antes de iniciar seu processamento efetivo. Nesta situação, seu tempo de início também é atrasado, mas nem sempre por causa da última mensagem a chegar. Dessa forma, torna-se importante a verificação do comportamento da comunicação nesses casos.

6.1.1 Sobrecarga de envio

A aplicação (grafo *fork*) para teste da influência da sobrecarga de envio, ilustrada na Figura 6.1, é formada por uma tarefa inicial T0 que envia mensagens de tamanho 4Mb Kbytes para as tarefas filhas (T1,T2,...,T9), sendo que a tarefa T9 é executada na mesma unidade de processamento da tarefa pai e as demais em outras. A última mensagem enviada pela tarefa T0 é a referente à tarefa T9, que roda no mesmo processador. Nos testes realizados, usando o comando MPI_Send, verificou-se que para uma sobrecarga de envio média, para os hosts P1 a P8 igual a $\mathbf{357.241}\boldsymbol{\mu s}$, o início da tarefa T9 foi atrasado em $\mathbf{2.866.306}\boldsymbol{\mu s}$, em função das sobrecargas de envio para as tarefas sucessoras. Quando não se considera o envio das mensagens para as tarefas T1 a T8, o tempo de início da tarefa T9 foi atrasado em apenas $\mathbf{8.136}\boldsymbol{\mu s}$, que é o atraso referente à mensagem de T0 para T9.

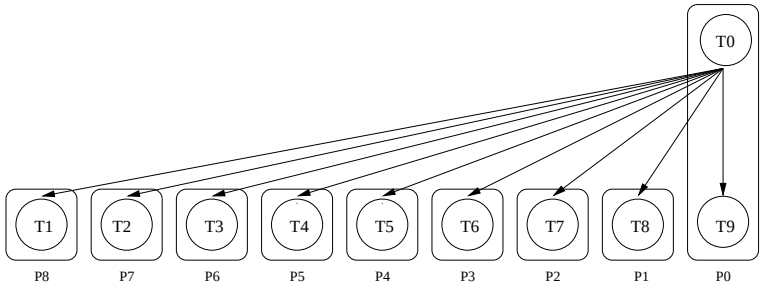


Figura 6.1: Aplicação para testar a sobrecarga de envio.

6.1.2 Sobrecarga de recebimento

Para a verificação da influência da sobrecarga de recebimento, a aplicação (grafo *join*) utilizada foi a ilustrada na Figura 6.2. Esta aplicação é formada por um conjunto de tarefas (T0,T1,...,T8) que enviam, cada uma, uma mensagem para a tarefa T9. A tarefa T9, é executada na mesma unidade de processamento que a tarefa T8. Nos testes realizados, verifiou-se que o início da tarefa T9 é atrasado em função das sobrecargas de recebimento das mensagens enviadas pelas tarefas predecessoras, se comparado com o tempo de início caso as mensagens não sejam enviadas. Para uma sobrecarga de recebimento média igual a $\mathbf{361.216}\boldsymbol{\mu s}$, a tarefa T9 foi atrasada em $\mathbf{2.906.737}\boldsymbol{\mu s}$ quando aguarda as mensagens enviadas por (T0,T1,...T8). Quando não ocorre o envio das mensagens, a tarefa T9 foi atrasada em apenas $\mathbf{12.126}\boldsymbol{\mu s}$, referente ao recebimento da mensagem de T8.

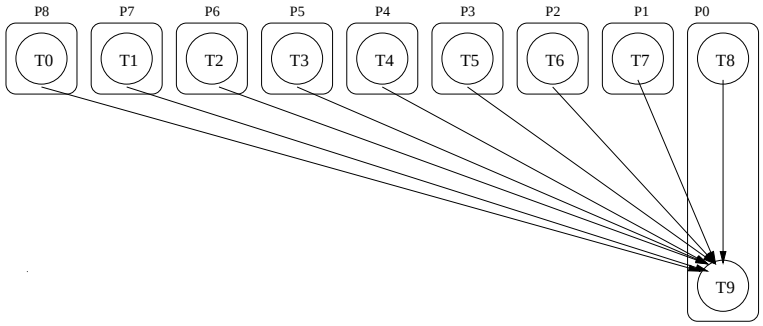


Figura 6.2: Aplicação para testar a sobrecarga de recebimento.

6.2 Escalonamento usando HLogP

Ainda não existe na literatura nenhum algoritmo para o problema de escalonamento de tarefas em ambientes heterogêneos sobre o modelo HLogP. Mesmo para o modelo LogP, em ambientes homogêneos, só são conhecidos três: MSA [5, 7] e HAL [6, 8, 34], algoritmos baseados na aglomeração e que utilizam replicação de tarefas, e uma versão do algoritmo ETF modificado por Trystram [28]. Atualmente encontram-se em desenvolvimento e avaliação técnicas para tratar as sobrecargas de

envio e de recebimento em algoritmos de escalonamento [12]. Para gerar o escalonamento da aplicação baseada no modelo HLogP, foi criada uma versão modificada do algoritmo HEFT, cujo objetivo é investigar a usabilidade do modelo e seus benefícios. Esse novo algoritmo é uma extensão do trabalho preliminar de Moura e Menezes [32]. Baseado na técnica de *list scheduling*, o algoritmo escolhe a tarefa de maior prioridade e a aloca no processador onde terminará mais cedo. No escalonamento de uma tarefa, as sobrecargas de recebimento são alocadas imediatamente antes da tarefa, em ordem do tempo de chegada das mensagens correspondentes. Depois do escalonamento da tarefa, uma reserva de tempo é alocada para todos as sobrecargas de envio desta tarefa. Inicialmente, o tamanho desta reserva é determinado pelo número de envios, o tamanho da mensagem e a maior sobrecarga de envio desta mensagem no processador. Durante o escalonamento, os tamanhos das janelas de reserva são ajustados de acordo com o processador destino da mensagem. Devido ao tempo e complexidade do problema de escalonamento, pouco esforço foi feito para desenvolver um algoritmo de alta qualidade (por exemplo, nenhuma das técnicas propostas em [12] foi utilizada).

O tempo de execução ou *makespan* do escalonamento gerado pela heurística é comparado com os *makespans* produzidos pela heurística original baseada no modelo latência e pelo escalonamento *round-robin* feito pelo comando *mpirun*. Para fazer uma comparação justa entre os modelos de escalonamento, as seguintes suposições foram feitas com respeito ao modelo latência:

- Sobrecarga de envio e de recebimento são consideradas zero.
- A latência no mesmo processador também é considerada zero.
- Os valores da latência para comunicação entre processadores diferentes não são os mesmos considerados pelo modelo HLogP, mas são calculados baseados na metade do RTT. Dessa maneira, o custo de enviar e receber mensagens **é considerado**, mas como custo de comunicação e não de processamento. Isto é, a única diferença considerada entre os dois modelos é que as sobrecargas bloqueiam a *CPU* no modelo HLogP.

Para a validação do modelo, foi utilizado o grafo da classe diamante com 25 tarefas, Figura 6.3. O gráfico foi escalonado pelo algoritmo HEFT modificado tanto para o modelo HLogP, como para o modelo latência. Os tempos estimados de execução a partir dos escalonamentos obtidos foram, respectivamente, $\mathbf{50.153.376}\boldsymbol{\mu s}$ (algoritmo HEFT modificado baseado nas medidas da versão II(b) do Modelador), $\mathbf{51.474.996}\boldsymbol{\mu s}$ (basedo na versão analítica) e $\mathbf{47.238.064}\boldsymbol{\mu s}$ (baseado no modelo de latência). Os escalonamentos gerados pela heurística HEFT modificado, usando os valores medidos pelas duas versões do modelador, foram iguais. No caso do modelo de latência, o escalonamento gerado foi diferente. Foi implementada uma aplicação paralela em MPI que foi compilada usando a versão MPI-LAM para ser executada no Grid. Após a execução da aplicação segundo os escalonamentos obtidos, o tempo de execução foram, respectivamente, $\mathbf{50.193.367}\boldsymbol{\mu s}$ e $\mathbf{57.476.327}\boldsymbol{\mu s}$. Podemos verificar que, mesmo sendo menor o tempo previsto no escalonamento obtido a partir do modelo latência, o tempo de execução da aplicação foi superior ao tempo de execução da aplicação considerando o modelo HLogP. Isso mostra que o escalonamento da aplicação segundo o modelo HLogP foi mais eficiente. Podemos verificar também que o modelo HLogP proporcionou uma estimativa mais precisa do tempo de execução da aplicação, onde neste caso o erro foi de $\mathbf{0,08\%}$ (usando a versão II(b)) e $\mathbf{2,55\%}$ (usando a versão analítica) para o modelo HLogP e $\mathbf{21,70\%}$ para o modelo latência. O tempo de execução da aplicação utilizando o escalonamento *round-robin* foi de $\mathbf{82.422.787}\boldsymbol{\mu s}$. Em comparação com o tempo obtido baseado no modelo HLogP, o tempo de execução foi $\mathbf{64,2\%}$ mais lento.

6.3 Resumo do Capítulo

Neste capítulo foi apresentada a validação do modelo HLogP. Inicialmente foi mostrada a importância (influência) das sobrecargas de envio e recebimento no escalonamento de tarefas em um mesmo processador. Foi apresentada também a comparação do escalonamento obtido com base nos modelos HLogP e latência.

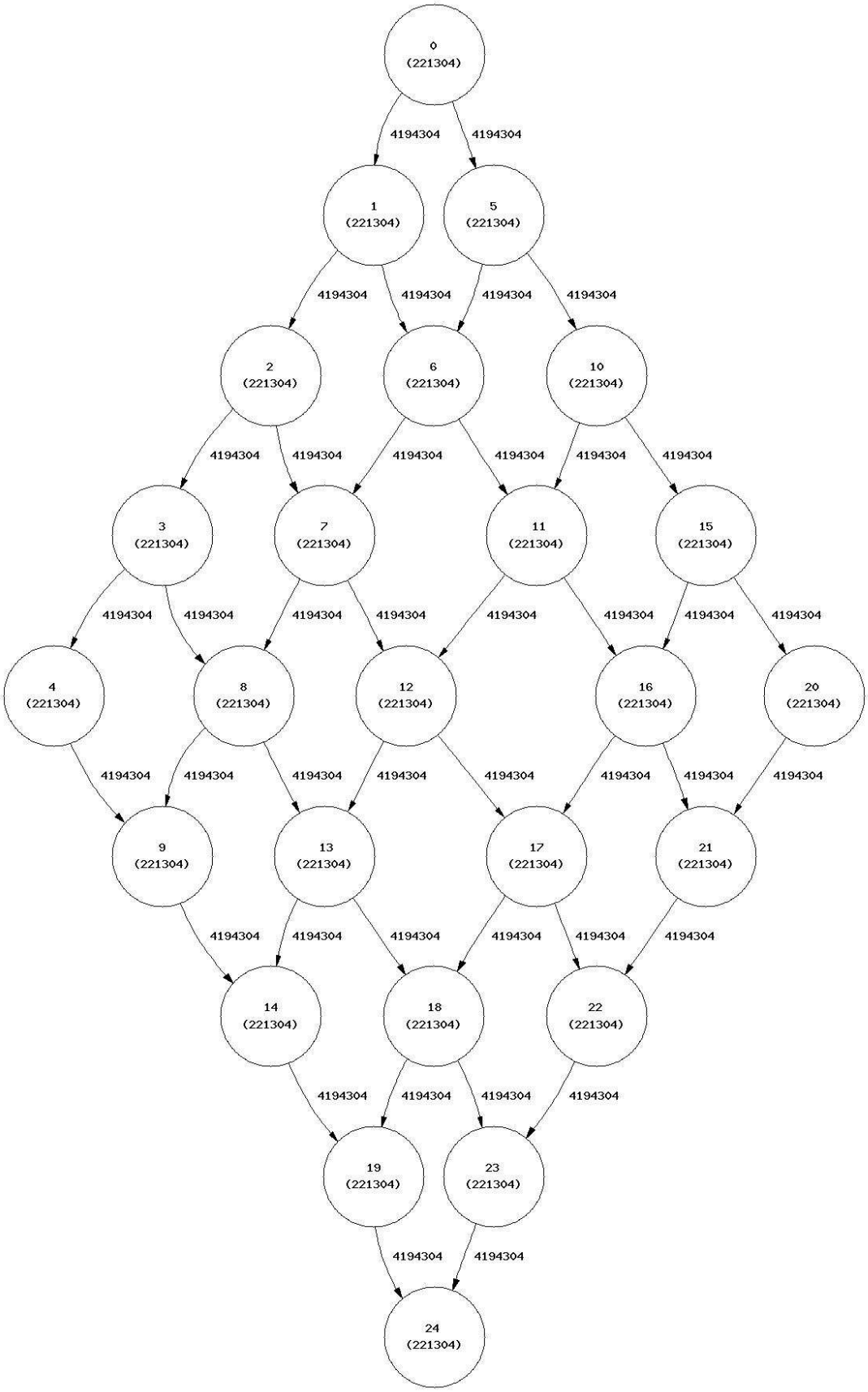


Figura 6.3: Representação GAD que representa a aplicação utilizada na validação do modelo.

Capítulo 7

Conclusões e Trabalhos Futuros

Este trabalho teve como objetivo o estudo dos modelos de escalonamento para ambientes paralelos existentes, e a definição e avaliação de um novo modelo adequado para o escalonamento de aplicações MPI em Grids computacionais. Durante o trabalho foi verificado que os modelos existentes não possuem caracaterísticas que os tornem aplicáveis aos ambientes Grid. A maioria dos modelos estudados são para ambientes homogêneos ou, mesmo quando são para ambientes heterogêneos (que é o caso do Grid), não tratam de maneira adequada as características do ambiente.

O modelo HLogP para escalonamento de aplicações MPI em ambientes Grid, proposto neste trabalho, foi baseado nos modelos LogP [2, 14, 27, 41], que definem basicamente 4 parâmetros: latência, sobrecarga de envio, sobrecarga de recebimento e gap, para o custo de envio de uma mensagem de um computador a outro. No modelo realístico de comunicação em *cluster* [41], os mesmos parâmetros de comunicação são definidos como funções do tamanho do conjunto de dados enviados. O novo modelo proposto também define a latência como função do tamanho da mensagem e dos processadores envolvidos no envio e recebimento dos dados. No entanto, as sobrecargas de envio e recebimento são definidas como funções do tamanho da mensagem e do processador envolvido no processo de envio ou recebimento dos dados, diferentemente dos outros modelos que tratam esses parâmetros.

Para verificação e validação do modelo proposto foi desenvolvida uma fer-

ramenta de modelagem dos parâmetros do modelo para ser utilizada em ambientes Grid. Um dos objetivos foi criar uma ferramenta que fosse independente do sistema instalado em cada *host* dos *sites* do Grid. No desenvolvimento da ferramenta, três versões foram apresentadas, sendo que cada versão tinha como objetivo diminuir a intrusão gerada pela versão anterior no sistema. A primeira versão foi desenvolvida com o objetivo de verificar a utilização da biblioteca MPI, utilizada na implementação de aplicações paralelas, para a realização da modelagem da comunicação entre processos, segundo o modelo HLogP proposto, e também de verificar a modelagem das características de processamento e carga dos *hosts*. Para a modelagem da comunicação foi utilizado o método rápido de medida dos parâmetros LogP proposto por Kielman [29] e para a modelagem da computação, um trecho símples de código. Na segunda versão, foi proposta para a modelagem da comunicação a utilização do método baseado no tradicional *ping-pong*, com o objetivo de diminuir o tempo gasto na modelagem. Para a computação, outros trechos de código, envolvendo computações tradicionais, como multiplicação de matrizes e eliminação gaussiana, foram utilizados. Foi mostrado que esta versão não perde na precisão dos valores obtidos. Uma versão mais escalável do modelador anterior foi proposta para diminuir o número de medidas de comunicação necessárias, escolhendo apenas alguns *hosts* pertencentes a cada *site* que participaram da modelagem. Os valores dos parâmetros obtidos para o conjunto reduzido são assumidos para os demais *hosts*. A terceira versão do modelador, a versão analítica, propõe a utilização de um conjunto de equações para a modelagem dos parâmetros do modelo. A utilização das equações tem como objetivo diminuir o número de vezes em que um modelador não analítico precisa ser executado, e então reduzir o grau de intrusão no sistema.

Foi verificado nos resultados obtidos pela modelagem, utilizando a versão I do modelador, que é possível coletar os valores dos parâmetros do modelo HLogP, utilizando os comandos da biblioteca MPI. Os valores dos parâmetros obtidos foram utilizados para fazer a estimativa do tempo de execução de uma pequena aplicação paralela. O tempo estimado obtido foi comparado com o tempo real de execução da aplicação em um Grid, sendo que o erro foi de apenas $\mathbf{0,97\%}$ para comunicação entre *hosts* de um mesmo *site* e de $\mathbf{4,69\%}$ para comunicação envolvendo *hosts* de

sites diferentes. Os resultados obtidos com as versões II(a) e II(b) do modelador mostraram que as simplificações introduzidas nessas novas versões não prejudicaram o poder de previsão feito com base nos valores dos obtidos para os parâmetros. No caso da versão II(a) os erros da estimativa do tempo de execução foram de $\mathbf{0,21\%}$ e $\mathbf{2,08\%}$ para *hosts* de um mesmo *site* e de *sites* diferentes respectivamente; e pela versão II(b), $\mathbf{0,21\%}$ e $\mathbf{2,18\%}$ de erro. Com relação à intrusão gerada pela execução das aplicações de modelagem das três versões não analíticas, foi verificado que as versões simplificadas (II(a) e II(b)) gastam um tempo consideravalmente menor de execução, sobrecarregando menos os recursos do Grid. Em relação à versão I, os tempos de execução dos modeladores II(a) e II(b) foram $\mathbf{43,75\%}$ e $\mathbf{1,62\%}$s mais rápidos respectivamente. A versão analítica proposta também obteve uma boa precisão na estimativa do tempo de execução da aplicação de teste, $\mathbf{0,45\%}$ e $\mathbf{0,56\%}$ para *hosts* de um mesmo *site* e de *sites* diferentes respectivamente.

Durante os testes das versões I, II(a) e II(b) da ferramenta de modelagem, foram utilizadas as versões MPICH-G2 e MPI-LAM da biblioteca de comunicação MPI. Os resultados obtidos, referentes aos parâmetros de comunicação do modelo, foram diferentes. Dessa forma, foi proposta também uma versão analítica para a modelagem dos parâmetros de comunicação para o comando MPI_Send da versão MPI-LAM. Como os mesmos testes de verificação foram realizados, o erro na estimativa do tempo de execução da aplicação ficou na mesma ordem dos outros modeladores ($\mathbf{2,29\%}$ e $\mathbf{2,30\%}$). Outra diferença importante entre as duas versões é com relação à execução de dois processos dependentes da mesma aplicação em um mesmo host. Na versão MPICH-G2, quando um processo que aguarda o envio de uma mensagem é executado no mesmo *host* em que o processo que envia é executado, o tempo de processamento do processo que envia é afetado pelo processo que recebe a mensagem. Se esse fato não for tratado pelos escalonadores, os escalonamentos produzidos podem levar a resultados ruins na execução de uma aplicação paralela. Já na versão MPI-LAM, o mesmo não foi observado, um processo não interfere no tempo de execução do outro quando existe dependência e são executados em um mesmo *host*.

Por fim, foi feita uma validação do modelo proposto onde foram comparados

o escalonamento produzido com base nos modelos HLogP, latência e no escalonamento *round-robin* feito pelo MPI. Nos testes de verificação da precisão do modelo, foi utilizada uma aplicação teórica da classe *fork-join* com 14 tarefas. A aplicação foi escalonada para um ambiente Grid segundo os modelos HLogP e latência, sendo obtida a distribuição dos processos e o tempo estimado de execução da aplicação. Os tempos estimados obtidos para cada escalonamento foram, respectivamente, **24.686 milesegundos** e **21.074 milesegundos**. Comparando com o tempo real de execução da aplicação (**24.726 milesegundos**), foi constatado que o escalonamento baseado no modelo HLogP produziu uma estimativa mais precisa. Isso mostra que o modelo HLogP abstrai melhor o ambiente real de um Grid computacional.

Podemos concluir que o modelo HLogP e os mecanismos de modelagem propostos neste trabalho permitem que ferramentas de escalonamento de aplicações MPI mais eficientes sejam construídas para ambientes heterogêneos e dinâmicos, como é o caso do Grid. E, consequentemente, que a execução de aplicações MPI em ambientes tenha resultados melhores.

Embora bons resultados tenham sido obtidos, alguns pontos deste trabalho devem ser investigados em trabalhos futuros. A seguir apresentamos algumas sujestões de trabalhos futuros na área.

- Investigar versões do código para modelagem da capacidade de processamento dos hosts que sejam mais abrangentes.
- Estudar a aplicação do modelador analítico para o caso de um Grid maior (mais equações).
- Fazer um refinamento e estudo do comportamento das equações do modelo analítico com a variação na utilização dos recursos.
- Para o modelador versão II(b), definir uma forma mais precisa de escolha de mais de um *host* de um mesmo *site* quando os *hosts* forem heterogêneos.
- Investigar como escalonadores podem tratar as outras funções de comunicação da biblioteca MPI.

- Estudar o efeito da concorrência no modelo: avanços na tecnologia (*Direct Memory Access*, *hyperthreading*) de processadores modernos permitem que processamento e sobrecarga sejam executados simultaneamente.
- Desenvolver algoritmos de escalonamento eficientes para o modelo HLogP.
- Validação com problemas maiores e mais representativos.

Apêndice A

Gráficos de Comunicação da versão compilada para MPICH-G2

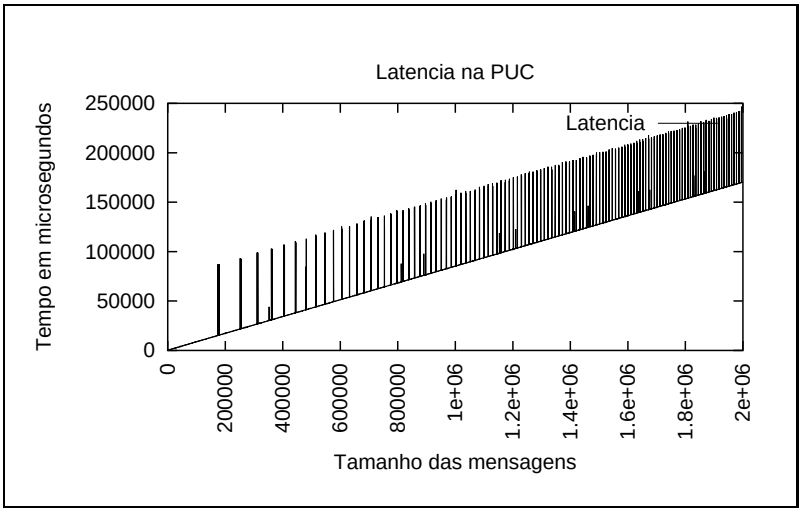


Figura A.1: Latencias para tamanho de mensagens variados obtidas da comunicação entre hosts do site PUC.

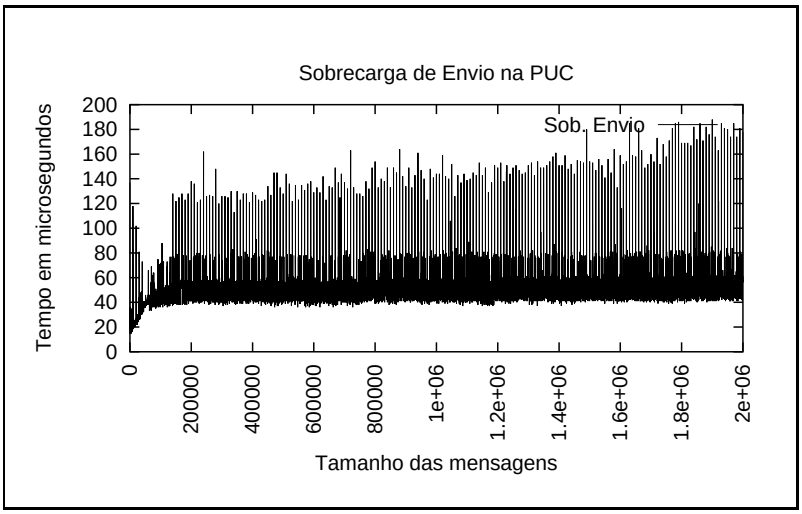


Figura A.2: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do site PUC.

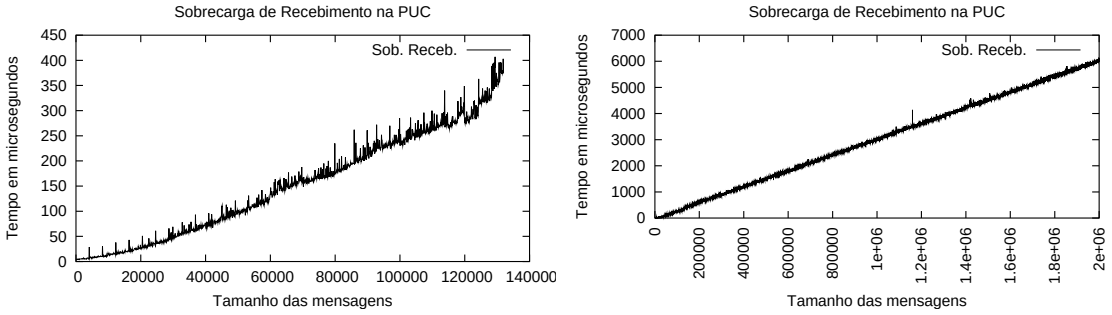


Figura A.3: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação entre hosts do site PUC.

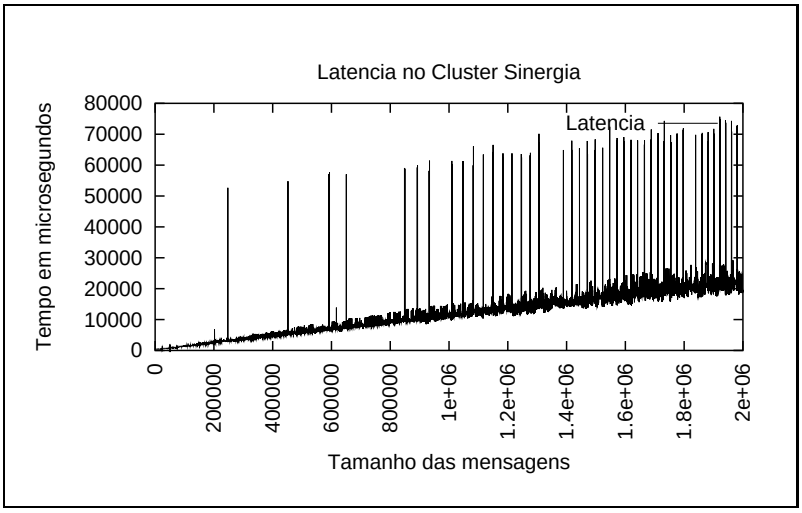


Figura A.4: Latencias para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia.

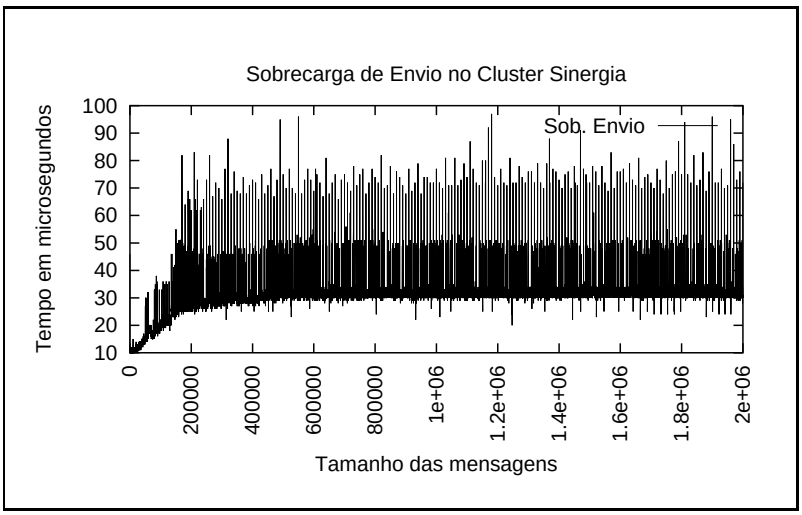


Figura A.5: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia.

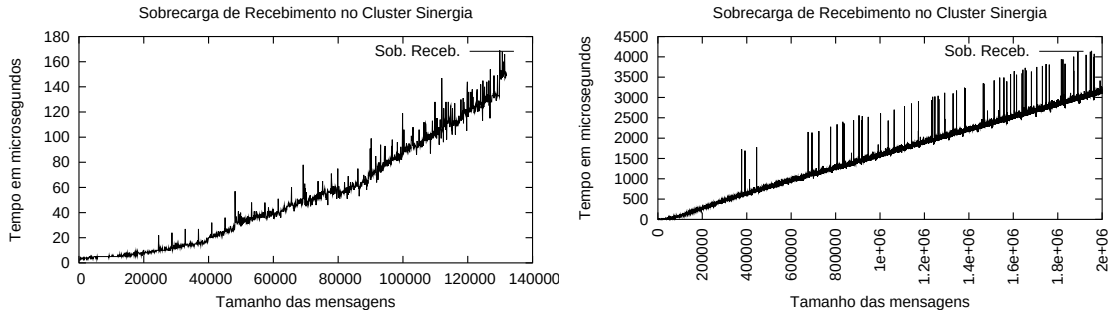


Figura A.6: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia.

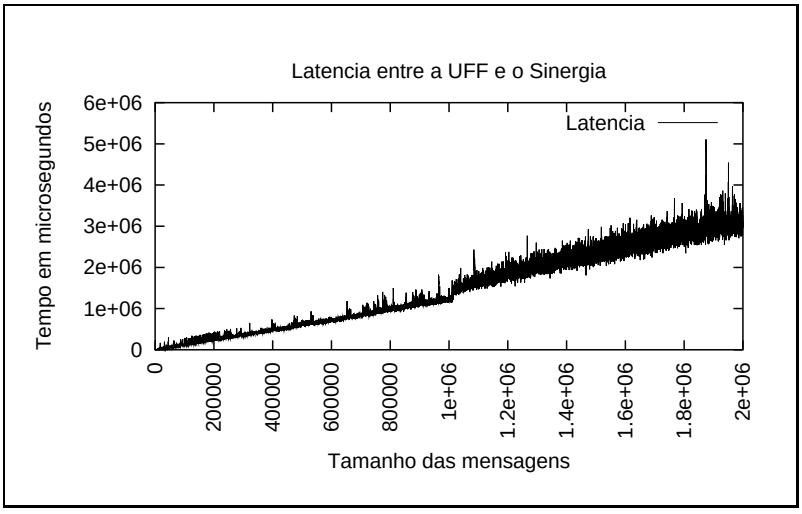


Figura A.7: Latencias para tamanho de mensagens variados obtidas da comunicação entre hosts do site da UFF e sinergia.

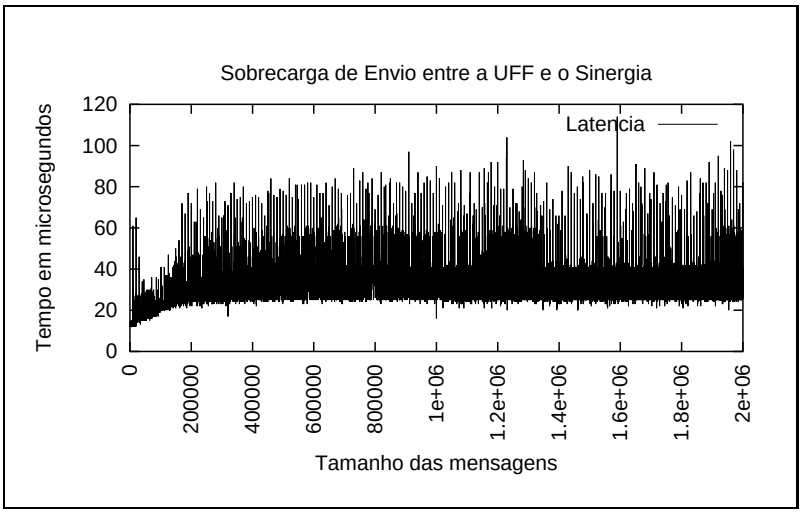


Figura A.8: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do site da UFF e sinergia.

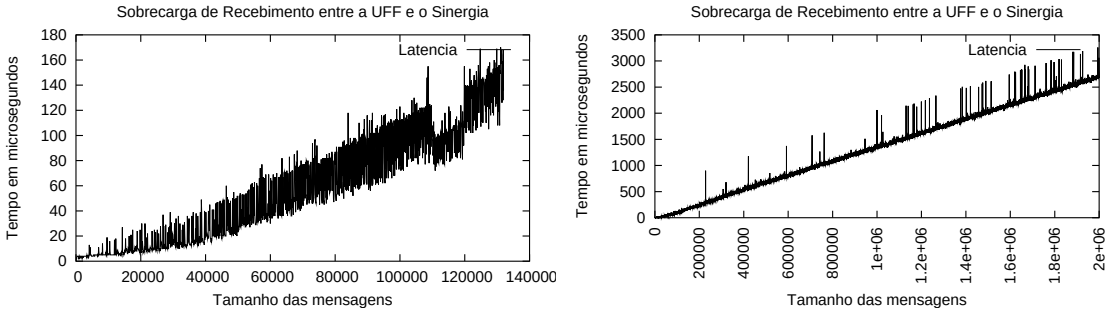


Figura A.9: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação entre hosts do site da UFF e sinergia.

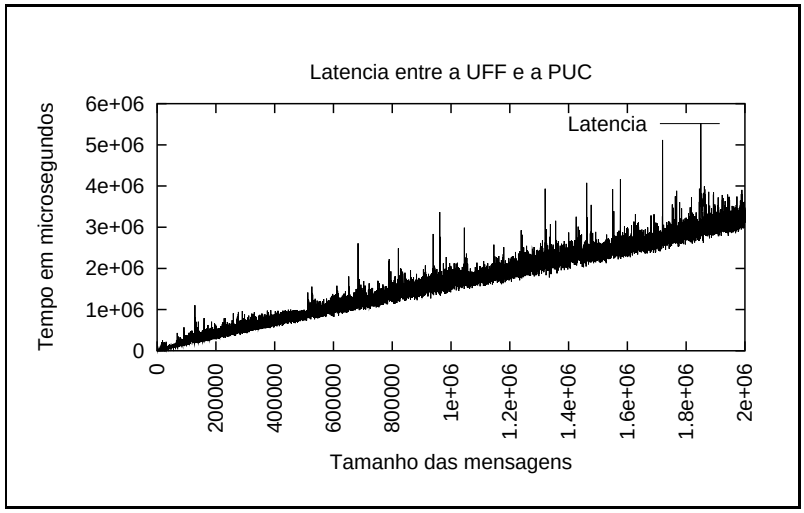


Figura A.10: Latencias para tamanho de mensagens variados obtidas da comunicação entre hosts da UFF e da PUC.

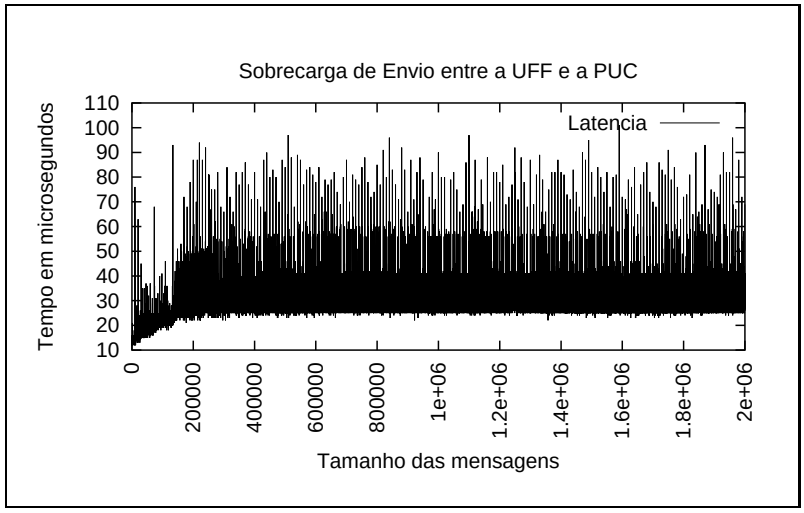


Figura A.11: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do site da UFF e da PUC.

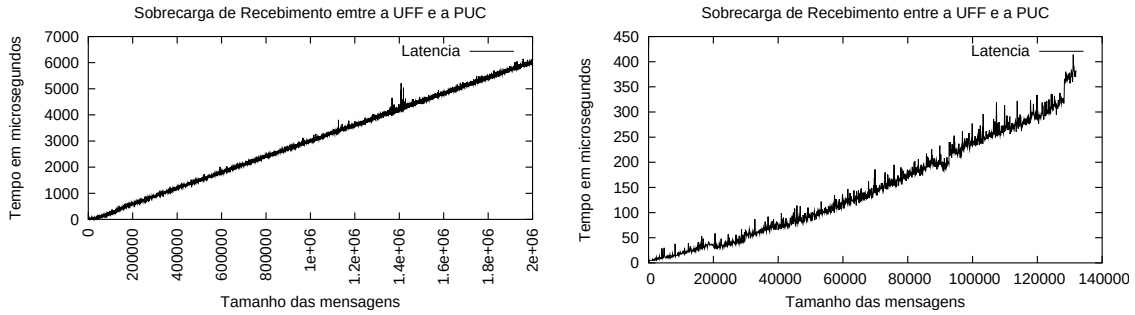


Figura A.12: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação entre hosts do site da UFF e da PUC.

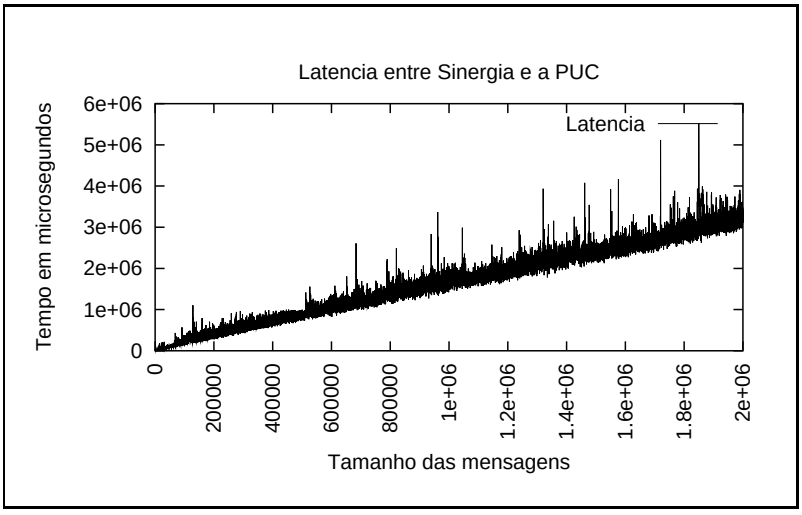


Figura A.13: Latencias para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia e da PUC.

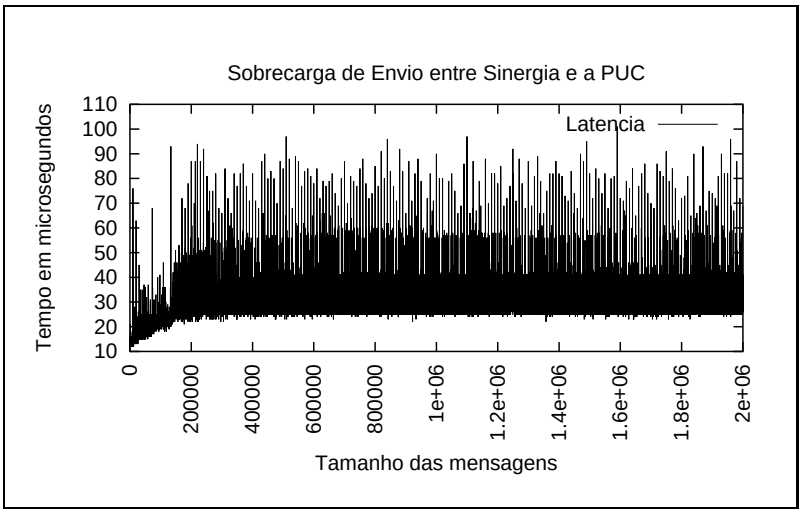


Figura A.14: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia e da PUC.

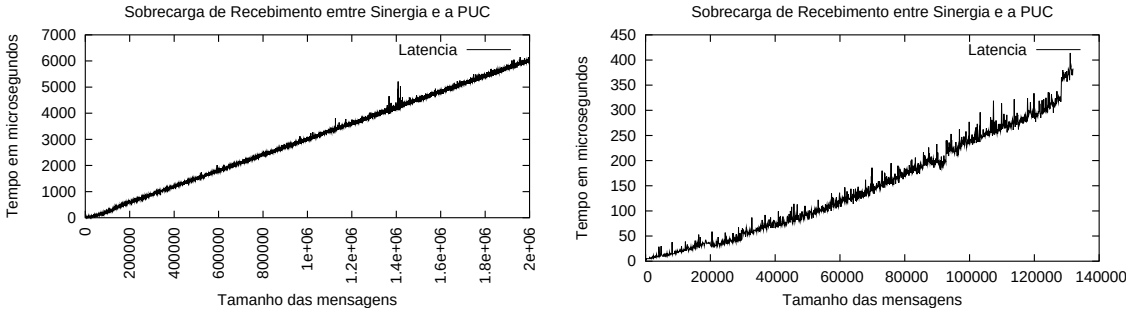


Figura A.15: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia e da PUC.

Apêndice B

Gráficos de Comunicação da versão compilada para MPI-LAM

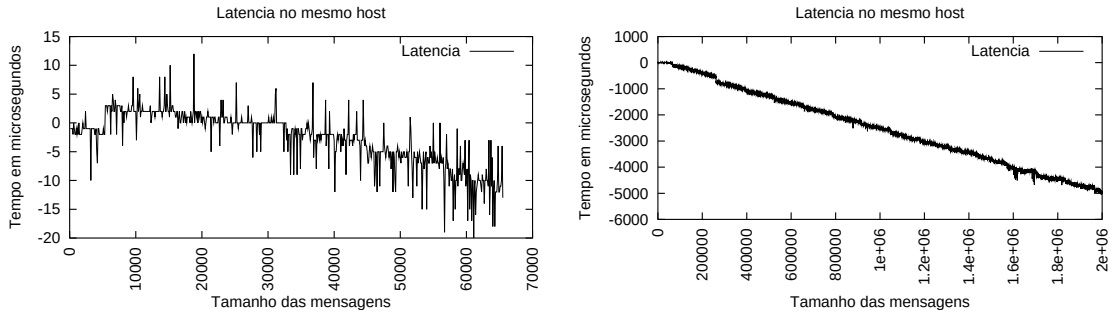


Figura B.1: Latências para tamanho de mensagens variados obtidas da comunicação entre um mesmo host.

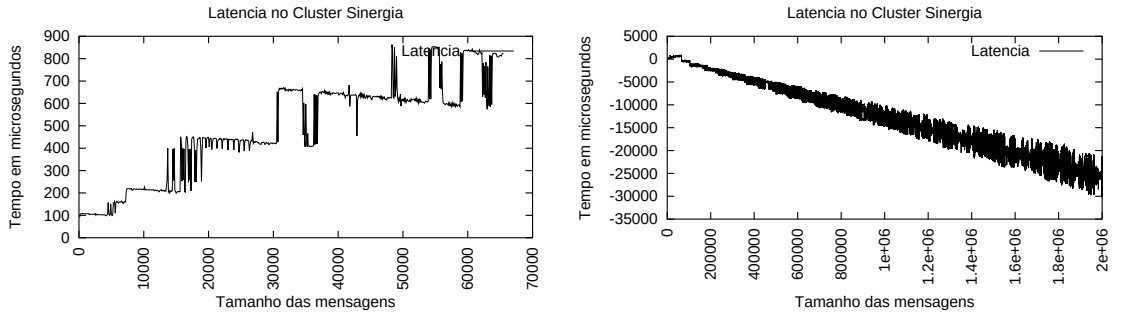


Figura B.2: Latências para tamanho de mensagens variados obtidas da comunicação entre hosts do site Sinergia.

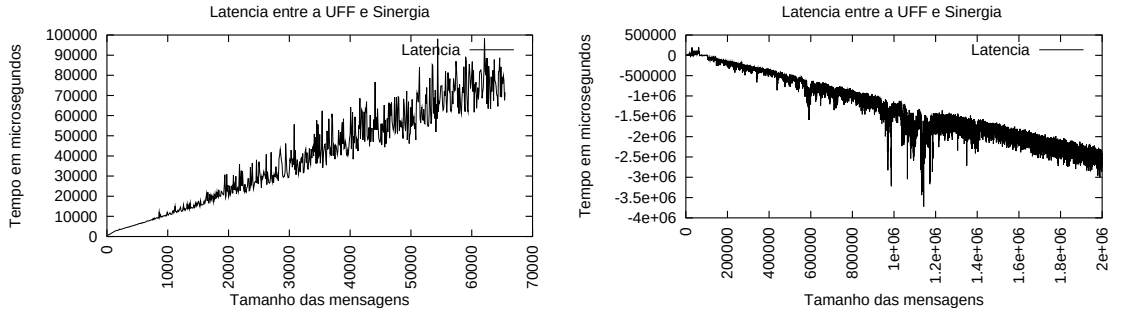


Figura B.3: Latências para tamanho de mensagens variados obtidas da comunicação entre hosts do site UFF e sinergia.

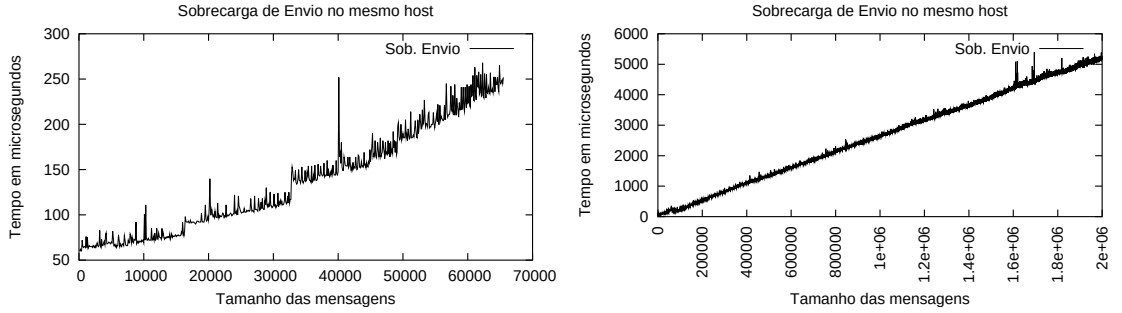


Figura B.4: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do mesmo site.

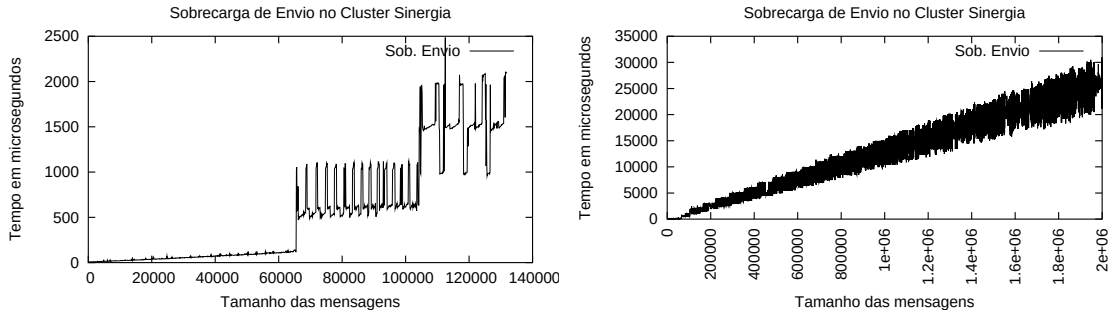


Figura B.5: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia.

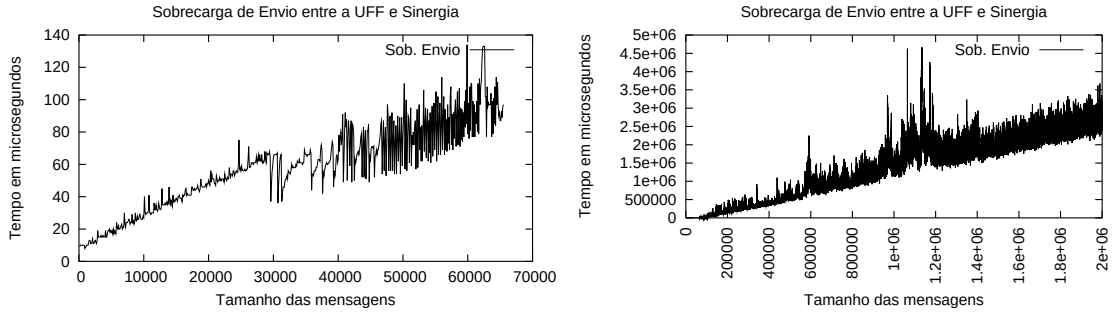


Figura B.6: Sobrecargas de envio para tamanho de mensagens variados obtidas da comunicação entre hosts do site UFF e sinergia.

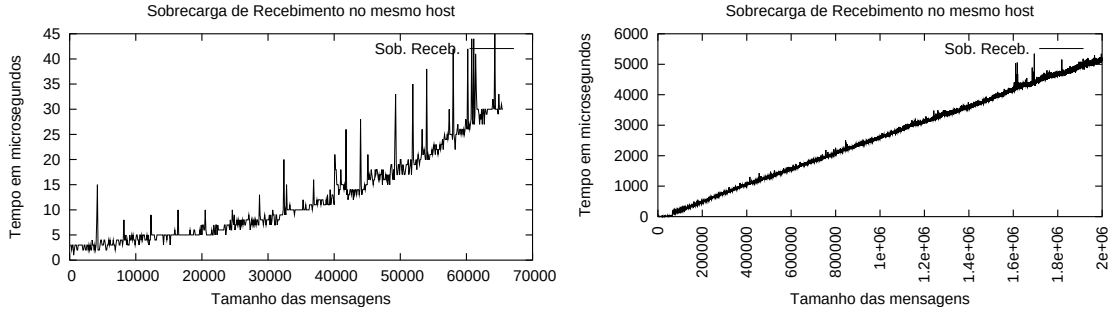


Figura B.7: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação em um mesmo host.

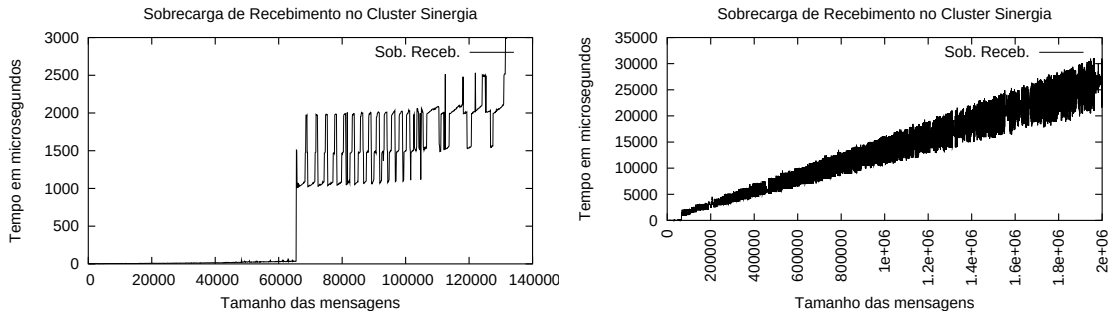


Figura B.8: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação entre hosts do site sinergia.

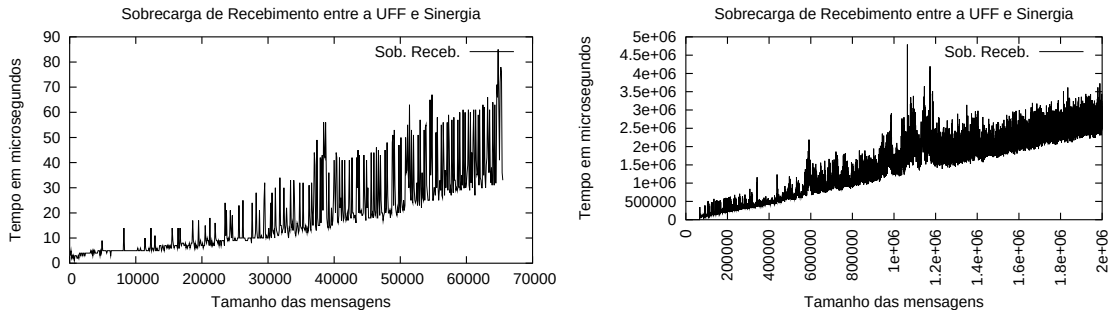


Figura B.9: Sobrecargas de recebimento para tamanho de mensagens variados obtidas da comunicação entre hosts do site UFF e sinergia.

Bibliografia

[1] K. Al-Tawil and C. A. Moritz. LogGP quantified: The case for MPI. *In Proc. 7th IEEE International Symposium on High Performance Distributed Computing (HPDC-7)*, August 1998.

[2] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating long messages into the LogP model - one step closer towards a realistic model for parallel computation. *In Proc. Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 95–105, 1995.

[3] M. Livny Basney and P. Mazzanti. Harnessing the capacity of computarional grid for high energy physics. In *Proc. of the Conference on Computing in High Energy and Nuclear Physics*, 2000.

[4] F. Berman and R. Wolski. The AppLeS project: A status report. In *Proceedings of the 8th NEC Research Symposium, Germany*, May 1997.

[5] C. Boeres. *Versatile Communication Cost Modelling for Multicomputer Task Scheduling Heuristics*. PhD thesis, Department of Computer Science, University of Edinburgh, 1997.

[6] C. Boeres, A. P. Nascimento, and V. E. F. Rebello. Cluster-based task scheduling for LogP model. *International Journal of Foundations of Computer Science*, 10(4):405–424, 1999.

[7] C. Boeres and V. E. F. Rebello. A versatile cost modelling approach for multicomputer task scheduling. *Parallel Computing*, 25(1):63–86, 1999.

[8] C. Boeres and V.E.F. Rebello. LogP cluster-based scheduling: Theory and practice. In *The Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002)*, Vitória, Brazil, October 2002. IEEE Computer Society Press.

[9] C. Boeres and V.E.F. Rebello. Easygrid: Toward a framework for the automatic grid enabling of legacy MPI applications. In *Concurrency and Computation: Practice Experience*, volume 16 (5), pages 425–432. John Wiley and Sons, 2004.

[10] R. Buyya. *High Performance Cluster Computing: Architectures and Systems*, volume 1. Prentice Hall, 1999.

[11] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *IEEE Computer Society Press*, 2000.

[12] D. F. Cardoso. Escalonameno estático de tarefas em ambientes computacionais heterogêneos sob modelo logp. Master's thesis, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brazil, 2004. (In Portuguese).

[13] C. Catlett and L. Smarr. Metacomputing. *Communications of the ACM*, 35(6):44–52, 1992.

[14] D. E. Culler, Richard M. Karp, D. A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.

[15] D.E. Culler, L.T.Liu, R.P.Martin, and C.O.Yoshikawa. Acessing fast network interfaces. *IEEE Micro*, 16(1):35–43, February 1996.

[16] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proc. 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.

[17] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. Harnessing the capacity of computational grid for high energy physics. In *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[18] G. Allen et al. The Cactus tools for grid applications. In *Cluster Computing*, pages 179–188, 2001.

[19] Universidade Federal Fluminense. *The EasyGrid Project's Research, Reference and Resource Library.* World Wide Web, http://easygrid.ic.uff.br, 2004.

[20] S. Fortune and J. Wyllie. Paralelism in random access machines. In *ACM Press*, 1978.

[21] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International journal of Supercomputing Applications*, 11(2):115–128, 1997.

[22] I. Foster and C. Kesselman. The globus project: A status report. *In IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[23] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrasctructure.* Morgan Kaufmann, 1999.

[24] M. Frumkin and R. Hood. Using grid benchmarks for dynamic scheduling of grid applications. *NAS Technical Report NAS-03-015*, October 2003.

[25] G. Iannello, M. Lauria, and S. Mercolino. Cross-platform analysis of fast messages for myrinet. *Lecture Notes in Computer Science*, 1362:217–231, 1998.

[26] Sun Microsystems Inc. *Sun Grid Engine.* http://www.sun.com/gridware/, 1998.

[27] F. Ino, N. Fujimoto, and K. Hagihara. LogGPS: A parallel computational model for synchronization analysis. In *Proceedings of the 8th ACM SIGPLAN symposium on Principles and practices of parallel programming*, pages 133–142, Snowbird, Utah, 2001. ACM Press.

[28] T. Kalinowski, I. Kort, and D. Trystram. List scheduling of general task graphs under LogP. *Parallel Computing*, 26(9):1109–1128, 2000.

[29] T. Kielmann, H. E. Bal, and K. Verstoep. Fast measurement of LogP parameters for message passing platforms. *In Proc*, 2001.

[30] J. F. Kurose and K. W. Ross. *Computer Networking.* Addison-Wesley, 2000.

[31] M. Litzkaw, M. Livny, M. Mutka, and M. W. Condor: A hunter of idle workstations. In *In proc. of the International Conference on Distributed Computing Systems*, June 1998.

[32] N. T. Moura and L.T. Menezes. Escalonameno dinâmico de tarefas em ambientes heterogêneos. Projeto final de curso, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brazil, 2004. (In Portuguese).

[33] MPICH-G2. *MPICH-G2.* World Wide Web, http://www.niu.edu/mpi/, 1998.

[34] A. Nascimento. Aglomeração de tarefas em arquiteturas paralelas com memória distribuída. Master's thesis, Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ, Brazil, 1999. (In Portuguese).

[35] International Journal of Supercomputing Applications. *MPI: a Message Passing Interface. Technical report.* University of Tenessee, 1995.

[36] University of Virginia. *Legion: A world wide virtual computer.* World Wide Web, http://legion.virginia.edu, 2003.

[37] C.H. Papadimitriou and M. Yannakakis. Towards an archicteture independent analysis of parallel algorithms. *SIAMJ Comp.*, 19:322–328, 1990.

[38] M. J. Quinn. *Parallel Computing Theory and Practice.* McGraw-Hill, 1994.

[39] Top500 Supercomputer Sites. *Top500 Supercomputer Sites.* World Wide Web, http://www.top500.org/list/2004/06/, 2004.

[40] D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and Answers about BSP. *Scientific Programming*, 6(3):249–274, Fall 1997.

[41] A. Tam and C. Wang. Realistic communication model for parallel computing on cluster. In *The Proceedings of the 1st IEEE International Workshop on Cluster Computing*, pages 92–101. IEEE Computer Society Press, December 1999.

[42] L.G. Valiant, editor. *A bridging Model for Parallel Computation.* In Communications of the ACM, 1990.

[43] T. L. Williams and R. J. Parsons. The heterogeneous Bulk Synchronous Parallel Model. *Lecture Notes in Computer Science*, 1800:102–105, 2000.

[44] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The network weather service. *In Proceedings of Supercomputing 1997*, 1997.

[45] R. Wolski, N. T. Spring, and J. Hayes. Predicting the CPU availability of time-shared Unix systems on the computational grid. *Cluster Computing*, 3(4):293–301, 2000.