Rodrigo Paiva Mendonça

Uma Proposta de Co-Escalonamento Adaptativo para um Ambiente de Computação Oportunista de Recursos Distribuídos

Florianópolis – SC

Agosto / 2008

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rodrigo Paiva Mendonça

Uma Proposta de Co-Escalonamento Adaptativo para um Ambiente de Computação Oportunista de Recursos Distribuídos

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Mário Antônio Ribeiro Dantas, Dr.

Florianópolis, Agosto de 2008

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Mário Antônio Ribeiro Dantas, Dr.
Departamento de Informática e Estatística - UFSC
Orientador

Prof. Frank Augusto Siqueira, Dr.
Departamento de Informática e Estatística - UFSC

Prof. Nelson Francisco Favilla Ebecken, Dr.
COPPE - UFRJ

Prof. Roberto Willrich, Dr.
Departamento de Informática e Estatística - UFSC

Agradecimentos

Ao meu orientador, professor Dr. Mário Dantas, pela confiança, orientação, paciência e conversas motivadoras ao longo desta pesquisa.

A todos os colegas do LaPeSD, especialmente ao Lucas, Luigi e Parra pelo incentivo e pelas sugestões que contribuíram para um melhor desenvolvimento deste trabalho. Ao companheiro de trabalho e pesquisa Aldelir que muito me ajudou com os experimentos realizados.

Aos meus pais Regina e Marcelino e a minha irmã Karol pelo carinho, confiança e apoio em todos os meus objetivos e sonhos.

À minha namorada Elisa e minha irmã Fran pela paciência que tiveram comigo, compreendendo as minhas ausências e me apoiando sempre nas dificuldades encontradas neste período.

Por fim, agradeço a todos que participaram de alguma forma para mais esta conquista da minha vida.

"O que for a profundeza do teu ser, assim será teu desejo.

O que for o teu desejo, assim será tua vontade.

O que for a tua vontade, assim serão teus atos.

O que forem teus atos, assim será teu destino."

Brihadaranyaka Upanishad IV, 4.5

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

ATHA-RSAE ..... *ATHA-Resource Scheduler ATHA Environment*

CMP ..... *Chip Level Multiprocessor*

CPU ..... *Central Processing Unit*

CRM ..... *Customer Relationship Management*

CT ..... Chaves Testadas por Segundo

CTG ..... Chaves Testadas por Segundo Global

ERP ..... *Enterprise Resource Planning*

EP ..... Elemento de Processamento (máquina ou *host*)

HTT ..... *Hyper-Threading Technology*

IP ..... *Internet Protocol*

MFLOPS ..... *Millions of Floating-point Operations per Second*

MPI ..... *Message Passing Interface*

OV ..... *Organização Virtual*

P ..... Performance

PG ..... Performance Global

PVM ..... *Parallel Virtual Machine*

SPG ..... *Speedup* de Performance Global

SCTG ..... *Speedup* de Chaves Testadas por Segundo Global

UP ..... Unidade de Processamento (processador ou *core* de processador)

VOIP ..... *Voice Over Internet Protocol*

Resumo

Configurações multi-core (CMP - *Chip Level Multiprocessor*) e de multi-processadores fracamente acoplados vêm sendo adotadas por muitas organizações com o propósito de aumentar o poder computacional para as suas aplicações. Contudo, o aumento no uso dessas configurações traz um novo desafio para a comunidade científica, que pode ser traduzido como: o limitado número de aplicações desenvolvidas visando aproveitar toda a capacidade dessas configurações. Sistemas de computação oportunista, os quais buscam usar a capacidade ociosa dos recursos disponíveis, também precisam se adaptar a essas novas configurações. Caso contrário, não é esperado que esses sistemas utilizem de forma eficiente os recursos com múltiplas unidades de processamento.

Nesta dissertação, é realizada uma análise de uma proposta de co-escalonamento adaptativo para ser agregada a um sistema de computação oportunista. O principal objetivo é aproveitar toda a capacidade ociosa de recursos multi-core e de multiprocessadores que estejam disponíveis em uma determinada organização. Esta proposta adota o paradigma de múltiplas *threads* para alcançar um melhor aproveitamento de cada unidade de processamento dos *hosts* escravos.

Resultados empíricos indicam que o esforço obteve sucesso em garantir o uso eficiente dos recursos multiprocessados. Também foi possível se notar uma redução no tempo total de execução de uma aplicação utilizada para demonstração. Em adição, notou-se a manutenção do desempenho quando o número de máquinas desse ambiente distribuído foi aumentado gradativamente.

Palavras-chave: Computação Oportunista, Co-Escalonamento, Multi-Core.

Abstract

Multi-core (CMP - *Chip Level Multiprocessor*) and multi-processor loosely coupled configurations have been considered interesting architectures to increase computational power in many organizations. However, these environments brought a new challenge for the scientific community that can be understood as the limited number of applications developed to take advantage of these new configurations. Opportunist software environments also need to adapt to these new configurations, otherwise it is not expected that these software packages will be able to use efficiently the idle capacity from these architectures.

In this dissertation, it is presented an empirical study of an adaptive coscheduling approach, using an opportunist package to gather resources from an environment compound by multi-core and multi-processor loosely coupled elements available in an organization. The proposal approach adopts multiple threads to enhance the use of each processing unit of the slave hosts.

Empirical results proved that the proposal was successful in gathering idle resources from these configurations, enhancing the performance of a demonstration application, including when we increased the number of machines available in this environment.

Keywords: Opportunist Computing, Coscheduling, Multi-Core.

1 Introdução

1.1 Considerações Iniciais

Vem sendo verificada uma tendência no uso de configurações multi-core (CMP) e de multi-processadores nas organizações, tendo em vista melhores desempenhos para as suas aplicações. No entanto, como (FORTES, 2006) observa, essas configurações sozinhas não representam a resposta para um grande número de aplicações. Em outras palavras, um *middleware*, adotando alguns parâmetros, é necessário para auxiliar na submissão de aplicações para esses ambientes.

Os sistemas de processamento oportunista (e.g. SETI@Home (SETI@HOME, 2008), Boinc (BOINC, 2008), POPCORN (NISAN et al., 1998), Charlotte (BARATLOO et al., 1996) e ATHA (HOSKEN; DANTAS, 2004)), podem ser classificados como *middlewares* que permitem a execução de aplicações complexas em grandes configurações de sistemas distribuídos. Esses sistemas, também chamados de *Metacomputing Systems*, são usualmente projetados para reunir os recursos ociosos disponíveis em uma rede para executar aplicações que requerem o uso intenso de processadores. Alguns exemplos clássicos dessas aplicações são: os chamados *Grand Challenge Applications* (GRANDCHALLENGE, 2008); seqüenciamento molecular (STRUMPEN, 1993); otimização combinatória (PAPADIMITRIOU; STEIGLITZ, 1982); e pesquisa de números primos (GIMPS, 2008).

Um dos principais problemas encontrados na utilização desses sistemas é a dis-

tribuição ou escalonamento de tarefas entre os recursos disponíveis em um grande ambiente distribuído. Segundo (PAPADIMITRIOU; STEIGLITZ, 1982) este problema é considerado NP-completo e por esta razão é comum a utilização de métodos heurísticos como critérios para realizar essa distribuição.

Nesta dissertação é apresentado um estudo empírico do uso de uma estratégia de escalonamento de tarefas para um sistema de processamento oportunista, executando em um ambiente dotado de recursos com múltiplas unidades de processamento. O foco principal da estratégia utilizada é reunir toda a capacidade ociosa de um ambiente de configurações multi-core e de multi-processadores fracamente acoplados, para ser usado de forma eficiente por uma aplicação complexa executada através de um sistema de computação oportunista.

1.2 Motivação

Diante do exposto até o momento, podemos inferir que ao executar aplicações desenvolvidas sem o uso de técnicas que visam aproveitar as configurações multi-processadas disponíveis em um ambiente, esta não será capaz de utilizar de forma eficiente toda a capacidade de processamento ocioso dessas configurações. Sendo assim, a principal motivação do presente estudo foi propor uma estrutura de escalonamento adaptativo no sistema de computação oportunista denominado ATHA, para que este seja capaz de aproveitar de uma maneira eficiente toda a capacidade computacional dos recursos disponíveis em uma rede, ainda que estes possuam múltiplas unidades de processamento e a aplicação utilizada não tenha sido desenvolvida buscando utilizar de forma eficiente essas configurações.

1.3 Objetivos

O **objetivo geral** desta pesquisa pode ser entendido como uma investigação dos resultados obtidos ao agregar uma estratégia de escalonamento adaptativo em um sistema de processamento oportunista, quando consideramos um ambiente distribuído real dotado de configurações multi-core e de multi-processadores, frequentemente encontrado nas organizações.

Os **objetivos específicos** desta dissertação foram:

- Realizar uma revisão bibliográfica sobre os tipos de sistemas distribuídos que buscam utilizar os ciclos ociosos dos computadores de uma rede para auxiliar na resolução de problemas complexos;
- Pesquisar estratégias de escalonamento de processos em sistemas distribuídos e escolher uma técnica e as regras para realizar uma distribuição de tarefas de forma eficiente em um ambiente de processamento oportunista dotado de recursos com múltiplas unidades de processamento;
- Realizar um experimento com um sistema de computação oportunista largamente conhecido, para avaliar o seu comportamento diante de computadores multiprocessados;
- Adaptar o módulo mestre do sistema ATHA, para que quando um recurso escravo com múltiplas unidades de processamento for identificado, este seja capaz de enviar novas tarefas para este recurso, o que não é possível na versão original do ATHA;
- Adaptar o módulo escravo para receber mais do que uma tarefa e executá-las simultaneamente utilizando múltiplas *threads* da aplicação principal;
- Preparar um ambiente experimental com computadores heterogêneos, a fim

de analisar o comportamento da estratégia proposta diante de uma situação mais próxima do real em muitas organizações; e

- Analisar o comportamento do ambiente diante de um número crescente de máquinas, verificando com isso a escalabilidade do sistema.

1.4 Metodologia

Para o desenvolvimento deste trabalho, estudou-se na literatura os conceitos relacionados aos objetivos relatados. Na seqüência foram realizados estudos de casos em um ambiente real de máquinas heterogêneas, comparando os resultados quando utilizado ou não a estratégia de escalonamento proposta. Foi avaliada também a escalabilidade do ambiente diante de um número crescente de máquinas.

Para isso, foi desenvolvido um módulo escalonador no sistema ATHA com o objetivo de realizar a distribuição das tarefas de uma forma mais flexível, quando consideramos a possibilidade de encontrar configurações multi-core e de multi-processadores no ambiente distribuído utilizado.

1.5 Organização do texto

Este documento está organizado como segue:

- O Capítulo 2 relata uma visão geral das arquiteturas distribuídas e as principais definições relacionadas a sistemas distribuídos. Neste capítulo descrevemos também, alguns conceitos sobre programação paralela e distribuída;

- O Capítulo 3 apresenta uma revisão bibliográfica sobre escalonamento de processos em sistemas distribuídos, detalhando as principais classes de uma taxonomia amplamente aceita no meio científico;

- O Capítulo 4 apresenta algumas pesquisas relacionadas com o presente trabalho, já realizando um primeiro experimento com o conhecido sistema oportunista (BOINC, 2008), o qual possui algumas características semelhantes ao sistema usado neste trabalho de pesquisa. Na seqüência descrevemos o sistema de processamento oportunista utilizado em nossos experimentos e concluímos este capítulo detalhando a abordagem de co-escalonamento proposta;

- O Capítulo 5 caracteriza os ambientes experimentais utilizados durante os estudos de caso e os principais resultados empíricos alcançados nesses experimentos;

- Por fim, no Capítulo 6, as considerações finais e trabalhos futuros são apontados.

2 *Ambientes Distribuídos e Paralelos*

2.1 Introdução

Ao longo deste capítulo apresentamos alguns conceitos sobre sistemas distribuídos e paralelos. Inicialmente, descrevemos uma classificação de arquiteturas computacionais muito aceita no meio acadêmico, para na seqüência apresentar uma definição de sistemas distribuídos. Esses conceitos são necessários para a compreensão dos próximos capítulos desta dissertação.

2.2 Classificação das Arquiteturas de Computadores

Dentre as classificações encontradas na literatura para as arquiteturas de computadores seqüenciais e paralelos, a proposta por Flynn, em (FLYNN, 1972), é a mais aceita (TANENBAUM; STEEN, 2006)(QUINN, 1994). Isso, pelo fato de que esta classificação é baseada apenas nos fluxos de instruções e de dados. Dessa forma, Flynn estabeleceu quatro categorias para as arquiteturas computacionais:

- **SISD (*Single Instruction Single Data*)**: um único fluxo de instruções é executado sobre um único conjunto de dados. Um exemplo dessa classe são os computadores pessoais com apenas uma unidade de processamento;
- **SIMD (*Single Instruction Multiple Data*)**: envolve múltiplas unidades

de processamento, onde apenas um fluxo de instrução é executado sobre várias combinações de dados. Alguns supercomputadores (e.g. ILLIAC IV, GF11 e CM-2) se enquadram nessa classe;

- **MISD (*Multiple Instruction Single Data*)**: utiliza vários fluxos de instruções que são executados sobre apenas um conjunto de dados. Não se tem conhecimento de computadores classificados nesse tipo (TANENBAUM; STEEN, 2006)(DANTAS, 2005);

- **MIMD (*Multiple Instruction Multiple Data*)**: executa vários fluxos de instruções sobre diferentes conjuntos de dados. As configurações com múltiplas unidades de processamento que executam instruções de forma paralela e independente, são exemplos de computadores pertencentes a essa classe.

Os ambientes distribuídos que serão considerados ao longo deste trabalho enquadram-se nesta última classe, por isso, nos ocuparemos em detalhar apenas a classe MIMD no decorrer deste capítulo. A figura 1 apresenta uma subdivisão das arquiteturas MIMD muito encontrada na literatura (TANENBAUM; STEEN, 2006) (DANTAS, 2005). Nela encontramos duas classes para as arquiteturas MIMD: os multiprocessadores e os multicomputadores.

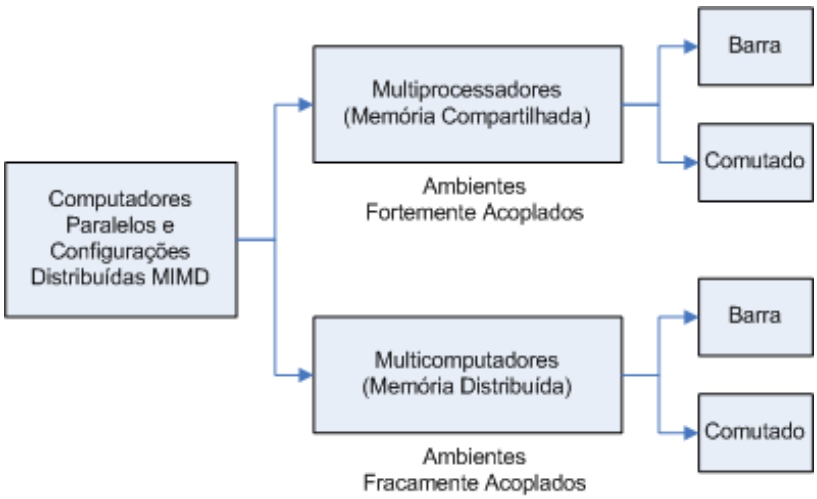

Figura 1: Arquiteturas MIMD segundo (TANENBAUM; STEEN, 2006).

- **Multiprocessadores (memória compartilhada)**: nesta primeira classe, existe apenas uma memória virtual que é compartilhada entre vários processadores (TANENBAUM; STEEN, 2006). Esta arquitetura também é conhecida como fortemente acoplada (*tightly coupled*), pelo fato dos processadores e memórias físicas estarem interligados através do seu sistema local de interconexão (DANTAS, 2005). A principal vantagem no uso dessas arquiteturas é o pequeno *overhead* existente na comunicação entre os processos. Entretanto, existe uma restrita escalabilidade da quantidade de processadores possíveis nessas configurações, devido a limitações como o custo da fabricação desses recursos e também problemas de dissipação de calor. Na figura 2 podemos verificar um exemplo ilustrativo, adaptado de (DANTAS, 2005), dos multiprocessadores;

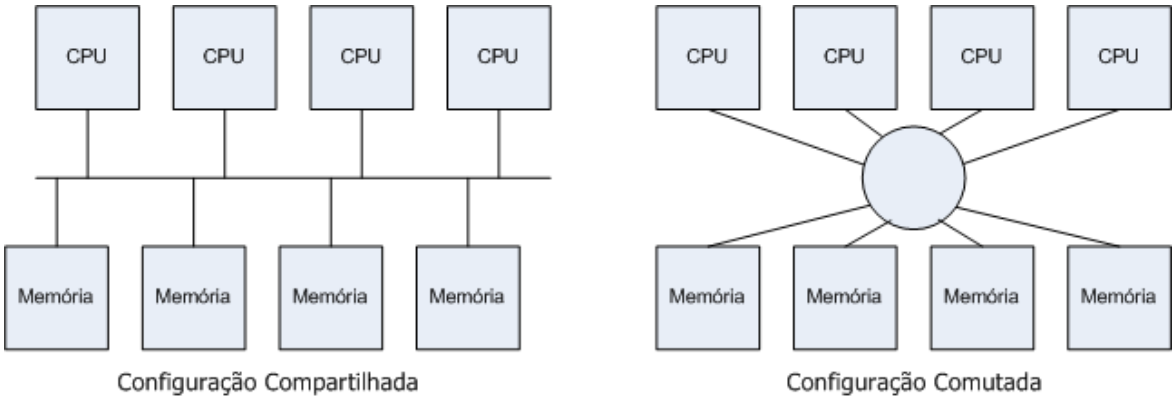

Figura 2: Exemplos Genéricos de Configurações de Multiprocessadores.

- **Multicomputadores (memória distribuída)**: estas configurações possuem processadores e memórias locais independentes, sendo realizada a comunicação entre as máquinas através de uma rede. Por isso, essa arquitetura também é conhecida como fracamente acoplada (*loosely coupled*) (TANENBAUM; STEEN, 2006). Essas arquiteturas solucionam a deficiência dos multiprocessadores com relação à escalabilidade, todavia traz consigo um maior *overhead* na comunicação entre os processos, visto que os dados compartilhados entre proces-

sadores, são transferidos através de uma rede utilizando trocas de mensagens (DANTAS, 2005). A figura 3, também adaptada de (DANTAS, 2005), ilustra alguns exemplos possíveis das configurações de multicomputadores.

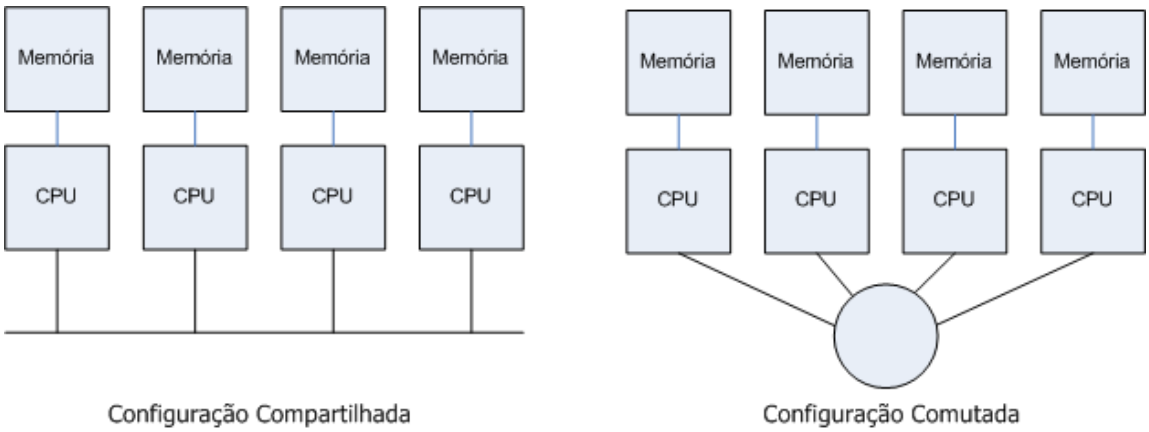

Figura 3: Exemplos Genéricos de Configurações de Multicomputadores.

Uma vez apresentada a classificação das arquiteturas computacionais, podemos enquadrar o ambiente distribuído utilizado nos experimentos desta dissertação, como um MIMD de multicomputadores. No entanto, algumas das máquinas utilizadas neste ambiente possuíam arquiteturas de multiprocessadores, em outras palavras, foi utilizada uma arquitetura de multicomputadores com multiprocessadores. Dessa forma, seguiremos com algumas definições de sistemas distribuídos utilizados nesse tipo de ambiente.

2.3 Sistemas Distribuídos

De acordo com (PFISTER, 1998) e (BUYYA, 1999b), quando buscamos alcançar melhor desempenho de uma aplicação, existem três caminhos a seguir: utilizar recursos computacionais mais rápidos; aperfeiçoar os algoritmos e as técnicas utilizadas para resolver o problema; ou ainda, dividir o processamento entre vários computadores. O uso da primeira alternativa traz consigo algumas restrições financeiras, devido ao alto custo na fabricação desses recursos, e também físicas com relação ao

aquecimento dos mesmos (EL-REWINI; LEWIS, 1998). A segunda opção é limitada no sentido da complexidade de alguns problemas em que não se tem o conhecimento de algoritmos mais otimizados. Por fim, a terceira alternativa se apresenta como uma solução para quando não for possível aplicar nenhuma das duas anteriores. Neste sentido, são utilizados os sistemas distribuídos e paralelos.

Na literatura encontramos várias definições de sistemas distribuídos (TANENBAUM; STEEN, 2006) (COULOURIS; DOLLIMORE; KINDBERG, 2005) (DANTAS, 2005). No entanto, para este trabalho iremos considerar o conceito apresentado por (TANENBAUM; STEEN, 2006), que descreve os sistemas distribuídos como uma coleção de computadores independentes que aparecem para o usuário de um sistema como um único computador. Com relação às máquinas envolvidas em um sistema distribuído, destacamos ainda a possibilidade de encontrar *software* e *hardware* heterogêneos se comunicando através de troca de mensagens (COULOURIS; DOLLIMORE; KINDBERG, 2005). De acordo com (TANENBAUM; STEEN, 2006), para a construção de um sistema distribuído deve-se ter em mente as seguintes preocupações:

- **Transparência**: os recursos de um sistema distribuído devem estar disponíveis para serem acessados como um sistema único, independente de onde estejam localizados;
- **Flexibilidade**: um sistema distribuído deve ser flexível e adaptável com relação à diversidade de recursos encontrados no mesmo;
- **Confiabilidade**: a idéia é que se uma máquina falhar, as outras máquinas irão finalizar o processamento em andamento. Isso torna o sistema distribuído mais confiável que um sistema com um único elemento de processamento (EP);
- **Desempenho**: quando executar uma aplicação em um sistema distribuído, o tempo de processamento não deve ser pior do que executar a mesma aplicação em um único processador;

- **Escalabilidade**: define a capacidade de crescimento de um sistema distribuído, que deve ser capaz de aumentar os recursos disponíveis para melhorar cada vez mais o desempenho das aplicações.

A seguir, serão apresentados alguns dos principais sistemas distribuídos utilizados nas organizações, são eles: os *clusters* e os *grids*. Descreveremos também o tipo de sistema utilizado para os experimentos desta dissertação, que usualmente na literatura é conhecido como sistema de computação oportunista.

2.3.1 *Cluster*

De acordo com (BUYYA, 1999b), um *cluster* (ou agregado computacional), é um tipo de sistema de processamento paralelo e distribuído formado por um conjunto de computadores independentes, e que trabalham juntos como se fosse um único. Usualmente esse conjunto de computadores faz parte do ambiente de um único proprietário, sendo gerenciados por uma unidade administrativa central.

A principal vantagem desses ambientes trata-se do *overhead* de comunicação entre os processos, que geralmente é pequeno. Isso ocorre pelo fato de que usualmente um *cluster* está interconectado por uma LAN (*Local Area Network*), que é administrada por um órgão único. No entanto, isso torna a sua escalabilidade restrita à disponibilidade de recursos financeiros da organização que o administra.

Segundo (DANTAS, 2005), essas configurações podem ser projetadas de uma forma **dedicada** a executar aplicações específicas de uma organização. Ou ainda, **não dedicadas**, quando os computadores executam aplicações dos usuários locais e ainda participam do processamento de algumas tarefas de aplicações que exigem um maior poder computacional.

Apesar dos recursos computacionais utilizados para os testes empíricos deste trabalho fazerem parte de uma única organização, o propósito do sistema de proces-

samento oportunista usado é um ambiente que pode envolver recursos espalhados na internet, ou seja, pertencerem a vários domínios administrativos. Dessa forma iremos relatar algumas características dos sistemas distribuídos do tipo *Grid*, os quais possuem essa característica.

2.3.2 *Grid*

De acordo com (BUYYA, 2008), um *grid* (grade ou malha) computacional é um tipo de sistema paralelo e distribuído que permite o compartilhamento, seleção, e agregação de recursos autônomos geograficamente distribuídos, de acordo com a disponibilidade, capacidade, performance, custo e qualidade de serviço requeridas pelos usuários deste sistema.

A nomenclatura *grid* tem a sua origem na idéia do funcionamento das redes de energia elétrica, ou seja, quando um usuário utiliza um recurso ou serviço, neste caso a energia, ele não se preocupa sobre a origem deste recurso ou serviço. De forma análoga temos os *grids* computacionais, que procuram disponibilizar os seus recursos ou serviços de maneira geograficamente distribuídos através da Internet para usuários utilizarem sem a preocupação de onde o recurso ou serviço está sendo fornecido (FOSTER; KESSELMAN, 1999) (DANTAS, 2005).

Os recursos e serviços do um ambiente de *grid* computacional são disponibilizados por entidades conhecidas como organizações virtuais (OVs). Essas entidades podem ser empresas, centros de pesquisas, universidades, entre outros, e cada uma delas possui as suas próprias políticas de compartilhamento, que define como os usuários do *grid* poderão utilizar os seus recursos e serviços (DANTAS, 2005).

A existência de várias entidades que disponibilizam recursos e serviços em um *grid* pode ser entendida como a principal diferença entre este tipo de ambiente distribuído e os *clusters* computacionais, ou seja, enquanto em um ambiente de *grid*

são encontradas várias organizações que disponibilizam e gerenciam os recursos, nos *clusters* o ambiente usualmente é gerenciado por apenas uma unidade administrativa central.

No entanto, os *grids* computacionais possuem alguns pontos fracos, que estão relacionados também à grande quantidade e a diversidade de configurações dos recursos disponíveis nestes ambientes, entre eles encontramos: a dificuldade na gerência dos recursos; baixas velocidades de comunicação em algumas redes heterogêneas; e a complexidade de se desenvolver *softwares* para lidar com esse tipo de ambiente (TANENBAUM, 2001).

2.3.3 Computação Oportunista

Uma outra classe de sistemas distribuídos e paralelos são os ambientes de computação paralela oportunista, que tem como a principal premissa, aproveitar da melhor forma todo o poder computacional disponível em uma rede. Em outras palavras, esses sistemas buscam "roubar" os ciclos ociosos dos computadores disponíveis em uma Intranet ou Internet, para auxiliar no processamento de tarefas complexas e que exigem um uso intenso de processador (DANTAS, 2005).

Esses sistemas tornam-se mais interessantes quando consideramos alguns estudos apresentados em (FOSTER; KESSELMAN, 1999) indicando que apenas 30% do poder de processamento disponível nas organizações acadêmicas e comerciais são utilizados de fato. Sendo assim, um sistema que utiliza os ciclos ociosos dos computadores de uma rede é uma alternativa financeiramente interessante para organizações que não conseguem adquirir supercomputadores, embora possuam uma grande quantidade de computadores convencionais que geralmente são utilizados como estações de trabalho.

A tabela 1, adaptada de (DANTAS, 2005), apresenta as principais características

dos três sistemas distribuídos (Oportunista, *cluster* e *grid*) discutidos anteriormente.

Tabela 1: Diferenças entre as configurações: Oportunista, Cluster e Grid

Características	Oportunista	*Cluster*	*Grid*
Domínio	Múltiplos	Único	Múltiplos
Nós	Milhões	Milhares	Milhões
Segurança do Processamento e Recurso	Necessária	Desnecessária	Necessária
Custo	Baixo, uso exclusivo de recursos ociosos	Alto, pertencente a um único domínio	Alto, todavia dividido entre domínios
Granularidade do problema	Muito grande	Grande	Muito Grande
Sistema Operacional	Heterogêneo	Geralmente Homogêneo	Heterogêneo

O sistema distribuído utilizado nos experimentos desta dissertação é classificado como um sistema de computação oportunista, ou seja, são aproveitados os ciclos ociosos dos computadores disponíveis em uma rede, para o processamento de aplicações complexas. Com o objetivo de compreender o funcionamento das aplicações paralelas e distribuídas, descrevemos na seção seguinte os principais conceitos sobre esse assunto.

2.4 Programação Paralela e Distribuída

A utilização de algoritmos paralelos, pode ser entendida como uma alternativa para se alcançar melhores resultados no processamento de aplicações complexas. Uma classificação muito usada na literatura (FREEH, 1996) se baseia na forma com que o código paralelo foi gerado. Nesse aspecto, o paralelismo é dividido entre implícito e explícito.

- **Paralelismo Implícito**: busca a paralelização sem a intervenção do programador, realizando de forma automática a análise, decomposição dos dados

e tarefas, e também o escalonamento das mesmas. Alguns compiladores paralelizadores permitem esta técnica, como por exemplo: *High Performance Fortran* (HPF, 2008);

- **Paralelismo Explícito**: neste tipo, o programador é o responsável por desenvolver o programa utilizando técnicas de programação paralela e já definindo neste momento quais partes do programa serão processadas de forma paralela. Exemplos muito conhecidos no meio acadêmico deste tipo de paralelismo, correspondem à programação paralela utilizando as bibliotecas MPI (MPI, 2008) e PVM (PVM, 2008).

Apesar das facilidades inerentes com a utilização do paralelismo implícito, geralmente espera-se que o programador, o especialista da aplicação, possua o melhor conhecimento para decidir como paralelizar o código. Dessa forma, usualmente o uso do paralelismo explícito permite alcançar melhores resultados.

2.4.1 Granularidade

Um outro fator relevante sobre o uso eficiente de algoritmos paralelos está diretamente ligado à maneira que o programa foi particionado. O tamanho e complexidade dos blocos das aplicações possuem uma grande influência no tempo total de processamento das mesmas (GERASOULIS; YANG, 1993). Dessa forma introduzimos o conceito de granularidade de uma tarefa, que segundo (SHIRAZI; KAVI; HURSON, 1995) é a razão entre o tempo de computação pelo tempo de comunicação de uma aplicação. Por exemplo, uma aplicação é considerada de granularidade fina (*fine grain*) quando a razão do tempo de computação pelo de comunicação é pequena, entretanto quando essa razão é um valor alto a aplicação é caracterizada como de granularidade grossa (*coarse grain*) (KRUATRACHUE; LEWIS, 1988).

Dessa forma, podemos inferir que as aplicações de granularidade grossa (e.g.

multiplicação de matrizes) apresenta certa independência quanto à eficiência da rede de interconexão. Por outro lado, o desempenho de aplicações de granularidade fina (e.g. transposição de matrizes) está muito mais dependente dos dispositivos de interconexão utilizados, conforme experimentos realizados em (PINTO; MENDONÇA; DANTAS, 2008).

2.4.2 Algoritmos Paralelos

Com relação ao desenvolvimento de algoritmos paralelos, Foster (1995) propõe alguns passos, ilustrados na figura 4, que devem ser seguidos para a construção dos mesmos. Na seqüência descrevemos mais detalhes sobre cada um desses passos.

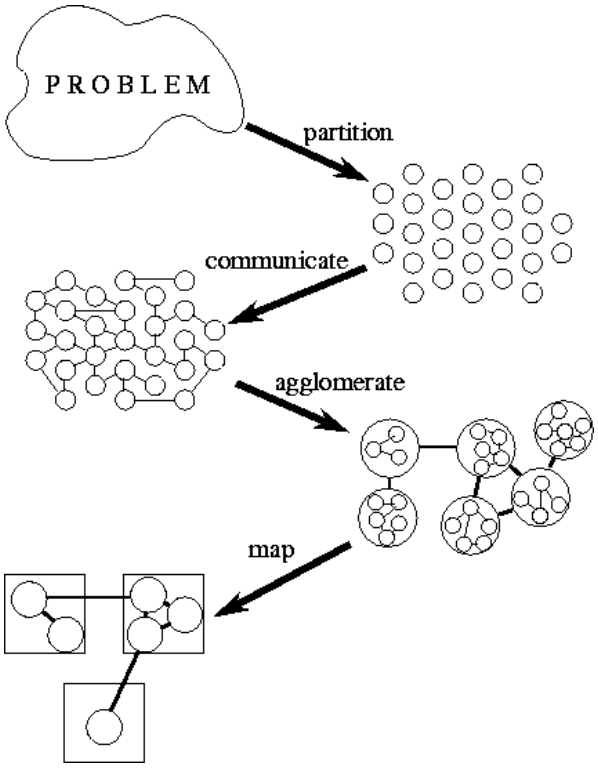

Figura 4: Metodologia de Projeto para Algoritmos Paralelos (FOSTER, 1995).

2.4.2.1 Particionamento (*Partition*)

Nesta fase é feita uma análise dos pontos possíveis de paralelização do programa e realizada a divisão do problema principal em problemas menores. Esta decomposição

pode ser de **dados**, isto é, particiona-se os dados do problema e posteriormente é feita a divisão do processamento utilizando os dados já divididos. Ou ainda, essa divisão pode ser **funcional**, que corresponde a particionar as instruções da aplicação e em seguida submeter para cada pedaço de instrução os seus dados correspondentes.

2.4.2.2 Comunicação (*Communicate*)

Com o particionamento do problema principal realizado pelo passo anterior, se faz necessário coordenar a execução dessas novas tarefas, estabelecendo uma estrutura de comunicação entre as mesmas; em outras palavras, é preciso criar uma associação entre as tarefas paralelas que possuem dependência de dados entre elas.

Foster (1995) também divide os métodos de comunicação em quatro classes: local e global; estruturada e não estruturada; estática e dinâmica; e síncrono e assíncrono.

- Na comunicação **local** cada tarefa comunica-se com um grupo pequeno de tarefas enquanto que na **global** essa comunicação ocorre entre várias tarefas;
- Na comunicação **estruturada**, as tarefas montam uma estrutura bem definida, como por exemplo, uma árvore ou uma grade. Já na **não estruturada**, as tarefas podem montar grafos arbitrários;
- Na **estática** a comunicação ocorre sempre entre as mesmas tarefas; já na **dinâmica** não é garantido que a comunicação será realizada entre os mesmos parceiros sempre;
- A comunicação **síncrona** é realizada de uma forma coordenada entre as partes da comunicação; já na **assíncrona** essa comunicação pode ser realizada sem a preocupação se o receptor irá cooperar com a comunicação.

2.4.2.3 Aglomeração (*Agglomerate*)

Este passo é responsável basicamente por avaliar os passos anteriores em termos de custo de implementação e necessidades de performance. Dessa forma, é possível combinar algumas dessas tarefas em tarefas maiores, buscando facilitar a implementação e obter melhores desempenhos, tornando a aplicação menos influenciada pela rede de interconexão dos *hosts*, conforme (PINTO; MENDONÇA; DANTAS, 2008). Os principais objetivos desta fase, segundo Foster (1995) são: **aumentar a granularidade** diminuindo custos de comunicação; **preservar a flexibilidade**; e **reduzir custos da engenharia do software**

2.4.2.4 Mapeamento (*Map*)

Por fim, temos o passo do mapeamento, que é responsável por realizar a submissão das tarefas para os elementos de processamento disponíveis, buscando maximizar a sua utilização e minimizar o tempo total de processamento da aplicação. Esse mapeamento, também conhecido como escalonamento, pode ser realizado através de várias técnicas, conforme serão apresentados no capítulo 3, que é dedicado aos conceitos de escalonamento em sistemas paralelos e distribuídos.

2.4.3 Paradigmas de Programação

No tocante aos paradigmas de programação paralela, (BUYYA, 1999b) descreve os seguintes paradigmas como os principais na programação paralela:

- **Mestre/Escravo (*Master/Slave*)**: nesta situação, um processo mestre é criado, sendo este o responsável por dividir o problema em tarefas menores, distribuí-las entre os processos escravos, reunir os resultados parciais obtidos pelos escravos e apresentar o resultado final do problema. Os processos escravos, por vez, são responsáveis basicamente por receber as suas tarefas,

executá-las e retornar o resultado parcial para o processo mestre;

- **SPMD - *Simple Program Multiple Data***: neste paradigma, ocorre inicialmente uma divisão dos dados que serão processados pelo problema e na seqüência, estes dados são enviados para serem processados em máquinas diferentes utilizando uma mesma instrução de código. Nesse tipo, usualmente existe a necessidade dos processos se comunicarem, para manter um sincronismo na execução do problema (BUYYA, 1999b);

- **Pipelining de Dados**: neste caso, o problema principal é dividido em uma seqüência de etapas, sendo cada uma dessas submetidas para uma máquina diferente e os dados gerados como resultado em uma etapa são utilizados na etapa seguinte;

- **Dividir e Conquistar**: aqui, o problema é dividido em novos problemas, que poderão ser novamente subdivididos de forma independente, sendo os seus resultados combinados no final do processamento e apresentando assim, um resultado final do problema. Pelo fato desses subproblemas serem independentes, não existe a necessidade da comunicação entre esses processos (BUYYA, 1999b);

- **Paralelismo Especulativo**: este paradigma é utilizado quando existe a dificuldade de empregar os outros paradigmas apresentados anteriormente, decorrente de complexas dependências de dados de alguns problemas (BUYYA, 1999b). Neste caso o problema é dividido em pedaços pequenos onde é utilizada a estratégia de especulação ou processamento otimista a fim de facilitar o paralelismo.

2.5 Resumo

Este capítulo apresentou uma visão geral dos principais conceitos sobre sistemas distribuídos e paralelos. Inicialmente descrevemos a classificação das arquiteturas computacionais proposta por Flynn, na seqüência relatamos a divisão da classe MIMD, detalhando características sobre multiprocessadores e multicomputadores. Definimos sistema distribuído, apresentando também alguns dos principais tipos de sistemas distribuídos. Por fim, apresentamos a metodologia proposta por Foster para projetar algoritmos paralelos e descrevemos os principais paradigmas de algoritmos paralelos utilizados.

3 Escalonamento em Sistemas Distribuídos e Paralelos

3.1 Introdução

Este capítulo apresenta alguns dos principais conceitos sobre escalonamento de processos e balanceamento de carga em sistemas distribuídos, os quais serão importantes para o entendimento dos próximos capítulos deste trabalho.

3.2 Escalonamento de Processos

Como comentando no capítulo 2, o uso de ambientes distribuídos de grande escala têm sido uma alternativa interessante para melhorar o desempenho no processamento de aplicações complexas. No entanto, com o uso desses ambientes torna-se necessário decidir qual a melhor forma de distribuir as tarefas entre os recursos disponíveis no ambiente. Esse esforço é essencial quando se tem como objetivos: minimizar o tempo de execução das aplicações; minimizar os retardos de comunicação; e maximizar a utilização dos recursos (CASAVANT; KUHL, 1988). Na literatura essa distribuição de tarefas é usualmente conhecida como escalonamento (*scheduling*) de processos (SHIRAZI; KAVI; HURSON, 1995) (DANTAS, 2005).

De acordo com (CASAVANT; KUHL, 1988), um sistema de escalonamento possui três principais componentes conforme ilustrado na figura 5. Os **recursos** são os elementos de processamento (EPs), memórias, rede de comunicação. Os **con-**

sumidores correspondem as aplicações e tarefas que são submetidas ao ambiente computacional. Por fim, existe o **escalonador** de tarefas com as suas **políticas** que serão utilizadas para decidir a forma que será realizada a distribuição das tarefas.

Figura 5: Sistema de Escalonamento.

Com relação às técnicas de escalonamento de processos existentes, a classificação mais aceita na comunidade científica é a proposta em (CASAVANT; KUHL, 1988) e ilustrada na figura 6. O autor classifica inicialmente os métodos de escalonamento de uma forma hierárquica (figura 6) e em seguida de uma forma não hierárquica, sendo possível encontrar características dessa segunda classificação em qualquer sistema de escalonamento.

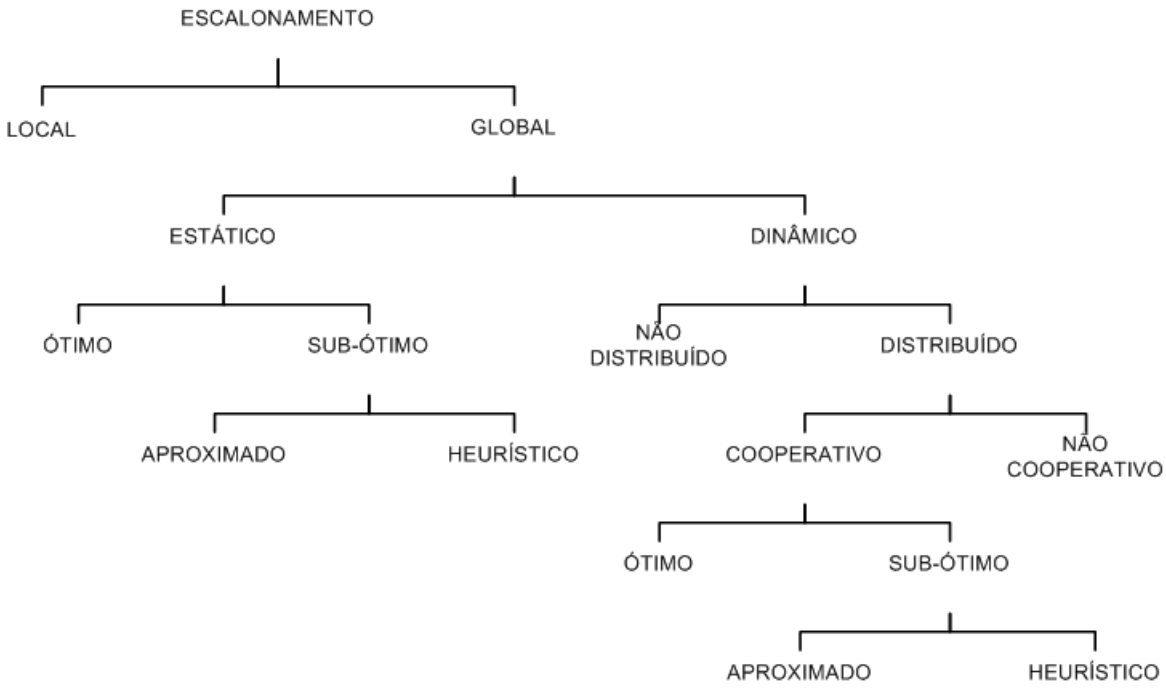

Figura 6: Taxonomia de Escalonamento Proposta por (CASAVANT; KUHL, 1988).

3.2.1 Local *versus* Global

O nível mais alto da classificação de Casavant divide escalonamento entre local e global. O primeiro está relacionado à atribuição de fatias de tempo de um processador para os processos. Já o global tem o foco em decidir qual dos *hosts*, entre vários disponíveis em uma rede, é o mais capacitado para executar uma determinada tarefa, segundo algumas políticas bem definidas.

O escalonamento local usualmente é tratado pelo sistema operacional (SHIRAZI; KAVI; HURSON, 1995). Por isso, nos ocuparemos em detalhar a classificação de Casavant a partir do escalonamento global, já que esse método é o responsável pela distribuição de tarefas entre os recursos disponíveis em um sistema distribuído como o estudado nesta pesquisa.

3.2.2 Estático *versus* Dinâmico

Pela taxonomia de Casavant, o escalonamento global está dividido entre estático e o dinâmico. Essa classificação indica em que momento do processamento da aplicação é realizada a decisão de como as tarefas serão distribuídas.

No escalonamento estático, essa decisão é realizada antes do processamento da aplicação, sendo geralmente baseada em informações da aplicação e dos recursos computacionais, já coletadas previamente. Além disso, uma vez definida essas regras, elas não poderão ser alteradas durante a execução da aplicação.

Esse método é indicado para ambientes homogêneos nos quais se conheça, com antecedência, o comportamento das aplicações submetidas no mesmo. Essa técnica de escalonamento tem como principal vantagem o pequeno *overhead* no "raciocínio" do escalonador, visto que as regras para essa decisão já são conhecidas no momento da distribuição das tarefas. As desvantagens desse método estão relacionadas com a grande quantidade de ambientes distribuídos dotados de configurações heterogêneas

e de aplicações com diferentes características, que são utilizadas pelas organizações. Em outras palavras, esse tipo de escalonamento muitas vezes é inadequado para ser utilizado em ambientes distribuídos reais.

No escalonamento dinâmico as regras utilizadas para a tomada de decisão do escalonador são coletadas durante o processamento da aplicação. Essas regras, em tempo de execução, buscam transferir as tarefas mais "pesadas" para processadores menos carregados, com o objetivo de melhorar o desempenho de uma aplicação (EAGER; LAZOWSKA; ZAHORJAN, 1986).

Ambientes com recursos heterogêneos, ou até mesmo onde não se tem conhecimento da configuração dos recursos disponíveis e das aplicações utilizadas, indica-se o uso de técnicas de escalonamento dinâmico (DANTAS, 2005). Nesse tipo, a distribuição das tarefas é realizada durante o processamento da aplicação, utilizando informações que serão disponibilizadas ao escalonador durante o processamento da aplicação.

O uso desse tipo de escalonamento traz consigo um maior *overhead* para a interpretação das informações na tomada de decisão de acordo com o algoritmo de escalonamento utilizado, todavia é o tipo mais indicado para a maioria dos ambientes distribuídos encontrados (DANTAS, 2005). Podemos classificar o sistema distribuído utilizado como objeto de estudo nesta dissertação como do tipo dinâmico, pelo fato de que durante a execução da aplicação principal é possível a entrada de novos *hosts* no ambiente distribuído e o escalonador deve ser capaz de "perceber" essas máquinas e considerá-las como recursos para o ambiente, levando em consideração ainda a configuração desses novos escravos.

3.2.3 Distribuído *versus* Não Distribuído

O escalonamento dinâmico é dividido entre distribuído e não distribuído. Essa classificação leva em consideração quem é o responsável pela distribuição das tarefas, ou seja, se essa regra está centralizada em um *host* ou distribuída entre vários. Em outras palavras, podemos dizer que uma estratégia de escalonamento que realiza a tomada de decisão em apenas um *host* é considerada como não distribuída, já uma estratégia em que essa decisão pode ser realizada por outros *hosts*, seria considerada como distribuída.

Analisando as características do ambiente experimental utilizado nesta dissertação com relação a essas duas técnicas, foi possível classificá-lo como não distribuído, visto que todas as regras e decisões sobre a distribuição das tarefas, são avaliadas de forma centralizada no *host* mestre.

3.2.4 Adaptativo *versus* Não-Adaptativo

Como já comentado, em (CASAVANT; KUHL, 1988), os autores descrevem também uma classificação de escalonamento de uma forma não hierárquica. Uma dessas divisões leva em consideração a capacidade de adaptação ao meio, ou seja, se um algoritmo é capaz de modificar suas regras dinamicamente de acordo com o comportamento atual ou anterior do sistema em que atua.

Para o presente trabalho, o termo adaptativo está relacionado à maneira com a qual o sistema trata as diferentes configurações disponíveis no ambiente distribuído. Neste caso é possível a entrada de novas máquinas escravas no ambiente durante o processamento de uma aplicação, ou até mudança de recursos entre um processamento e outro da aplicação, e espera-se que o escalonador identifique essas mudanças se adaptando ao ambiente disponível naquele momento.

3.2.5 Preemptivo *versus* Não Preemptivo

Uma outra classe de algoritmos de escalonamento, apresentada por (SHIVARATRI; KRUEGER; SINGHAL, 1992), leva em consideração a capacidade de um algoritmo permitir que uma tarefa seja suspensa de processar em uma unidade de processamento e ser transferida para ser finalizada em uma outra. Em outras palavras, a classe dos algoritmos preemptivos considera que as tarefas podem ser interrompidas e migradas para outro *host* em tempo de execução. Essa interrupção e migração de um processo em execução é considerada "cara", pelo fato do sistema precisar armazenar o estado do processo para transferi-lo juntamente com o seu estado no momento. O tipo não preemptivo envolve os algoritmos que não permitem a interrupção e migração de processos durante a execução de uma tarefa, ou seja, cada tarefa submetida para um *host*, deve iniciar e finalizar neste mesmo *host*.

Podemos classificar o tipo de escalonamento adotado neste trabalho como sendo não preemptivo, em outras palavras, um processo não pode ser interrompido e transferido entre *hosts* após ter sido iniciado em outro recurso escravo.

3.3 Balanceamento de carga

Ao utilizar o tipo de escalonamento dinâmico, é possível empregar técnicas de balanceamento de carga (*load balance*) para transferir tarefas dos *hosts* mais "carregados" para os menos (SHIRAZI; KAVI; HURSON, 1995)(DANTAS, 2005). Com o uso dessas técnicas espera-se uma distribuição das tarefas mais equilibrada entre os *hosts* disponíveis, e conseqüentemente, alcançar melhores desempenhos nas aplicações e o aproveitamento de todos os recursos disponíveis no ambiente (SHIVARATRI; KRUEGER; SINGHAL, 1992)(EAGER; LAZOWSKA; ZAHORJAN, 1986).

Segundo (ZHOU, 1988), um algoritmo de balanceamento de carga é baseado em três **políticas** inerentes:

- **Política de informação**: define quais informações serão disponibilizadas para o tomador de decisão e a forma que essas informações serão fornecidas;
- **Política de transferência**: essa política identifica as regras que definem se é adequado transferir um processo para um determinado *host* ou não. É comum essas regras levarem em consideração as cargas dos *hosts*, tamanho dos processos, entre outras métricas;
- **Política de localização**: identifica o *host* para o qual a tarefa deve ser transferida.

3.4 Co-Escalonamento (*Coscheduling*)

Para finalizar este capítulo, descrevemos nesta seção o conceito de co-escalonamento (*Coscheduling*) de processos que é utilizado ao longo desta dissertação. De acordo com (SQUILLANTE et al., 2002)(NAGAR et al., 1999), o co-escalonamento de tarefas corresponde basicamente à distribuição de tarefas para diferentes processadores em um mesmo intervalo de tempo; em outras palavras, o escalonador de tarefas é capaz de submeter ao mesmo tempo N tarefas aos processadores disponíveis. Segundo (TANENBAUM, 2001), usualmente é submetido um número de tarefas correspondente ao número de processadores disponíveis no *host*, buscando utilizar todo o poder computacional do recurso.

Alguns autores utilizam o termo *Gang Scheduling* como sinônimo de *Coscheduling* (TANENBAUM, 2001). No entanto, segundo (SODAN, 2005), existe uma diferença relevante entre estes dois conceitos. No *Gang Scheduling* o escalonamento de tarefas é realizado em um mesmo intervalo de tempo de forma síncrona, ou seja, todos os processos iniciam e finalizam em um mesmo espaço de tempo. Já o *Coscheduling* o processamento dessas tarefas é realizada de forma assíncrona, isto é, não existe uma dependência entre os processos que estão em execução, conseqüentemente não

é possível garantir que os processos finalizem em um mesmo espaço de tempo.

Para este trabalho, consideramos que ao submeter de forma simultânea um número de tarefas correspondente ao número de unidades de processamento de um *host*, sendo que estas tarefas são executadas de forma assíncrona, estamos realizando uma técnica de *Coscheduling*. Sendo assim, essas tarefas são submetidas ao mesmo tempo somente na primeira distribuição de tarefas para cada *host*, ou seja, o próximo escalonamento para este *host* dependerá do momento em que cada unidade de processamento finalizar o processamento da sua tarefa recebida anteriormente, que pode ser diferente do momento das outras unidades de processamento.

3.5 Resumo

Esse capítulo apresentou uma visão geral dos conceitos relacionados a escalonamento de processos em sistemas distribuídos. Inicialmente apresentamos a taxonomia de escalonamento proposta por Casavant, detalhando os seus principais níveis. Definimos balanceamento de carga com as suas políticas. E por fim, apresentamos o conceito de co-escalonamento de processos que é citado ao longo desta dissertação.

4 Abordagem de Co-Escalonamento Proposta

4.1 Introdução

Este capítulo relata alguns trabalhos relacionados com o assunto desta dissertação, juntamente com um experimento preliminar realizado com o *software* de computação oportunista BOINC. Na seqüência descrevemos o ATHA, que foi utilizado como sistema de computação oportunista para este trabalho. Por fim detalhamos a abordagem de escalonamento adotada para os testes empíricos realizados para esta dissertação.

4.2 Trabalhos Correlatos

Muitas pesquisas têm sido focadas no uso eficiente de configurações multi-core e de multi-processadores. Em (CVETANOVIC, 1987) é apresentada uma investigação dos efeitos da decomposição, alocação e granularidade de tarefas em arquiteturas de multi-processadores com memória compartilhada. Alguns aspectos sobre o uso de aplicações MPI em ambientes de configurações multi-core e de multi-processadores fracamente acoplados são apontados em (POURREZA; GRAHAM, 2007). O trabalho (EL-MOURSY et al., 2006) explora algumas políticas de co-escalonamento de tarefas em máquinas multiprocessadas.

Quanto aos sistemas de computação oportunista, na literatura é possível encon-

trar alguns projetos que buscam utilizar os ciclos ociosos dos *hosts* disponíveis em uma rede, para contribuir no processamento de aplicações de granularidade grossa (*coarse grain*) e que necessitem de um uso intenso de processadores (*CPU-Bound*). Alguns exemplos clássicos de sistemas oportunistas são: Charlotte (BARATLOO et al., 1996), POPCORN (NISAN et al., 1998), SETI@Home (SETI@HOME, 2008) e BOINC (BOINC, 2008). Conforme testes realizados, este último já é dotado de uma técnica de escalonamento que busca aproveitar de uma maneira mais eficiente as configurações multi-core e de multi-processadores disponíveis na rede como *hosts* escravos. Mais detalhes sobre esse experimento preliminar podem ser verificados na seção seguinte.

4.2.1 BOINC

O sistema de processamento oportunista BOINC (*Berkeley Open Infrastructure for Network Computing*) (BOINC, 2008) utiliza o paradigma mestre-escravo conforme o seu predecessor SETI@Home (SETI@HOME, 2008). Dessa forma, o módulo mestre é responsável por dividir os projetos em tarefas menores e distribuí-las entre os recursos voluntários disponíveis na Internet e que irão contribuir no processamento desses projetos. No BOINC é possível um usuário se cadastrar para participar de um ou mais projetos simultaneamente (e.g. SETI@Home, Proteins@Home e Predictor@Home).

Com o objetivo de observar como o BOINC lida com o problema tratado nesta dissertação (i.e. investigar o comportamento desses sistemas diante da disponibilidade de *hosts* escravos com múltiplas unidades de processamento), foram realizados alguns experimentos preliminares com este sistema. O BOINC foi escolhido, para este teste inicial, por ser um projeto de pesquisa aberto e possuir várias publicações e documentações relacionadas ao projeto que podem ser encontradas em (BOINC, 2008). O ambiente utilizado neste teste foi formado por uma máquina escrava Pentium(R) Dual-Core, com 1 GByte de RAM, rodando o sistema operacional Windows

XP.

Dentre os projetos disponíveis no BOINC, para este experimento foi utilizado apenas o projeto SETI@Home. A figura 7 ilustra a tela principal do BOINC executando duas tarefas do SETI@Home na máquina escrava citada anteriormente. Essa figura indica que o sistema BOINC foi capaz de identificar que o recurso escravo era dotado de duas unidades de processamento, e por isso realizou a submissão de duas tarefas para serem executadas de forma paralela neste recurso.

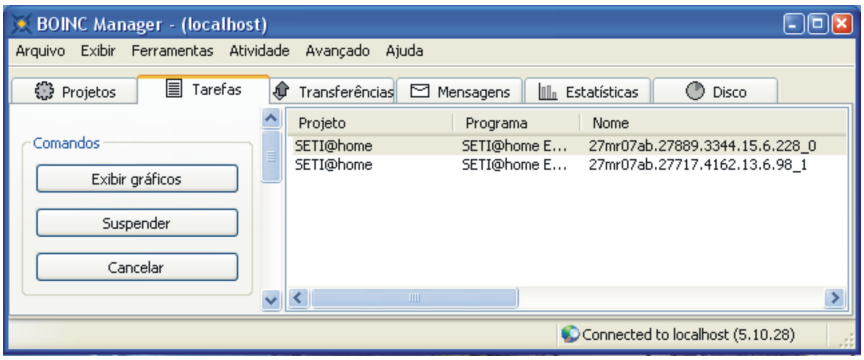

Figura 7: Tela das Tarefas do BOINC.

Com o auxílio do gerenciador de tarefas do Windows XP disponível no *host* escravo, foi possível observar o comportamento desta máquina quando estava processando as tarefas submetidas pelo sistema BOINC. Nas figuras 8 e 9 é possível analisar os efeitos da submissão de duas tarefas para uma máquina com duas unidades de processamento. Com a figura 8 é possível verificar que as duas unidades de processamento da máquina escrava estão sendo utilizadas completamente, em outras palavras, o sistema BOINC utilizou toda a capacidade ociosa do recurso escravo para o processamento da aplicação submetida. Na figura 9 é possível verificar que cada processo SETI@Home iniciado pelo BOINC está utilizando 50% da capacidade total de processamento do recurso, ou seja, cada um está utilizando, por completo, uma unidade de processamento da máquina, totalizando 100% de utilização do poder computacional total do mesmo.

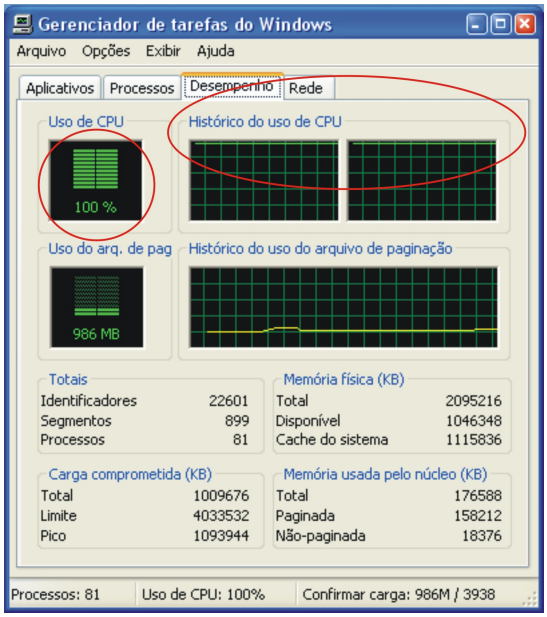

Figura 8: Estado Processadores Executando o BOINC.

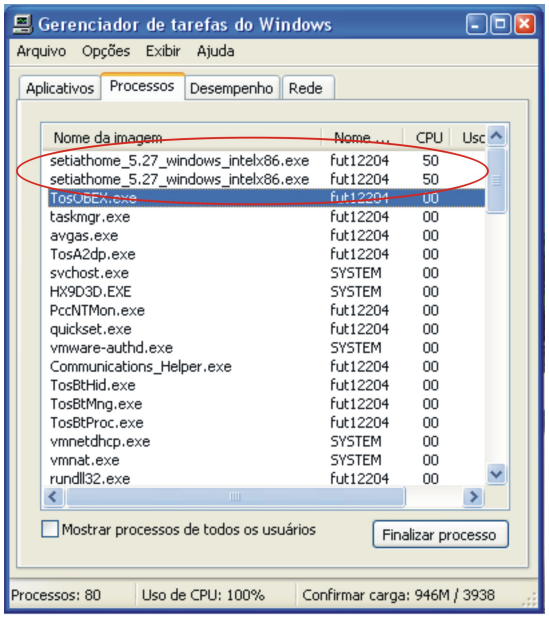

Figura 9: Processos Iniciados Executando o BOINC.

Com esse experimento, foi possível verificar que o sistema BOINC já possui um mecanismo que submete um número de tarefas baseado na quantidade de unidades de processamento disponíveis em cada recurso escravo, buscando utilizar desta forma toda a capacidade de processamento disponível nos *hosts*. Sendo assim, o BOINC é capaz de se adaptar às diversas configurações de máquinas, quando considerado a heterogeneidade do número de unidades de processamento das mesmas. Na literatura (e.g. (ANDERSON, 2007), (KONDO; ANDERSON; VII, 2007)) é possível encontrar pesquisas sobre algumas técnicas de escalonamento utilizadas pelo BOINC. Na próxima seção, apresentamos um sistema de computação oportunista denominado ATHA, que foi utilizado como objeto de estudo para os demais experimentos desta dissertação.

4.3 Sistema de Computação Oportunista ATHA

O sistema de computação oportunista ATHA (HOSKEN, 2003), foi desenvolvido pelo nosso grupo de pesquisa (LAPESD, 2008), com o objetivo de reunir a capacidade ociosa dos recursos disponíveis em uma rede (Intranet ou Internet) para a execução de aplicações complexas.

O ATHA possui uma abordagem similar ao BOINC, "roubando" os ciclos ociosos de máquinas em uma rede. O sistema foi desenvolvido na linguagem de programação Java (JAVA, 2008), tendo em vista a diversidade dos recursos encontrados em um grande sistema distribuído. O ATHA utiliza o paradigma mestre-escravo para executar as tarefas, ou seja, existe um módulo mestre, onde uma classe Java é submetida para ser executada de modo paralelo e distribuído. Nesse momento, também é necessário informar, como parâmetro, o intervalo de dados que será processado pela aplicação submetida. Este sistema possui também o módulo escravo, o qual é basicamente responsável por receber a tarefa, executá-la remotamente e retornar para o módulo mestre o resultado parcial obtido.

Uma vez submetida uma classe Java no módulo mestre do ATHA, esse será o responsável por dividir os dados recebidos como argumentos em pedaços menores e iguais, que serão submetidos para os *hosts* escravos disponíveis em uma rede. Neste sentido, é importante relatar que na versão original do ATHA, cada *host* recebe e executa **um intervalo** de dados, ou seja, uma parte do problema por vez. Quando este conclui a sua tarefa, ele retorna o resultado parcial para o mestre, que é responsável por submeter um novo intervalo de dados do problema para ser processado remotamente, isso enquanto o resultado final da aplicação não for alcançado.

A figura 10 ilustra um diagrama de seqüência do sistema ATHA, e que iremos explicar a seguir:

1. Inicialmente uma aplicação, em nosso caso uma classe Java do algoritmo RC5 (RIVEST, 1996), é submetida em uma interface gráfica com o usuário existente no módulo mestre do ATHA. Depois de submetida essa classe Java, o módulo mestre divide o intervalo de dados recebido como parâmetro em intervalos menores e iguais que serão enfileirados em um gerenciador de tarefas existente no módulo mestre;

2. O passo seguinte do diagrama de seqüência pode ocorrer em qualquer momento durante o processamento da aplicação. Em outras palavras, um *host* que venha a contribuir no processamento da aplicação principal poderá entrar no ambiente de processamento oportunista a qualquer instante, isso porque ao entrar, este automaticamente irá solicitar tarefas para o módulo mestre, que por sua vez, existindo tarefas pendentes na fila irá submetê-las para o novo voluntário;

3. O terceiro passo corresponde à solicitação de informações relacionadas a cada máquina escrava do ambiente oportunista. Neste momento existe também o passo 3.1, no qual o módulo escravo executa o *benchmark* Linpack serial (LINPACK, 2008) para obter a performance em MFlops deste recurso escravo. Por fim, o módulo escravo envia este resultado e mais algumas informações relacionadas ao *host* escravo (e.g. sistema operacional, arquitetura e quantidade de memória RAM), para o *host* mestre;

4. Após esse retorno, o módulo mestre submete **uma** tarefa para ser executada neste *host* escravo, que após executá-la remotamente, retorna o resultado parcial obtido para o módulo mestre;

5. Na seqüência, o módulo mestre avalia o resultado recebido do *host* escravo, e se este for o procurado pela a aplicação principal, o passo seguinte será executado, senão retorna ao passo 3;

6. Uma vez verificado que o resultado procurado da aplicação foi alcançado, o módulo mestre irá apresentar este retorno na interface com o usuário do ATHA.

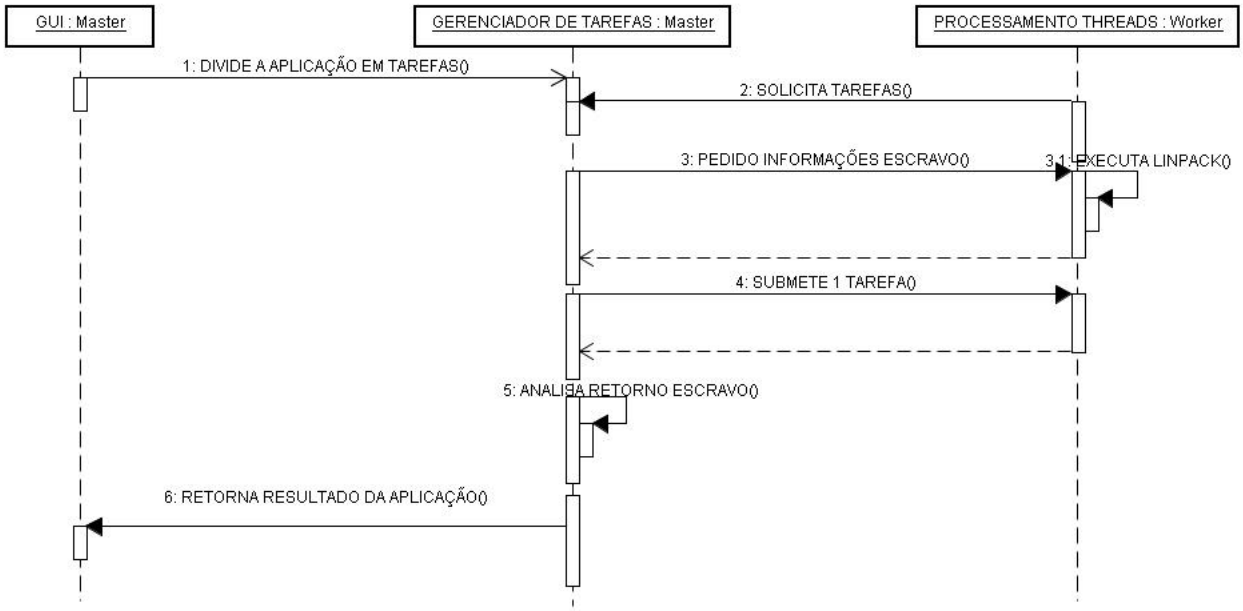

Figura 10: Diagrama Seqüência ATHA.

4.4 Modelo da Proposta

Tendo em mente a seqüência dos passos realizados pelos módulos mestre e escravo na versão original do ATHA, propusemos um módulo de escalonamento agregado ao gerenciador das tarefas do módulo mestre. Este módulo utiliza como base as informações recebidas dos *hosts* escravos tornando-se capaz de "raciocinar" e com isso enviar mais tarefas para serem executadas nas máquinas escravas dotadas de múltiplas unidades de processamento.

Apesar de ter sido utilizado neste trabalho de pesquisa apenas o número de unidades de processamento como política de escalonamento, podemos estender a idéia deste modelo utilizando outros índices (e.g. tempo de execução do *benchmark* Linpack serial ou o total de chaves testadas por segundo) para enviar mais tarefas para uns *hosts* e menos para outros, aperfeiçoando ainda mais o escalonamento e

balanceamento de carga realizado pelo sistema de computação oportunista ATHA.

A figura 11 ilustra um fluxograma do funcionamento do modelo proposto nesta dissertação, com ênfase nos passos realizados pelo módulo mestre. Como podemos verificar, após a submissão da aplicação na interface gráfica com o usuário, ocorre uma divisão do problema em tarefas menores e iguais, que farão parte de uma fila no gerenciador de tarefas. Podemos verificar na figura expandida, que este passo é responsável por avaliar a quantidade de unidades de processamento disponíveis em cada *host* escravo, juntamente com a quantidade de tarefas pendentes na fila, e com isso decidir quantas tarefas deverão ser submetidas para este *host*, de acordo com informações previamente coletadas do mesmo.

Uma vez decidido a quantidade de tarefas, o *host* escravo recebe as tarefas, que são executadas de forma paralela e independente. Ao finalizar o processamento, o módulo escravo envia cada resultado parcial obtido, para o módulo mestre. Este, por sua vez, avalia o resultado e se for o procurado, a aplicação é encerrada, senão o gerenciador de tarefas avalia novamente as informações do *host* escravo *versus* quantidade de tarefas pendentes na fila e mais uma vez decide quantas tarefas devem ser processadas neste *host*, isso até que o resultado final da aplicação seja alcançado.

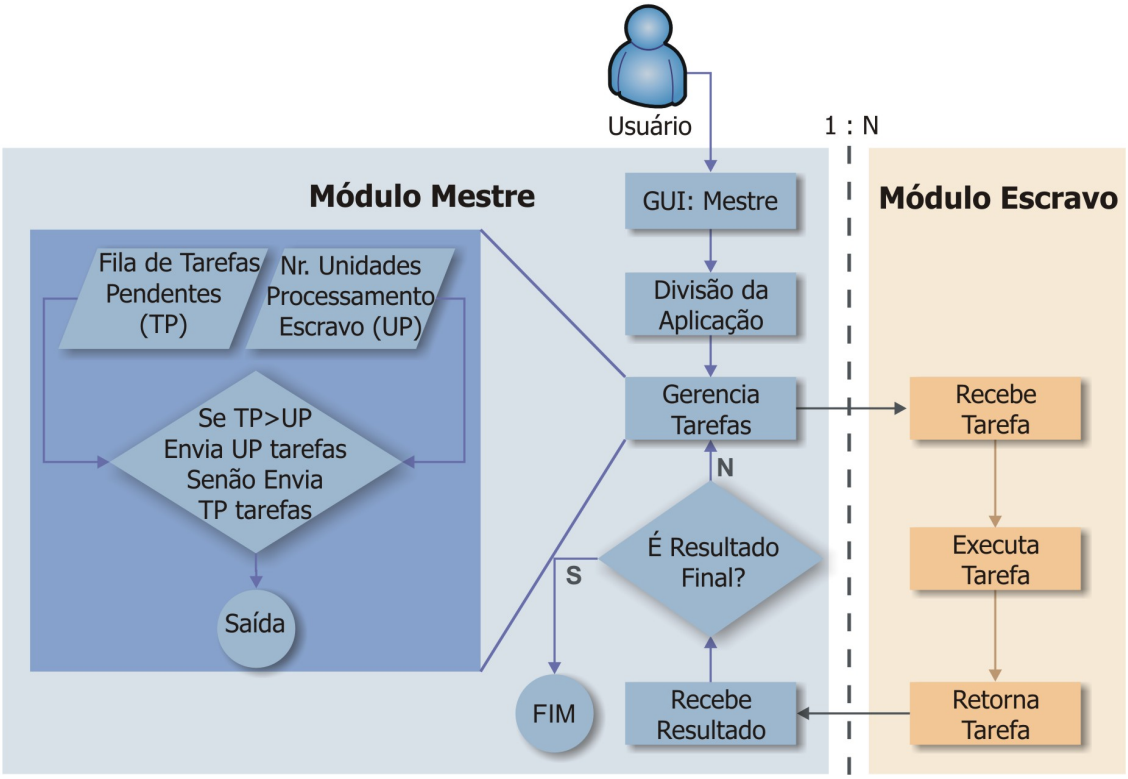

Figura 11: Fluxograma do Modelo Proposto Módulo Mestre.

Para analisarmos a proposta no ponto de vista do módulo escravo, verificamos a figura 12 que destaca os passos executados por este módulo seguindo a nossa proposta. Um primeiro aspecto é que o módulo escravo pode ser iniciado de forma independente do módulo mestre. Nesse caso, este módulo fica basicamente solicitando tarefas ao módulo mestre. Quando a aplicação principal é submetida através da interface com o usuário do módulo mestre, este passará a enviar tarefas para os voluntários que estiverem executando o módulo escravo do sistema de computação oportunista ATHA. Assim, com os dois módulos iniciados, o processamento remoto começa a ser realizado de fato.

Em um primeiro momento, o módulo escravo avalia a disponibilidade de unidades de processamento no *host*, e com essa informação solicita uma ou mais tarefas para o módulo mestre. Este por sua vez, avalia a configuração deste *host* e a partir dessa análise, envia uma ou mais tarefas para serem processadas de forma inde-

pendente no escravo. Na seqüência o módulo escravo inicia o processamento dessas tarefas de forma independente e após finalizado o processamento de cada uma delas, retorna o resultado parcial obtido para o módulo mestre. Neste momento, o módulo escravo ao identificar que existe novamente uma unidade de processamento livre, solicita uma nova tarefa ao módulo mestre. Vale observar que, para o módulo escravo do ATHA, não existe um estado final no fluxograma. Isso indica que, ao iniciar um módulo escravo em um *host*, este pode ficar rodando este módulo enquanto o mesmo estiver disponível na rede, contribuindo dessa forma para o processamento de outras aplicações que possam ser submetidas no módulo mestre.

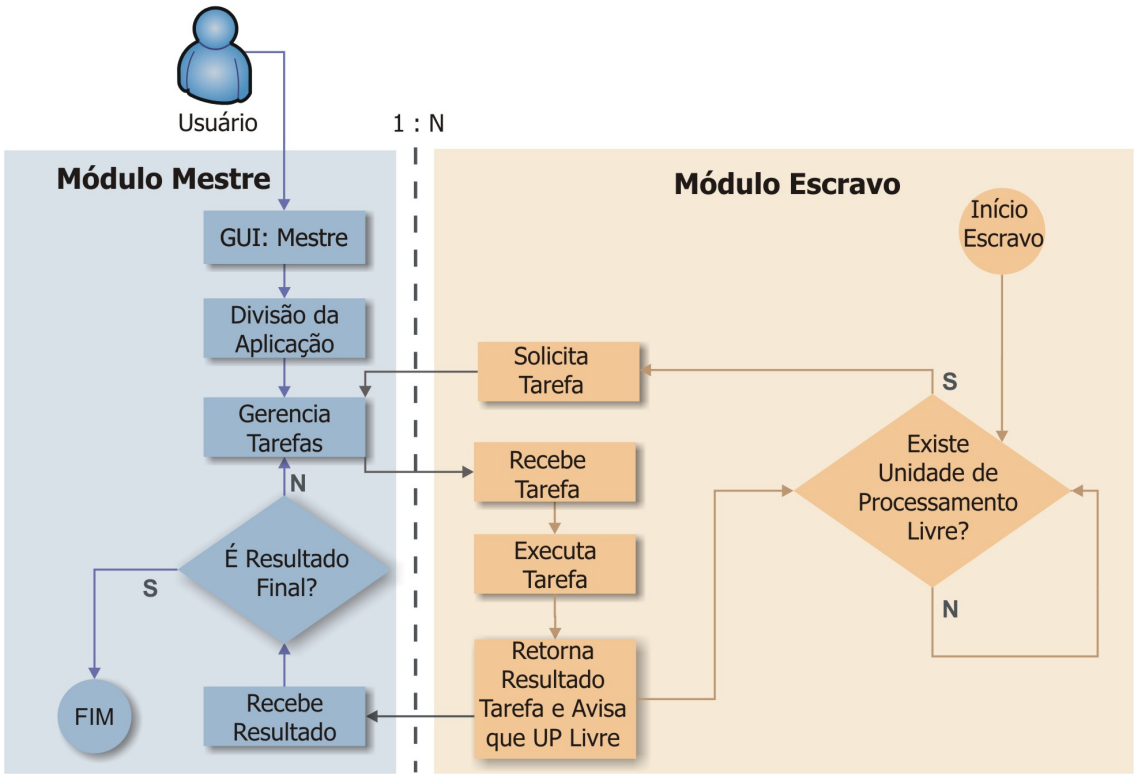

Figura 12: Fluxograma do Modelo Proposto Módulo Escravo.

Na próxima seção descrevemos com detalhes todos os passos executados para a implementação desse modelo no sistema de computação oportunista ATHA, o qual foi utilizado como objeto de estudo para esta dissertação.

4.5 Aspectos da Implementação da Proposta

O software ATHA foi escolhido como o sistema de processamento oportunista para os experimentos desta dissertação pelos seguintes motivos: foi desenvolvido pelo nosso grupo de pesquisa (LAPESD, 2008); possui 100% do seu código fonte desenvolvido na linguagem de programação Java, o que é uma característica relevante quando consideramos a diversidade de arquiteturas de computadores disponíveis em uma rede como a Internet; e finalmente por ter o código fonte aberto, o que é essencial para estudos futuros e novas melhorias no projeto.

Deste ponto em diante desta dissertação, para facilitar as comparações realizadas entre o ATHA na versão original e a sua versão melhorada com a estratégia de co-escalonamento proposta na seção anterior, iremos referenciar esta última através da sigla ATHA-RSAE (ATHA - *Resource Scheduler ATHA Environment*).

Considerando a versão original do ATHA, foi necessário modificar os módulos mestre e escravo deste sistema. Isso porque quando um recurso escravo dotado de múltiplas unidades de processamento é identificado no ambiente, o módulo mestre deve ser capaz de enviar mais tarefas para serem processadas de forma independente neste recurso, o que não é possível na versão original do ATHA, como comentado na seção 4.3. Com essa melhoria, essas novas tarefas passam a ser processadas nos *hosts* escravos através de múltiplas *threads* da aplicação principal, o que em nosso caso foi utilizado algoritmo RC5 (RIVEST, 1996). O objetivo com isso, é que toda a capacidade de processamento disponível nestes recursos seja usada, independentemente das configurações encontradas no ambiente distribuído. Em outras palavras, a estratégia considera que todas as máquinas do ambiente de computação oportunista podem ter múltiplas unidades de processamento e que a aplicação submetida não foi desenvolvida com a preocupação de "tirar vantagem" dessas arquiteturas multiprocessadas.

Com a figura 13, analisamos o diagrama de seqüência do ATHA-RSAE. Nele verificamos que até o passo três o ATHA e o ATHA-RSAE realizam as mesmas etapas. A diferença surge a partir do passo quatro, que no ATHA-RSAE ocorre uma análise das informações recebidas do escravo juntamente com uma verificação das tarefas pendentes na fila. Dessa forma, se ainda restarem tarefas pendentes, o módulo mestre enviará o número de tarefas correspondente ao número de unidades de processamento disponível no *host* escravo de forma simultânea, caracterizando assim o co-escalonamento de tarefas. Após a submissão dessas tarefas, os passos seguintes do diagrama de seqüência continuam de acordo com o que já foi explicado, na seção 4.3, para o sistema ATHA na sua versão original.

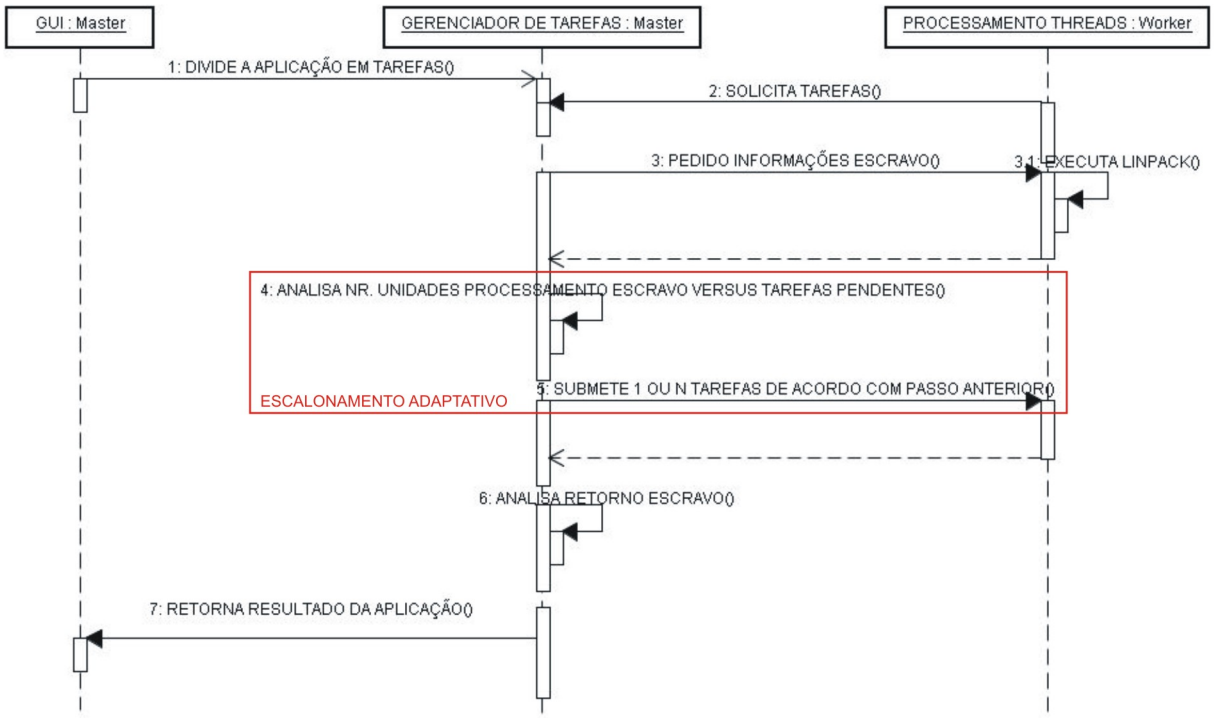

Figura 13: Diagrama Seqüência ATHA-RSAE.

Dessa forma, tornamos o sistema de computação oportunista ATHA capaz de analisar a configuração dos *hosts* disponíveis em um ambiente distribuído, e baseando-se na quantidade de unidades de processamento de cada *host* escravo, o

módulo mestre passa a submeter mais tarefas que serão executadas remotamente nessas máquinas. O módulo escravo, por sua vez, foi melhorado para permitir o processamento, de uma maneira paralela e independente, várias tarefas do problema principal, através da inicialização de múltiplas *threads* simultâneas. Para finalizar a apresentação da nossa proposta, descrevemos a seguir uma tabela comparativa com algumas características discutidas até o momento sobre o BOINC, ATHA e o ATHA-RSAE.

Tabela 2: Tabela Comparativa entre BOINC, ATHA e ATHA-RSAE

Características	BOINC	ATHA	ATHA-RSAE
Nr. Projetos Simultâneos	Vários	1 por vez	1 por vez
Escalonamento (Nr. UPs)	Sim	Não	Sim
Linguagem de Programação	C, C++ ou Fortran, dependendo da versão	100% Java	100% Java

4.6 Resumo

Neste capítulo foram apresentados os trabalhos correlatos, juntamente com uma experiência preliminar ao utilizar o sistema de computação oportunista BOINC. Detalhamos ainda os módulos do sistema oportunista ATHA, que foi utilizado como objeto de estudo neste trabalho de pesquisa. Por fim, descrevemos o modelo proposto e relatamos alguns aspectos sobre a implementação do modelo no sistema ATHA.

5 Resultados Experimentais

5.1 Introdução

Neste capítulo é descrito o ambiente computacional utilizado nos experimentos. Em seguida as métricas avaliadas são detalhadas. Por fim, são apresentados os resultados empíricos obtidos nos experimentos com o sistema de computação oportunista ATHA e com a sua versão aperfeiçoada, o ATHA-RSAE.

5.2 Ambiente Experimental

Nos experimentos realizados para esta dissertação, utilizamos o algoritmo RC5 (RIVEST, 1996) como aplicação de demonstração para ser executada no sistema de processamento oportunista ATHA. Inicialmente, este algoritmo foi escolhido por ser de granularidade grossa e *CPU-Bound*. Uma outra razão relevante para a sua escolha se deu ao fato de que esse algoritmo foi extremamente testado no ATHA durante o desenvolvimento da sua versão original, conforme (HOSKEN, 2003).

Com relação aos equipamentos utilizados em nossos testes empíricos, esses são parte do ambiente real de uma companhia brasileira de desenvolvimento de *software*. Nas máquinas dessa organização usualmente são executadas tarefas como: compilação de programas; edição de textos e planilhas eletrônicas; processamento de aplicações CRM e ERP; e-mail pessoal e VOIP.

Desta forma, com o propósito de verificar a importância de uma estratégia de

escalonamento de tarefas em um ambiente de computação oportunista dotado de recursos multi-core e de multi-processadores, foi utilizado o sistema ATHA na sua versão original, sendo que nessas condições este não foi capaz de reunir toda a capacidade ociosa disponível para o processamento da aplicação teste. A figura 14 ilustra a tela do módulo mestre do ATHA, quando este submete apenas uma tarefa para ser executada em uma máquina escrava de IP 10.15.10.1. É importante citar que este IP refere-se a uma máquina Intel(R) Xeon 4 CPUs 3,2 GHz com 6 GByte de memória RAM, ou seja, uma máquina dotada de quatro processadores.

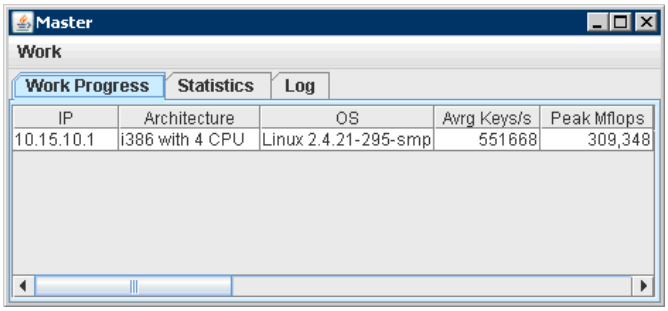

Figura 14: ATHA para Escravo com Quatro Processadores.

A ferramenta Top (TOP, 2008) do sistema operacional Unix foi utilizada nesse experimento, como forma de monitorar a utilização das CPUs disponíveis na máquina escrava. Desse monitoramento é possível verificar, com a figura 15, que apenas um processo Java foi iniciado, apesar das quatro unidades de processamento disponíveis na máquina escrava. Além disso, essa figura mostra que o processo Java iniciado utiliza 97,3% de uma unidade de processamento, deixando as CPUs 0, 1 e 3 com uma capacidade de processamento ociosa (coluna *idle*) de 57,8%, 46,6% e 57,5%, respectivamente.

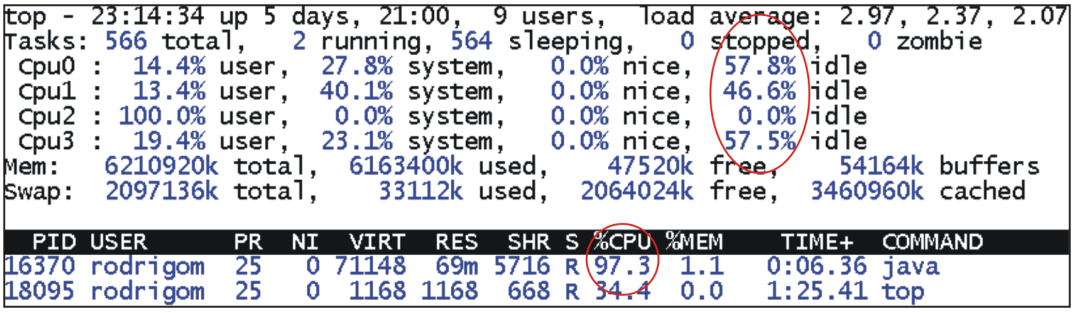

Figura 15: Top em Escravo com Quatro Processadores - ATHA.

Na seqüência, realizamos o mesmo experimento usando a versão do ATHA-RSAE. Isso com o objetivo de acompanhar o comportamento do ambiente ao iniciar múltiplas *threads*, nesse recurso escravo dotado de quatro unidades de processamento.

A figura 16 ilustra que dessa vez quatro tarefas foram iniciadas na máquina escrava de IP 10.15.10.1. Ademais, na figura 17 observa-se quatro processos Java em execução. Ainda na figura 17 verificamos que esses processos estão usando uma fatia relevante da capacidade processamento total dessa máquina escrava. Com isso, a capacidade de processamento ociosidade das quatro unidades de processamento ficou em 0%. Demonstrando assim que, quando utilizada uma estratégia de co-escalonamento em um ambiente com uma configuração multiprocessada, foi possível alcançar uma maior eficiência no uso deste recurso.

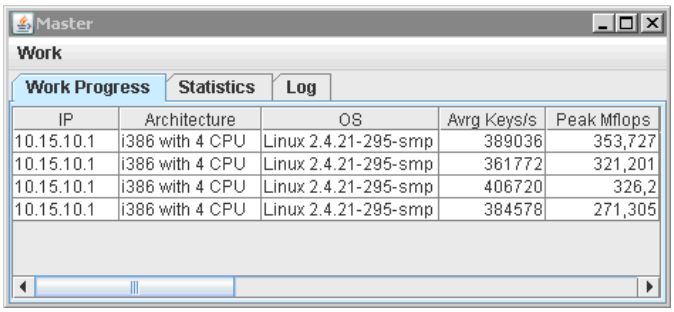

Figura 16: ATHA-RSAE para Escravo com Quatro Processadores.

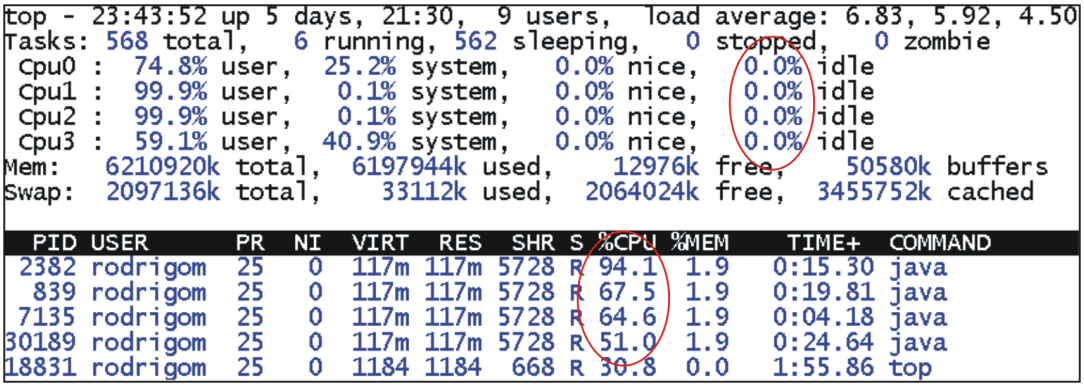

Figura 17: Top em Escravo com Quatro Processadores - ATHA-RSAE.

5.2.1 Métricas Utilizadas

Para os gráficos que serão apresentados a seguir, foram utilizadas as informações dos relatórios gerados pelo sistema de computação oportunista ATHA. A primeira medida considerada, trata-se da performance (P), em MFlops, de cada recurso escravo utilizado no ambiente. Essa informação é obtida utilizando o *benchmark* Linpack serial (LINPACK, 2008) antes de iniciar cada processamento em um *host* escravo. A segunda métrica utilizada foi o número de chaves testadas por segundo (CT) de cada *host* escravo. Entretanto, as informações para esta medida são coletadas no final de cada processamento remoto.

Vale destacar que pelo fato dessas métricas serem calculadas com informações coletadas em momentos distintos, observa-se uma diferença nos tempos totais de processamento da aplicação nos gráficos de performance e de chaves testadas por segundo. No entanto, este fato não prejudicou as análises visto que apenas comparamos a mesma métrica para o ATHA e o ATHA-RSAE. Durante o desenvolvimento do ATHA-RSAE, foi necessário adaptar os relatórios do ATHA para que ser possível coletar os dados necessários para estas métricas em nível de unidade de processamento.

Foram gerados também gráficos de performance global (PG) e de chaves testadas

por segundo global (CTG) do ambiente, que basicamente representam a somatória da P e CT de todas as unidade de processamento, respectivamente. Utilizando as informações de PG e CTG do ambiente, realizamos também estudos de caso com um número crescente de máquinas, tendo em vista analisar a escalabilidade do ambiente.

Finalizando as métricas adotadas temos o *Speedup* (S), que freqüentemente é utilizado na literatura para avaliação de algoritmos paralelos (QUINN, 1994) (JORDAN; ALAGHBAND, 2002). O *Speedup*, na sua essência, corresponde ao tempo de processamento de uma aplicação seqüencial dividido pelo tempo dessa aplicação paralelizada. No entanto, como o sistema ATHA, na sua versão original, já realiza um processamento paralelo, adaptamos essa medida para representar o tempo total de processamento utilizando a versão do ATHA dividido pelo tempo total do ATHA-RSAE. As fórmulas abaixo apresentam de forma resumida todas as métricas consideradas para analisar o sistema distribuído utilizado como objeto de estudo.

$$
\begin{aligned}
P &= Performance\ Individual \\
CT &= Chaves\ Testadas\ por\ Segundo\ Individual \\
PG &= \sum P \\
CTG &= \sum CT \\
SPG &= \frac{\sum PG\ \ ATHA}{\sum PG\ \ ATHA-RSAE} \\
SCTG &= \frac{\sum CT\ \ ATHA}{\sum CT\ \ ATHA-RSAE}
\end{aligned}
$$

5.2.2 Resultados Empíricos

Depois da análise preliminar realizada na seção 5.2, podemos inferir que o sistema ATHA, quando executado em um ambiente dotado de recursos multi-core e de multi-processadores, não utiliza de forma eficiente toda a capacidade de processamento disponível. Entretanto, novos experimentos foram necessários para analisar com mais detalhes as vantagens ao utilizar a abordagem de escalonamento proposta.

A tabela 3 mostra alguns detalhes das máquinas utilizadas nos testes apresentados nas seções 5.2.3 e 5.2.4. Além disso, estes *hosts* estão interconectados através de *Switches* Gigabit Ethernet da 3Com, modelo SuperStack 3.

Tabela 3: Ambiente Experimental Primeira Rodada de Testes

Máquina	Arquitetura	RAM	S.O.
Mestre	Pentium 4 HT, CPU 2.80GHz	512MBytes	Windows XP
(A)	Intel Xeon 4 CPUs 3.2GHz	6GBytes	SUSE Linux 9
(B)	Pentium 4 HT, CPU 2.80GHz	512MBytes	Windows XP
(C)	Celeron CPU 2.80 GHz	512MBytes	Windows XP
(D)	Pentium DualCore CPU 1.6Ghz	1GBytes	Windows XP

Tendo em vista um maior controle dos testes e um melhor acompanhamento dos resultados, foi decidido nesse primeiro momento utilizar apenas quatro máquinas como *hosts* escravos. Uma outra razão para esta decisão, se deu pelo fato de que com menos máquinas foi possível analisar, com mais detalhes, o comportamento de cada unidade de processamento disponível no ambiente diante da estratégia de co-escalonamento proposta.

5.2.3 Performance

Utilizando o ambiente descrito na seção anterior, geramos a figura 18 que apresenta a métrica P (Performance) de cada *host* escravo durante o tempo total de processamento da aplicação. Com essa figura podemos verificar que o ambiente experimental utilizado é formado por duas máquinas com mais capacidade de processamento (A e D), uma máquina intermediária (B) e uma mais limitada (C). Outro ponto de destaque neste gráfico é a grande oscilação da linha que representa a máquina (A). Isso ocorre porque essa máquina trata-se de um servidor de aplicações da companhia que executa uma grande quantidade tarefas concomitantes.

Por outro lado, quando observamos as linhas que representam os *hosts* (B, C e D), estas se mostram mais estáveis, por se tratarem de computadores pessoais usados na maior parte do tempo por apenas um único usuário.

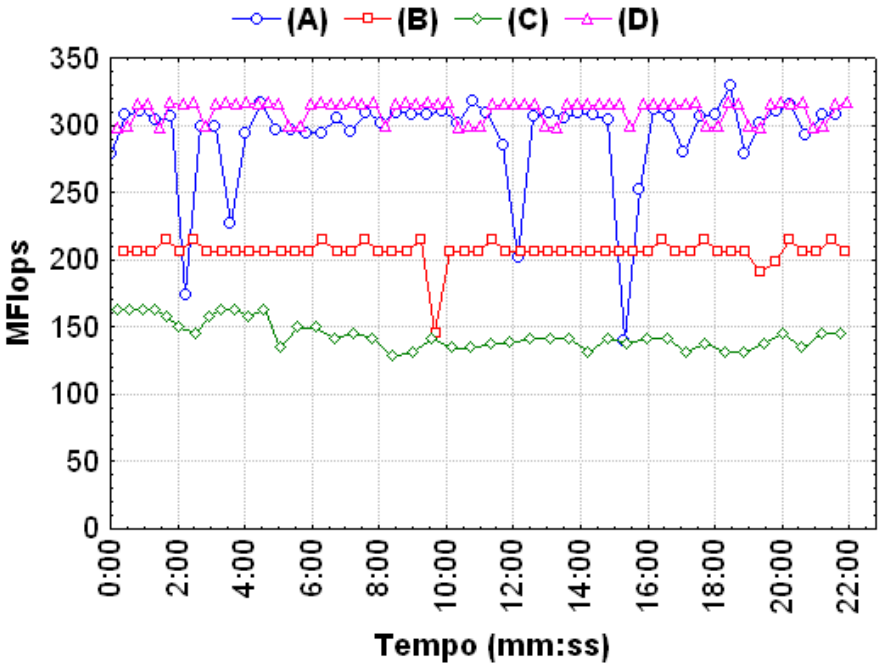

Figura 18: Performance Individual ATHA.

Complementando a análise realizada com a figura 18, foi gerada a figura 19 que representa a métrica PG (Performance Global) do ambiente. Pode-se observar que no melhor momento, a PG atinge 1.000 MFlops e aos 22 minutos, momento em que o resultado do algoritmo foi alcançado, a linha do gráfico é finalizada.

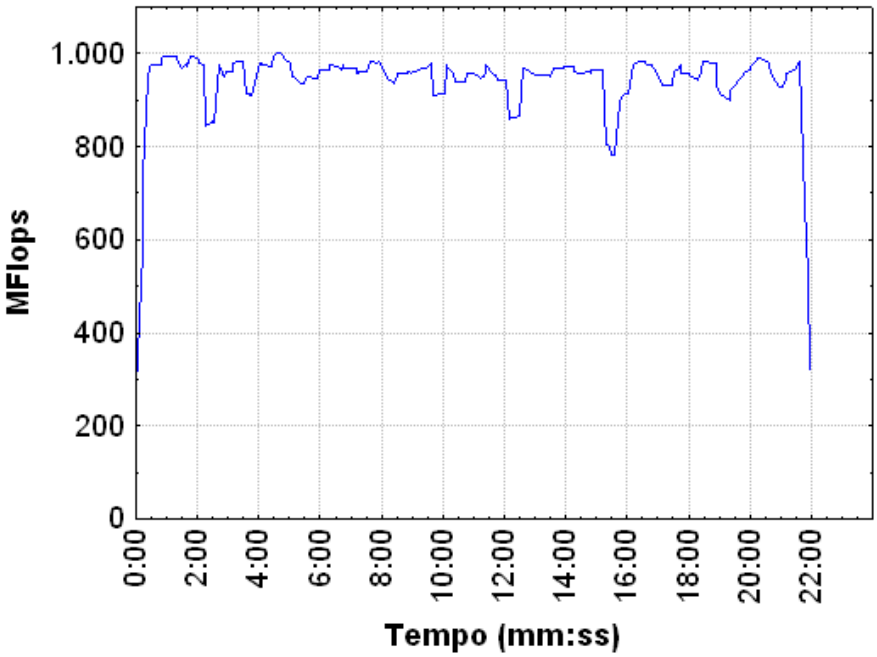

Figura 19: Performance Global ATHA.

Após analisar a P e PG com a versão original do ATHA, foram realizados os mesmos experimentos, entretanto, utilizando o ATHA-RSAE. Dessa forma, foi gerada a figura 20 que apresenta a P (Performance) de cada unidade de processamento disponível nos recursos, sendo estas unidades identificadas pelo número que segue a letra que representa cada máquina na tabela 3. Por exemplo, o identificador (A0) corresponde à unidade de processamento 0 da máquina (A).

Dessa forma, com a figura 20, verificamos que a oscilação das linhas que representam as unidades de processamento da máquina (A) é mais expressiva, quando comparamos com a linha que a representa na figura 18. Já para as demais máquinas, as linhas se mostram semelhantes nas duas figuras. Entendemos que isso ocorre, pois a máquina (A), como já citado, trata-se de um servidor de aplicações constantemente requisitado na companhia. Dessa forma, quando o ATHA submete apenas uma tarefa, o sistema operacional dessa máquina é capaz de escalonar esta tarefa para ser executada pela unidade de processamento menos “carregada”. No entanto,

quando são submetidas quatro tarefas, quando utilizado o ATHA-RSAE, o sistema operacional distribui essas tarefas igualmente entre as unidades de processamento, o que gera uma notável concorrência pelas unidades de processamento com as outras tarefas dessa máquina.

Ademais, com a figura 20, é possível verificar que a linha que representa a unidade de processamento da máquina (C), mostra-se superior às linhas da máquina (B). Já no gráfico 18 a linha da máquina (C) apresenta-se inferior à da (B). Acreditamos que isso se deve ao fato de que a máquina (B) não possui, *de facto*, múltiplas unidades de processamento, e sim utiliza a tecnologia *Hyper-Threading Technology*. Essa tecnologia, segundo (HTT, 2008), provê um paralelismo em nível de *threads* buscando tornar mais eficiente o processamento de tarefas concorrentes. Todavia, ainda que máquinas dotadas da tecnologia HTT simule a existência de duas unidades de processamento, as duas tarefas submetidas pelo ATHA-RSAE disputam a utilização de uma unidade de processamento apenas, diminuindo consideravelmente a performance das linhas que representam esta máquina. Essa diferença não é observada para as máquinas (A) e (D), que possuem realmente 2 *cores* e 4 processadores, respectivamente, e nem para a máquina (C) que continuou recebendo apenas uma tarefa com o uso do ATHA-RSAE.

Ainda com a figura 20, observamos que o tempo total de processamento da aplicação diminuiu para 12 minutos e 30 segundos quando usamos o ATHA-RSAE, enquanto que quando utilizado ATHA na sua versão original, esse tempo ficou em 22 minutos como observado nas figuras 18 e 19.

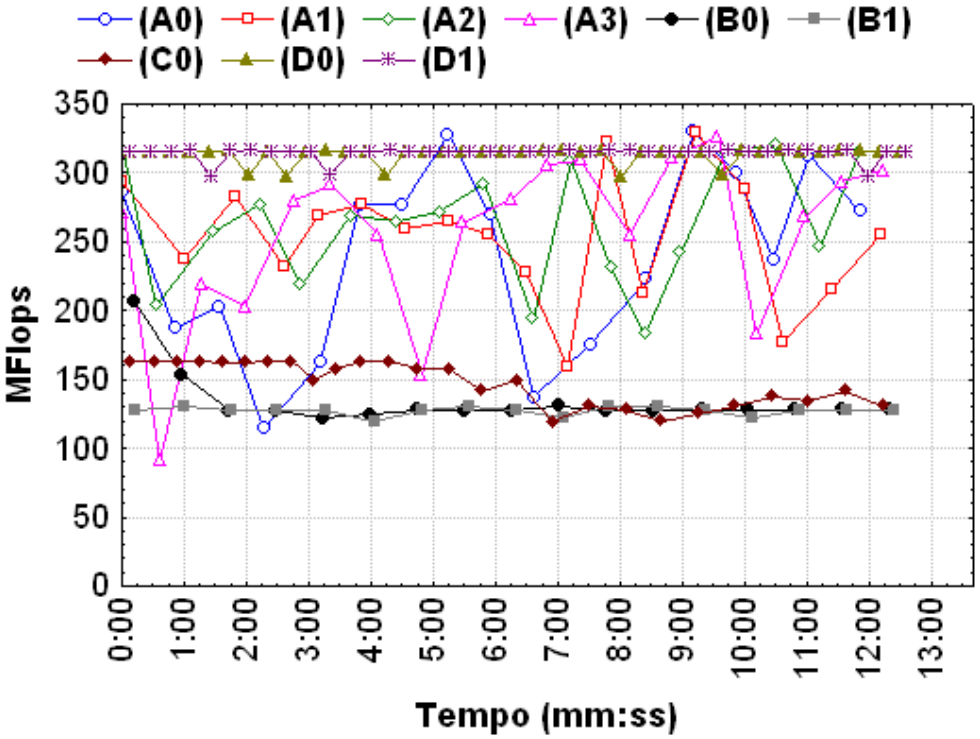

Figura 20: Performance Individual ATHA-RSAE.

Completando essa análise de performance, foi gerada a figura 21 que apresenta a métrica PG (Performance Global) deste ambiente, quando utilizado o ATHA-RSAE. É possível observar que a PG se mantém próxima de 2.000 MFlops no decorrer do tempo, enquanto que na figura 19 (ATHA) a PG atinge no máximo 1.000 MFlops em alguns momentos.

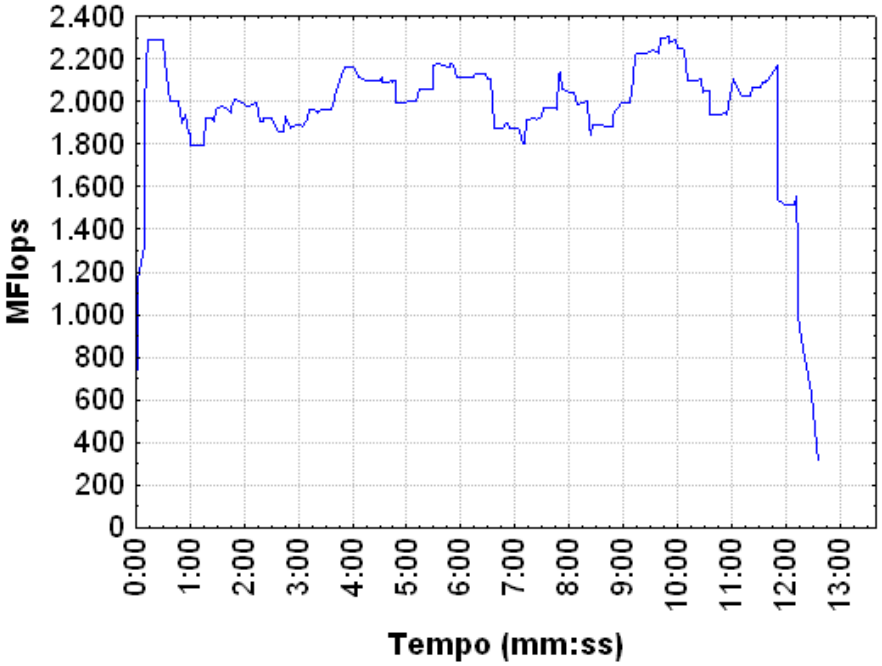

Figura 21: Performance Global ATHA-RSAE.

5.2.4 Capacidade de Processamento

Tendo em vista avaliar os ganhos obtidos para a aplicação escolhida para os experimentos com o uso do ATHA-RSAE, foram geradas as figuras 22 e 23, que apresentam a capacidade de chaves criptográficas testadas por segundo no ambiente, utilizando o ATHA e o ATHA-RSAE, respectivamente.

No gráfico da figura 22 podemos observar que no inicio do processamento, a CTG (Chaves Testadas por segundo Global) chega a 3.000.000 chaves. Em seguida, existe uma suave redução nessa métrica, estabilizando-se em aproximadamente 2.750.000. Também é possível observar que aos 21 minutos a linha sofre uma queda, onde acreditamos que o resultado da aplicação foi alcançado e aos 22 minutos a linha é finalizada, indicando que todas as tarefas ainda em processamento nos *hosts* escravos foram finalizadas.

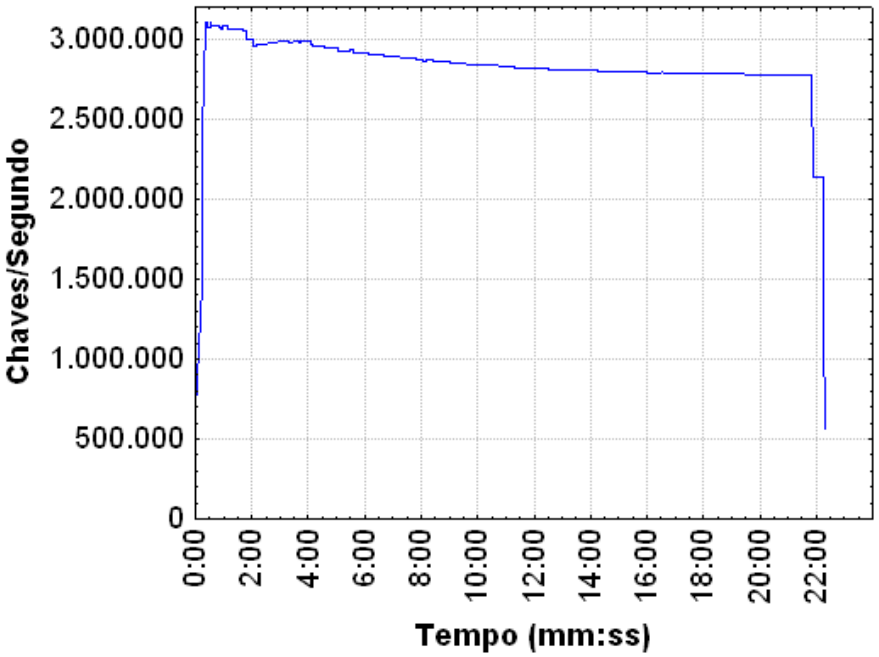

Figura 22: Chaves Testadas Global com o ATHA.

A figura 23 apresenta a CTG utilizando o ATHA-RSAE. Verificamos que o número de chaves testadas por segundo no primeiro momento ultrapassa 5.000.000, estabilizando-se em seguida um pouco abaixo desse valor. Todavia, esse número ainda é 60% superior ao apresentado na figura 22 (ATHA original). Este resultado destaca a relevância de uma técnica de escalonamento adaptativo de tarefas para um ambiente dotado de recursos com múltiplas unidades de processamento.

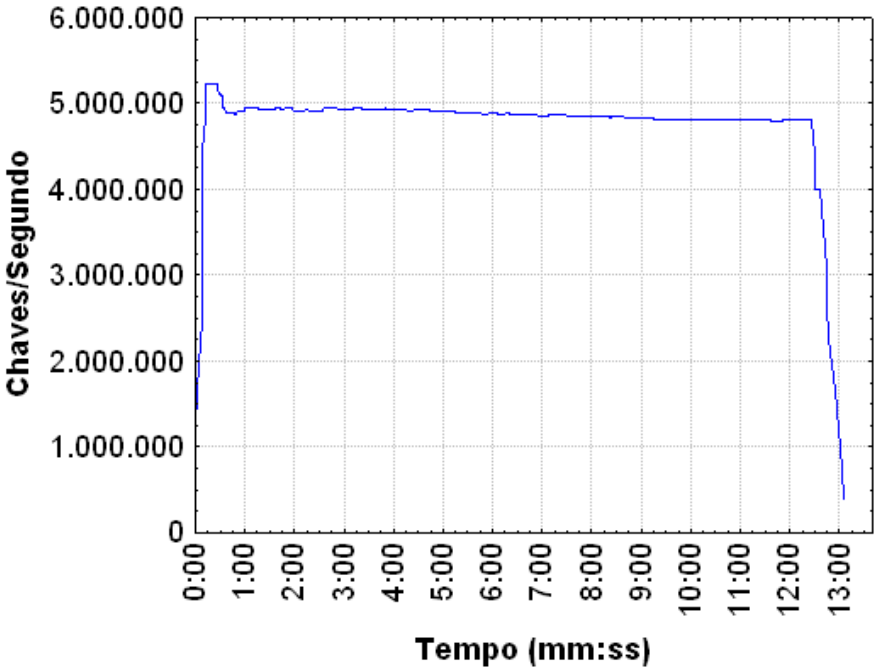

Figura 23: Chaves Testadas Global com o ATHA-RSAE.

5.2.5 Escalabilidade

Depois de realizadas as análises de performance e da capacidade de processamento, consideramos importante avaliar a escalabilidade do ATHA e do ATHA-RSAE, ou seja, verificamos o comportamento desses sistemas diante de um número crescente de máquinas escravas. Nesses testes, utilizamos os mesmos modelos de dispositivos de interconexão dos experimentos anteriores, todavia, utilizamos seis conjuntos diferentes de máquinas escravas. As configurações dos *hosts* escravos utilizados em cada bateria de teste podem ser verificadas na tabela 4. Como *host* mestre, utilizamos uma máquina com um processador Intel(R) Celeron de 1.6 GHz, 512 MByte de memória RAM, rodando o sistema operacional Linux Ubuntu 8.04.

Com a tabela 4 podemos verificar que para as três primeiras baterias de testes, consideramos apenas *hosts* multiprocessados ou que utilizam a tecnologia HTT. Apesar deste último tipo não se enquadrar na classe de máquinas com múltiplas

unidades de processamento, como já relatado, ele será considerado como tal caso pelo fato de simular paralelismo em nível de *threads*.

Para as três últimas baterias de testes, utilizamos também *hosts* com apenas um processador e que não eram dotados da tecnologia HTT. Dessa forma, tornamos o ambiente experimental mais heterogêneo e próximo da realidade em muitas organizações. Com isso, o componente *middleware* se mostrou uma peça chave do ambiente distribuído, sendo este elemento responsável por realizar uma análise de cada recurso disponível para utilizá-lo da maneira mais adequada, buscando alcançar melhores desempenhos da aplicação submetida.

Tabela 4: Ambientes Experimentais para Testes de Escalabilidade

Total Escravas	Qtd.	Arquitetura e Memória RAM	Sistema Operacional
5	1	Intel(R) Xeon(TM) 4 CPUs 3.20GHz, 6GByte	SUSE Linux 9
	1	Intel(R) Xeon(R) CPU E5310 (2CPUs QuadCore) 1.60GHz, 16GByte	SUSE Linux 10
	1	Intel(R) Pentium(R) 4 HT, CPU 2.80GHz, 1GByte	Windows XP
	2	Intel(R) Pentium(R) Core 2 Duo 1.86GHz, 1GByte	Windows XP
10	2	Intel(R) Xeon(TM) 4 CPUs 3.20GHz, 6GByte	SUSE Linux 9
	1	Intel(R) Xeon(R) CPU E5310 (2CPUs QuadCore) 1.60GHz, 16GByte	SUSE Linux 10
	4	Intel(R) Pentium(R) 4 HT, CPU 2.80GHz, 1GByte	Windows XP
	3	Intel(R) Pentium(R) Core 2 Duo 1.86GHz, 1GByte	Windows XP
15	3	Intel(R) Xeon(TM) 4 CPUs 3.20GHz, 6GByte	SUSE Linux 9
	1	Intel(R) Xeon(R) CPU E5310 (2CPUs QuadCore) 1.60GHz, 16GByte	SUSE Linux 10
	7	Intel(R) Pentium(R) 4 HT, CPU 2.80GHz, 1GByte	Windows XP
	3	Intel(R) Pentium(R) Core 2 Duo 1.86GHz, 1GByte	Windows XP
	1	Intel(R) Pentium(R) DualCore 1.60GHz, 1GByte	Windows XP
20	3	Intel(R) Xeon(TM) 4 CPUs 3.20GHz, 6GByte	SUSE Linux 9
	1	Intel(R) Xeon(R) CPU E5310 (2CPUs QuadCore) 1.60GHz, 16GByte	SUSE Linux 10
	10	Intel(R) Pentium(R) 4 HT, CPU 2.80GHz, 1GByte	Windows XP
	3	Intel(R) Pentium(R) Core 2 Duo 1.86GHz, 1GByte	Windows XP
	2	Intel(R) Pentium(R) DualCore 1.60GHz, 1GByte	Windows XP
	1	Intel(R) Celeron(R) CPU 2.4GHz, 1GByte	Windows XP
25	3	Intel(R) Xeon(TM) 4 CPUs 3.20GHz, 6GByte	SUSE Linux 9
	1	Intel(R) Xeon(R) CPU E5310 (2CPUs QuadCore) 1.60GHz, 16GByte	SUSE Linux 10
	14	Intel(R) Pentium(R) 4 HT, CPU 2.80GHz, 1GByte	Windows XP
	3	Intel(R) Pentium(R) Core 2 Duo 1.86GHz, 1GByte	Windows XP
	3	Intel(R) Pentium(R) DualCore 1.60GHz, 1GByte	Windows XP
	1	Intel(R) Celeron(R) CPU 2.4GHz, 1GByte	Windows XP
30	3	Intel(R) Xeon(TM) 4 CPUs 3.20GHz, 6GByte	SUSE Linux 9
	1	Intel(R) Xeon(R) CPU E5310 (2CPUs QuadCore) 1.60GHz, 16GByte	SUSE Linux 10
	17	Intel(R) Pentium(R) 4 HT, CPU 2.80GHz, 1GByte	Windows XP
	4	Intel(R) Pentium(R) Core 2 Duo 1.86GHz, 1GByte	Windows XP
	3	Intel(R) Pentium(R) DualCore 1.60GHz, 1GByte	Windows XP
	2	Intel(R) Celeron(R) CPU 2.4GHz, 1GByte	Windows XP

Ainda utilizando os relatórios gerados pelo o ATHA, foram geradas as figuras 24, 25, 26 e 27, com as quais é possível realizar uma análise do comportamento do ambiente de computação oportunista utilizado, diante de um número crescente de recursos escravos.

Com as figuras 24 e 25, podemos acompanhar a métrica PG (Performance Global) para cada bateria de testes realizada. Se analisarmos a linha que representa o teste com cinco máquinas, verificamos que o tempo total para o ATHA é aproximadamente 310% superior ao tempo do ATHA-RSAE. Esse número tão expressivo ocorre, porque dentre as cinco máquinas utilizadas nesse teste, quatro delas possuíam mais do que uma unidade de processamento, somando ao todo dezessete unidades de processamento executando tarefas de forma paralela. Dentre estas unidades são encontrados recursos com múltiplos processadores, processadores com múltiplos *cores* e processadores com tecnologia HTT. Já para o ATHA na sua versão original, são executadas apenas cinco tarefas simultaneamente para este teste, isto é, apenas uma tarefa é submetida para cada máquina.

Quando realizamos as mesmas comparações anteriores para trinta máquinas (veja tabela 4), esse percentual fica em torno de 230% superior. Apesar da queda deste percentual, ainda é evidente a importância de utilizar uma estratégia de escalonamento que distribui tarefas de acordo com a quantidade de unidades de processamento, quando o ambiente é dotado de um número elevado de recursos com múltiplas unidades de processamento.

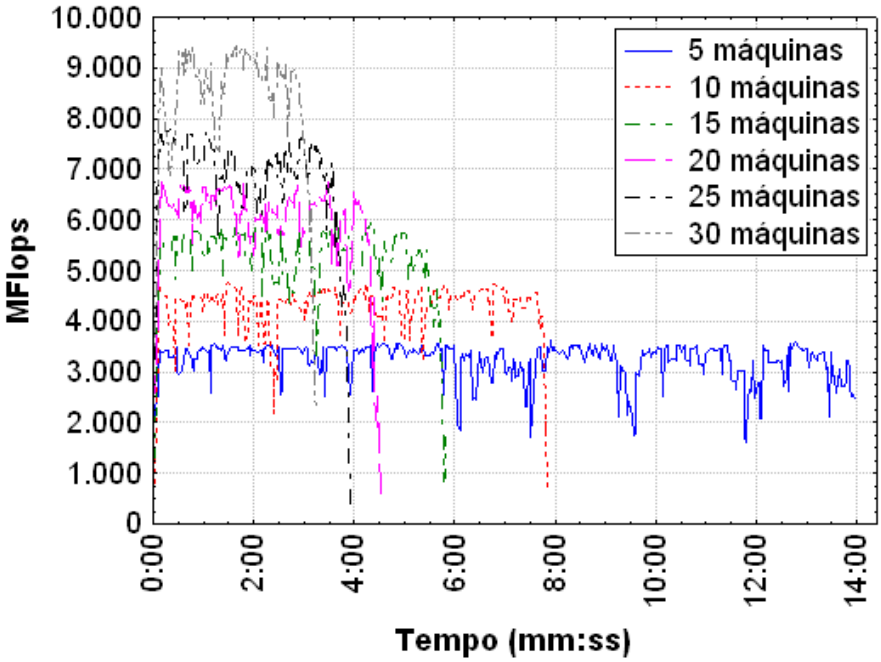

Figura 24: Performance Global (Escalabilidade) com o ATHA.

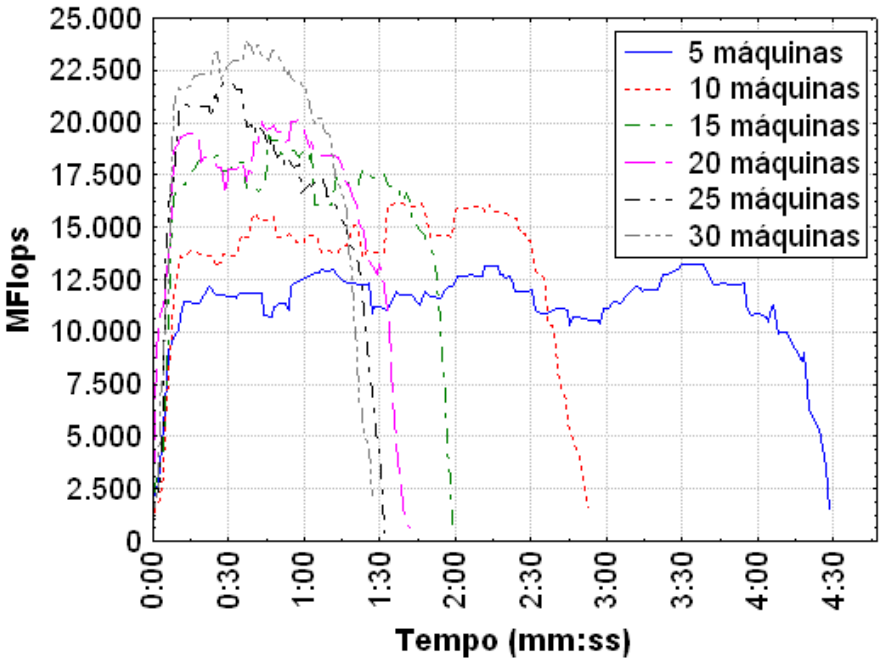

Figura 25: Performance Global (Escalabilidade) com o ATHA-RSAE.

Resultados semelhantes podem ser verificados nas figuras 26 e 27, que representam a CTG (Capacidade Total de chaves testadas por segundo Global) no ambiente. O tempo total de processamento do ATHA fica em torno de 280% superior para um

ambiente de cinco máquinas. Para trinta máquinas esse percentual é reduzido para aproximadamente 210%.

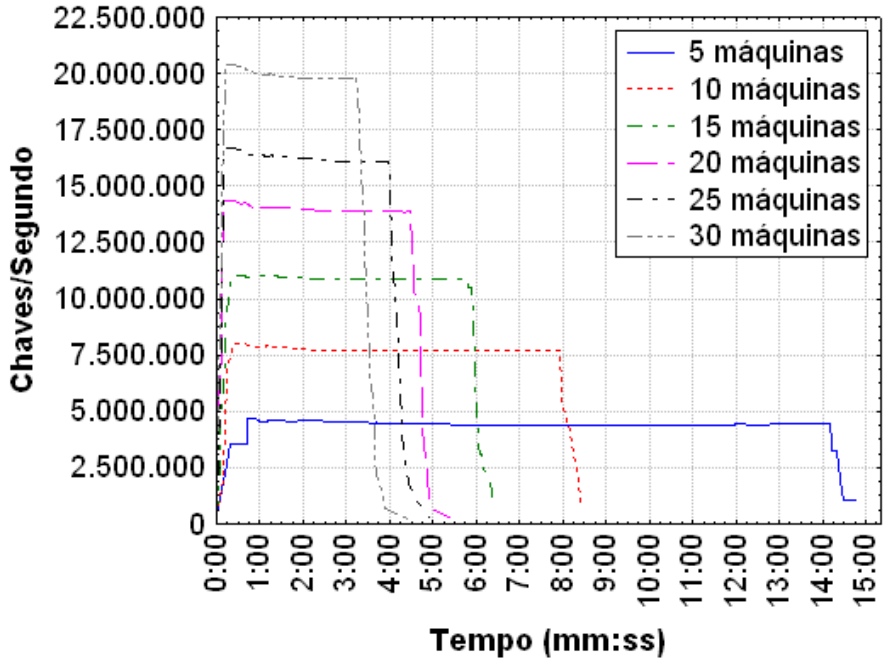

Figura 26: Chaves Testadas Global (Escalabilidade) com o ATHA.

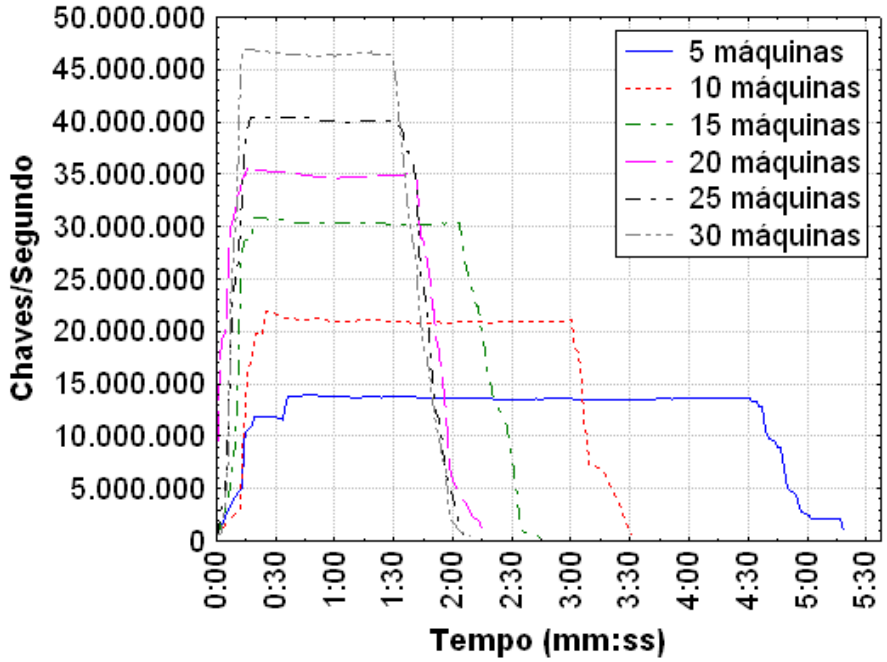

Figura 27: Chaves Testadas Global (Escalabilidade) com o ATHA-RSAE.

Com os gráficos de escalabilidade (24, 25, 26 e 27) foi possível perceber também

uma variação entre os tempos totais de processamento das figuras de PG e CTG. Entendemos que essa diferença ocorre porque o momento em que os dados são coletados para cada gráfico é diferente. Como já comentado, os dados de performance global são coletados no início de cada novo processamento remoto e o de capacidade de processamento ocorre no final, resultando assim, nos tempos totais das figuras de PG menores do que os das figuras de CTG.

Ademais, verificamos que à medida que incluímos recursos no ambiente, sendo estes recursos dotados de uma única unidade de processamento, os tempos totais, performance global e capacidade de processamento do ambiente se aproximem cada vez mais. Em outras palavras, como a abordagem de escalonamento proposta distribui um número de tarefas de acordo com o número de unidades de processamento da máquina, esta abordagem deixa de apresentar um maior destaque quando o ambiente distribuído possuir a maioria dos seus recursos com uma unidade de processamento apenas.

Para auxiliar na análise dos resultados apresentados nas figuras (24, 25, 26 e 27), tabulamos as médias dos resultados de cada gráfico na tabela 5. Nesta tabela podemos verificar que a PG e a CTG para o ATHA-RSAE permanecem superiores em todas as baterias de testes realizadas, no entanto, na medida em que o total de escravas aumenta este percentual diminui gradativamente, como já foi observado nas figuras de escalabilidade apresentadas anteriormente.

Tabela 5: Resumo dos Testes de Escalabilidade

Total Escravas	PG (Média MFlops)		CTG (Média Chaves/Seg)	
	ATHA	ATHA-RSAE	ATHA	ATHA-RSAE
5	3.222,26	11.255,56	4.343.711	12.714.018
10	4.286,50	13.367,88	7.476.334	18.933.118
15	5.249,48	15.598,26	10.347.469	25.705.345
20	5.908,92	15.799,17	13.126.129	29.105.906
25	6.729,79	16.135,36	15.084.453	31.706.499
30	8.299,64	18.288,16	17.972.047	36.677.890

Concluindo a análise de escalabilidade dos sistemas, geramos duas figuras (28 e 29) que representam o *Speedup* da PG (Processamento Global) e CTG (Chaves Testadas por segundo Global) do ambiente, respectivamente. Entretanto, para esse experimento, consideramos a métrica *Speedup* (SPG e SCTG) como sendo a razão entre os tempos totais de processamento do ATHA e do ATHA-RSAE. Em outras palavras, essa métrica apresenta de uma forma simplificada quanto o ATHA-RSAE é mais eficiente que o ATHA, na medida em que aumentamos a quantidade de máquinas do ambiente distribuído utilizado.

Esses gráficos ilustram, como esperado, que existe uma redução do SPG e SCTG quando aumentamos o número de máquinas escravas. Entendemos que uma parcela dessa redução está relacionada ao número maior de máquinas que o módulo mestre do ATHA precisa gerenciar, causando um "gargalo" nesta máquina. No entanto, podemos inferir que essa redução está relacionada, principalmente, às configurações das máquinas incluídas no ambiente, visto que a partir da bateria de testes com quinze *hosts* passamos a utilizar configurações com apenas uma unidade de processamento.

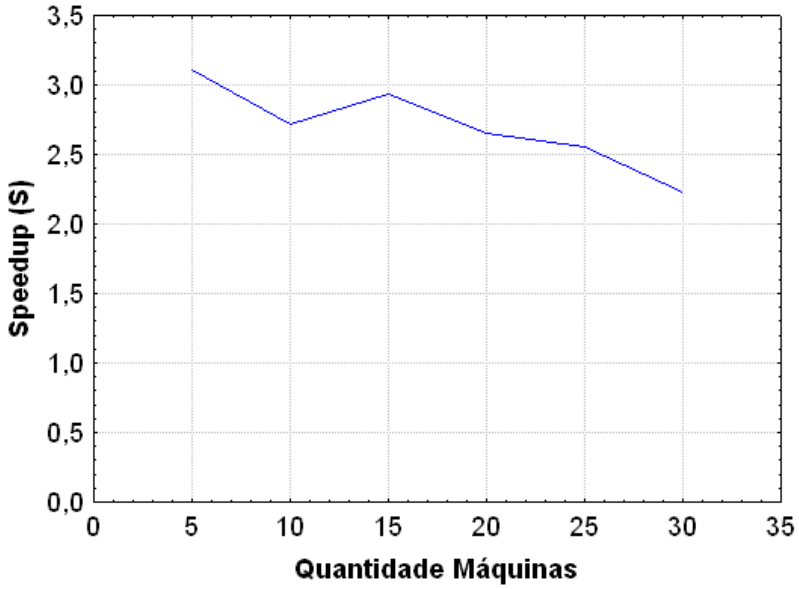

Figura 28: Speedup da Performance Global do Ambiente.

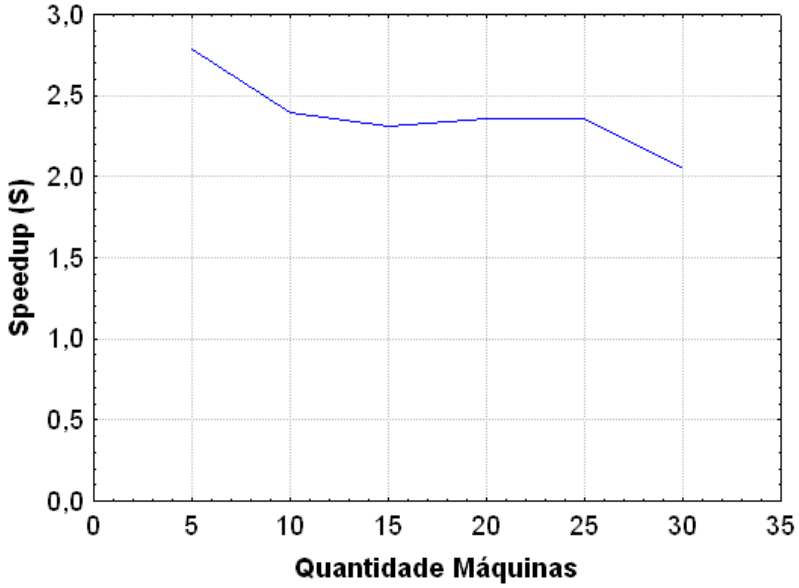

Figura 29: Speedup das Chaves Testadas pelo Ambiente.

Apesar dessa redução gradativa, é possível verificar com a figura 29, por exemplo, que o *Speedup* (SCTG) começa em 2,7 com cinco máquinas, fica em torno de 2,3 para 10, 15, 20 e 25 máquinas e reduz para 2,1 quando utilizado 30 máquinas escravas no ambiente distribuído. Apesar dos valores absolutos da figura 28 serem diferentes, os

mesmos estão próximos aos apresentados na figura 29. Novamente, podemos inferir que esta diferença é decorrente as diferentes maneiras em que as informações de P (Performance) e CT (Chaves Testadas por segundo) são coletadas.

5.3 Resumo

Nesse capítulo apresentamos os resultados empíricos obtidos neste trabalho de pesquisa. Inicialmente realizamos um estudo de caso utilizando uma máquina com quatro processadores. Na seqüência relatamos alguns resultados obtidos com a utilização de quatro máquinas escravas, sendo apresentado em detalhes o comportamento de cada unidade de processamento de cada *host*. Por fim, relatamos alguns experimentos buscando avaliar a escalabilidade dos sistemas ATHA e ATHA-RSAE.

6 Considerações Finais e Trabalhos Futuros

6.1 Conclusões

Nesta dissertação, foi proposta e implementada uma abordagem de co-escalonamento adaptativo baseada na quantidade de unidades de processamento disponíveis nos recursos de uma rede, para auxiliar na distribuição de tarefas em um ambiente de computação oportunista. Uma revisão bibliográfica realizada na literatura e alguns testes preliminares, apresentados no capítulo 4, mostraram a necessidade de uma proposta neste sentido.

Com este objetivo, utilizou-se um sistema de computação oportunista denominado ATHA, previamente desenvolvido pelo nosso grupo de pesquisa (LAPESD, 2008). Este sistema foi aprimorado para tratar múltiplas *threads* de uma aplicação de demonstração, surgindo assim o ATHA-RSAE (*Resource Scheduler ATHA Environment*). Ademais, essas *threads* são executadas de forma paralela e independente em máquinas escravas dotadas de múltiplas unidades de processamento.

Resultados experimentais indicam como esperado, que se uma aplicação não for preparada adequadamente, esta não utilizará de forma eficiente os recursos multiprocessados disponíveis em uma rede. Um exemplo disso foi a submissão de uma aplicação de busca de chaves criptográficas, que utiliza o algoritmo RC5 (RIVEST, 1996), através do sistema oportunista ATHA, processando em uma máquina escrava

de quatro processadores. Para esse experimento foi verificado que apesar da característica do algoritmo ser *CPU-Bound*, a aplicação não utilizou toda a capacidade de processamento disponível deste recurso. Contudo, quando adotada a estratégia de co-escalonamento proposta, observou-se que todas as unidades de processamento do recurso escravo foram utilizadas.

Nos testes realizados com quatro máquinas escravas, o número de chaves testadas por segundo global do ambiente obteve um aumento de 60%, quando usamos a estratégia proposta. Além disso, o tempo total de processamento da aplicação diminuiu em 56%. Isso indica que o uso de uma técnica de co-escalonamento, que submete o número de tarefas correspondente ao número de unidades de processamento disponível, garante um melhor uso dos recursos disponíveis, e conseqüentemente um menor tempo de processamento da aplicação.

A respeito da escalabilidade do sistema, foi analisado o *Speedup* do sistema, que neste caso trata-se da razão entre os tempos de execução da aplicação no ATHA pelos tempos do ATHA-RSAE. Esta métrica se mostrou em 2,7 para cinco máquinas e em 2,1 quando utilizamos trinta máquinas escravas no ambiente distribuído. Sobre esta redução, é possível de se inferir que também foi influenciada pela configuração dos recursos incluídos no ambiente distribuído, visto que nos experimentos realizados com 20, 25 e 30 *hosts* foram incluídas máquinas que não possuíam múltiplas unidades de processamento.

6.2 Trabalhos Futuros

Como trabalhos futuros, é possível estender a atual pesquisa nas seguintes direções:

- Realizar os experimentos da presente dissertação, contudo, utilizando um maior número de máquinas e procurando selecionar, preferencialmente, re-

cursos com múltiplas unidades de processamento;

- Utilizar outros índices de carga (e.g. performance individual ou chaves testadas por segundo) no módulo escalonador desenvolvido buscando tornar ainda mais eficiente a distribuição de tarefas e a utilização dos recursos disponíveis em uma rede, trazendo com isso melhores desempenhos no tempo total da aplicação;

- Outra pesquisa planejada é modificar os parâmetros da aplicação utilizada, para que o tempo de processamento seja maior e assim analisarmos o comportamento do sistema diante de problemas que necessitem ainda mais poder de processamento;

- Pretendemos também realizar testes com outras aplicações *CPU-Bound*, por exemplo, seqüenciamento molecular (STRUMPEN, 1993), otimização combinatória (PAPADIMITRIOU; STEIGLITZ, 1982) e pesquisa de números primos (GIMPS, 2008), para analisar os benefícios ao utilizar a estratégia de co-escalonamento proposta para outras aplicações;

- Desenvolver e analisar estratégias de tolerância a faltas no módulo de escalonamento desenvolvido, buscando tornar o sistema capaz de finalizar o processamento da aplicação, ainda que algum recurso do ambiente utilizado venha a falhar.

Referências

AGGARWAL, A. K.; KENT, R. D. An adaptive generalized scheduler for grid applications. IEEE Computer Society, Washington, DC, USA, p. 188–194, 2005.

ANDERSON, D. Local scheduling for volunteer computing. *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, p. 1–8, Março 2007.

BARATLOO, A. et al. Charlotte: Metacomputing on the web. 1996. Disponível em: <http://citeseer.ist.psu.edu/baratloo96charlotte.html>. Acesso em: 13 de Julho de 2008.

BERMAN, F.; FOX, G.; HEY, T. The grid: past, present, future. In: *Grid Computing ? Making the Global Infrastructure a Reality*. [S.l.]: John Wiley & Sons, Ltd, 2003. cap. 1.

BOINC. *BOINC: Berkeley Open Infrastructure for Network Computing*. 2008. Disponível em: <http://boinc.berkeley.edu/>. Acesso em: 13 de Julho de 2008.

BUYYA, R. *High Performance Cluster Computing: Architectures and Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN 0130137847.

BUYYA, R. *High Performance Cluster Computing: Programming and Applications*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN 0130137855.

BUYYA, R. *Grid Computing Info Centre (GRID Infoware)*. [S.l.], 2008. Disponível em: <http://www.gridcomputing.com/>. Acesso em: 13 de Julho de 2008.

CARINGI, A. M. *Escalonamento Estático de Processos de Aplicações Paralelas MPI em Máquinas Agregadas Heterogêneas com Auxílio de Históricos de Monitoração*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2006.

CASAVANT, T. L.; KUHL, J. G. A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 14, n. 2, p. 141–154, 1988. ISSN 0098-5589.

CHARLOTTE. *The Charlotte Research Project*. 2008. Disponível em: <http://www.cs.nyu.edu/milan/charlotte/>. Acesso em: 13 de Julho de 2008.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems (4th ed.): concepts and design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

CVETANOVIC, Z. The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems. *IEEE Transactions on Computers*, IEEE Computer Society, Los Alamitos, CA, USA, v. 36, n. 4, p. 421–432, 1987. ISSN 0018-9340.

DANTAS, M. *Computação Distribuída de Alto Desempenho (Redes, Clusters e Grids Computacionais)*. [S.l.]: Axcel Books do Brasil, 2005.

EAGER, D. L.; LAZOWSKA, E. D.; ZAHORJAN, J. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 12, n. 5, p. 662–675, 1986. ISSN 0098-5589.

EL-MOURSY, A. et al. Compatible phase co-scheduling on a cmp of multi-threaded processors. *IPDPS*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 119, 2006.

EL-REWINI, H.; LEWIS, T. G. *Distributed and parallel computing*. Greenwich, CT, USA: Manning Publications Co., 1998. ISBN 0-13-795592-8.

FARLEY, J. *Java Distributed Computing*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1998. ISBN 1565922069. Disponível em: <http://www.unix.com.ua/orelly/java-ent/dist/index.htm>. Acesso em: 17 de Julho de 2008.

FERRARI, D.; ZHOU, S. An empirical investigation of load indices for load balancing applications. North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands, p. 515–528, 1988.

FLYNN, M. J. Some computer organization and their effectiveness. *IEEE Transactions on Computers*, v. 21, p. p. 948–960, 1972.

FORTES, J. Hcw panel: programming heterogeneous systems - less pain! better performance! *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, p. 1 pp.–, 2006.

FOSTER, I. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN 0201575949. Disponível em: <http://www-unix.mcs.anl.gov/dbpp/>. Acesso em: 13 de Julho de 2008.

FOSTER, I.; KESSELMAN, C. *The grid: blueprint for a new computing infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999. ISBN 1-55860-475-8.

FREEH, V. W. A comparison of implicit and explicit parallel programming. *J. Parallel Distrib. Comput.*, Academic Press, Inc., Orlando, FL, USA, v. 34, n. 1, p. 50–65, 1996. ISSN 0743-7315.

GERASOULIS, A.; YANG, T. On the granularity and clustering of directed acyclic task graphs. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, Piscataway, NJ, USA, v. 4, n. 6, p. 686–701, 1993. ISSN 1045-9219.

GIMPS. *GIMPS: The Great Internet Mersenne Prime Search*. 2008. Disponível em: <http://www.mersenne.org/>. Acesso em: 13 de Julho de 2008.

GRANDCHALLENGE. *Grand Challenge Applications*. 2008. Disponível em: <http://www-fp.mcs.anl.gov/grand-challenges/>. Acesso em: 13 de Julho de 2008.

HOSKEN, A. *Um Ambiente para Processamento Paralelo Oportunístico na Internet*. Dissertação (Mestrado) — UnB - Brasil, 2003.

HOSKEN, A.; DANTAS, M. A. R. The atha environment: Experience with a user friendly environment for opportunistic computing. *18th International Symposium on High Performance Computing Systemsand Applications - HPCS*, 2004.

HPF. *HPF: High Performance Fortran*. 2008. Disponível em: <http://www-unix.mcs.anl.gov/dbpp/web-tours/hpf.html>. Acesso em: 13 de Julho de 2008.

HTT. *HTT: Hyper-Threading Technology*. 2008. Disponível em: <http://www.intel.com/technology/platformtechnology/hyper-threading/index.htm>. Acesso em: 13 de Julho de 2008.

JAVA. *The Source for Java Developers*. 2008. Disponível em: <http://java.sun.com/>. Acesso em: 13 de Julho de 2008.

JORDAN, L. E.; ALAGHBAND, G. *Fundamentals of Parallel Processing*. [S.l.]: Prentice Hall Professional Technical Reference, 2002. ISBN 0139011587.

KONDO, D.; ANDERSON, D. P.; VII, J. M. Performance evaluation of scheduling policies for volunteer computing. In: *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2007. p. 415–422. ISBN 0-7695-3064-8.

KRUATRACHUE, B.; LEWIS, T. Grain size determination for parallel processing. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 5, n. 1, p. 23–32, 1988. ISSN 0740-7459.

LAPESD. *LAPESD: Laboratório de Pesquisa de Sistemas Distribuídos, UFSC-CTC-INE*. 2008. Disponível em: <http://www.lapesd.inf.ufsc.br/>. Acesso em: 13 de Julho de 2008.

LINPACK. *Optimizing Java Linpack*. 2008. Disponível em: <http://www.cs.cmu.edu/%7Ejch/java/linpack.html>. Acesso em: 13 de Julho de 2008.

LITZKOW, M.; LIVNY, M.; MUTKA, M. Condor - a hunter of idle workstations. June 1988.

MENDONÇA, R.; DANTAS, M. A study of adaptive co-scheduling approach for an opportunistic software environment to execute in multi-core and multi-processor configurations. *IEEE 11th International Conference on Computational Science and Engineering - CSE*, 2008.

MENDONÇA, R.; DANTAS, M. Uma abordagem de co-escalonamento adaptativo para ambientes de processamento oportunista multiprocessados. *8ª Escola Regional de Alto Desempenho - ERAD*, 2008.

MPI. *MPI: Message Passing Interface*. 2008. Disponível em: <http://www-unix.mcs.anl.gov/mpi/>. Acesso em: 13 de Julho de 2008.

NAGAR, S. et al. A closer look at coscheduling approaches for a network of workstations. In: *SPAA '99: Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM, 1999. p. 96–105. ISBN 1-58113-124-0.

NISAN, N. et al. Globally distributed computation over the internet-the popcorn project. *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on*, p. 592–601, 1998. ISSN 1063-6927.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982. ISBN 0-13-152462-3.

PFISTER, G. F. *In search of clusters (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998. ISBN 0-13-899709-8.

PINTO, A. S. R. *Abordagem de Escalonamento Dinâmico de Tarefas Baseada em Sistemas Classificadores*. Dissertação (Mestrado) — Universidade Federal de Santa Catarina - UFSC, 2004.

PINTO, A. S. R.; DANTAS, M. Uma abordagem de balanceamento de carga baseada em algoritmo de aprendizado de máquina genético. *5º Workshop de Computação de Alto Desempenho, Brasil, Foz do Iguaçu*, 2004.

PINTO, L. C.; MENDONÇA, R. P.; DANTAS, M. Impact of interconnection networks and application granularity to compound cluster environments. *IEEE Symposium on Computers and Communications - ISCC*, 2008.

POURREZA, H.; GRAHAM, P. On the programming impact of multi-core,multi-processor nodes inmpi clusters. IEEE Computer Society, Washington, DC, USA, p. 1, 2007.

PVM. *PVM: Parallel Virtual Machine*. 2008. Disponível em: <http://www.csm.ornl.gov/pvm/>. Acesso em: 13 de Julho de 2008.

QUINN, M. J. *Parallel computing (2nd ed.): theory and practice*. New York, NY, USA: McGraw-Hill, Inc., 1994. ISBN 0-07-051294-9.

RIVEST, R. L. The RC5 encryption algorithm, from dr. dobb's journal, january, 1995. In: *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996*. [s.n.], 1996. Disponível em: <http://citeseer.ist.psu.edu/rivest95rc.html>. Acesso em: 17 de Julho de 2008.

SETI@HOME. *SETI@home*. 2008. Disponível em: <http://setiathome.berkeley.edu/>. Acesso em: 13 de Julho de 2008.

SHIRAZI, B. A.; KAVI, K. M.; HURSON, A. R. *Scheduling and Load Balancing in Parallel and Distributed Systems*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1995. ISBN 0818665874.

SHIVARATRI, N. G.; KRUEGER, P.; SINGHAL, M. Load distributing for locally distributed systems. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 25, n. 12, p. 33–44, 1992. ISSN 0018-9162.

SODAN, A. C. Loosely coordinated coscheduling in the context of other approaches for dynamic job scheduling: a survey: Research articles. *Concurr. Comput. : Pract. Exper.*, John Wiley and Sons Ltd., Chichester, UK, UK, v. 17, n. 15, p. 1725–1781, 2005. ISSN 1532-0626.

SOUSA, P. B. M. de. *WVM: Uma Ferramenta para Processamento Distribuído de Alto Desempenho na Web*. Dissertação (Mestrado) — Universidade Federal do Ceará, Outubro 2001.

SQUILLANTE, M. S. et al. Modeling and analysis of dynamic coscheduling in parallel and distributed environments. ACM, New York, NY, USA, p. 43–54, 2002.

STRUMPEN, V. Parallel molecular sequence analysis on workstations in the internet. 1993. Disponível em: <http://citeseer.ist.psu.edu/strumpen93parallel.html>. Acesso em: 13 de Julho de 2008.

TANENBAUM, A. S. *Modern Operating Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130313580.

TANENBAUM, A. S.; STEEN, M. van. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 0132392275.

TOP. *Unix Top*. 2008. Disponível em: <http://www.unixtop.org/>. Acesso em: 13 de Julho de 2008.

ZHOU, S. A trace-driven simulation study of dynamic load balancing. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 14, n. 9, p. 1327–1341, 1988. ISSN 0098-5589.

Apêndices

Apêndice A - Publicações

Título: Uma Abordagem de Co-Escalonamento Adaptativo para Ambientes de Processamento Oportunista Multiprocessados

Evento: Escola Regional de Alto Desempenho - ERAD 2008

Local: Santa Cruz do Sul - RS - Brasil **Data:** 11/03/2008 a 14/03/2008

Autores: Rodrigo Paiva Mendonça e Mário Antônio Ribeiro Dantas

Título: Impact of Interconnection Networks and Application Granularity to Compound Cluster Environments

Evento: IEEE Symposium on Computers and Communications - ISCC 2008

Local: Marrakech - Morocco **Data:** 06/07/2008 a 09/07/2008

Autores: Luiz Carlos Pinto, Rodrigo Paiva Mendonça e Mário Antônio Ribeiro Dantas

Título: A Study of Adaptive Co-Scheduling Approach for an Opportunistic Software Environment to Execute in Multi-core and Multi-Processor Configurations

Evento: IEEE 11th International Conference on Computational Science and Engineering - CSE 2008

Local: São Paulo - SP - Brasil **Data:** 16/07/2008 a 18/07/2008

Autores: Rodrigo Paiva Mendonça e Mário Antônio Ribeiro Dantas

Título: Uma Abordagem de Co-Escalonamento Adaptativo para Ambientes de Computação Oportunista de Configurações Multiprocessadas

Evento: Conferencia Latinoamericana de Informática - CLEI 2008

Local: Santa Fé - Argentina **Data:** 08/09/2008 a 12/09/2008

Autores: Rodrigo Paiva Mendonça, Marcio M. Piffer, Diogo R. Viegas e Mário Antônio Ribeiro Dantas

Apêndice B - Diagramas de Classe Simplificado do ATHA

Módulo Mestre

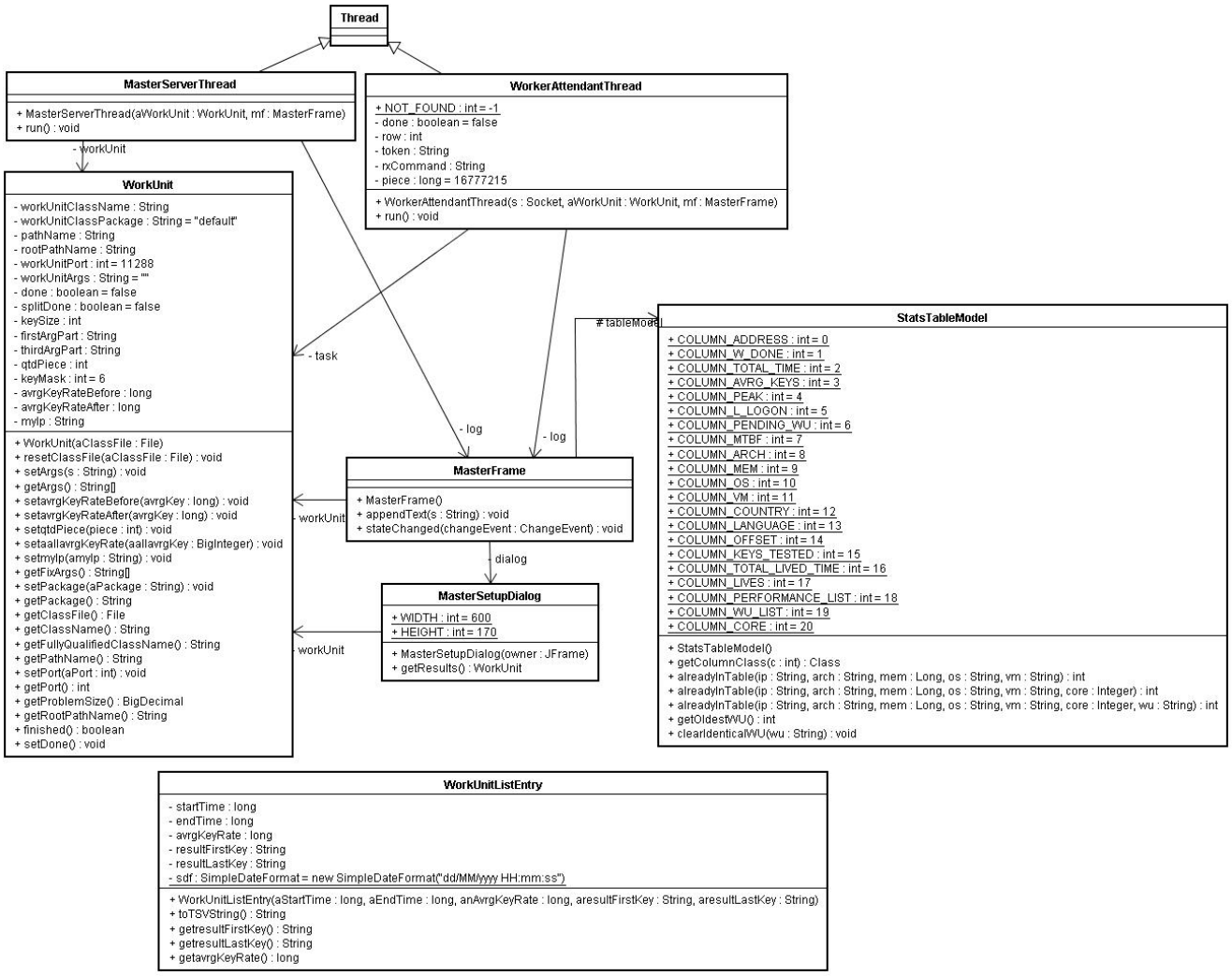

Figura 30: Diagrama Classe Módulo Mestre ATHA.

Módulo Escravo

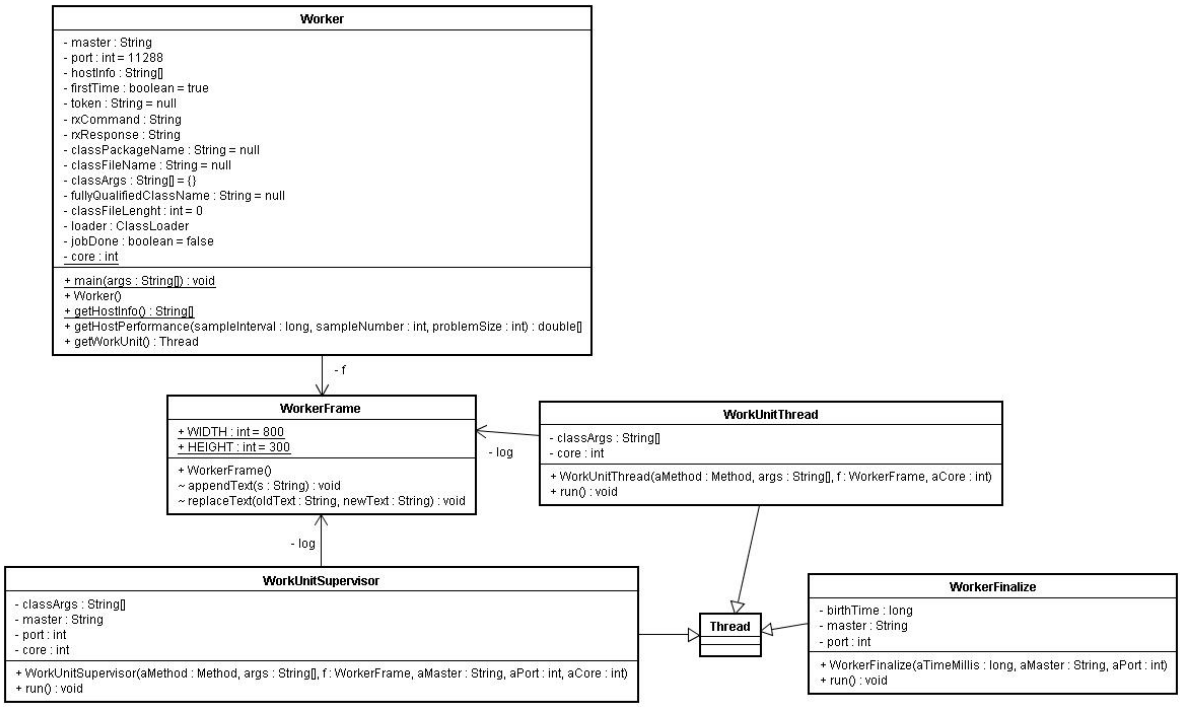

Figura 31: Diagrama Classe Módulo Escravo ATHA.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo