Marcelo Massocco Cendron

ESCALONAMENTO DE TAREFAS BASEADO EM LEILÃO DE RECURSOS NO MIDDLEWARE GRID-M

Florianópolis – SC
2008

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

Marcelo Massocco Cendron

ESCALONAMENTO DE TAREFAS BASEADO EM LEILÃO DE RECURSOS NO MIDDLEWARE GRID-M

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

Carlos Becker Westphall

Florianópolis, Fevereiro/2008

Escalonamento de Tarefas Baseado em Leilão de Recursos no Middleware Grid-M

Marcelo Massocco Cendron

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

_____

Prof. Dr. Mário Antonio Ribeiro Dantas
Coordenador do Programa de Pós-Graduação em Ciência da Computação

Banca Examinadora

_____

Prof. Dr. Carlos Becker Westphall
Orientador

_____

Prof. Dr. Bruno Richard Schulze

_____

Prof. Dr. Mário Antonio Ribeiro Dantas

_____

Prof.ª Dra. Carla Merkle Westphall

AGRADECIMENTO

Agradeço a meu professor orientador Dr. Carlos Becker Westphall pela confiança, disposição e ajuda durante o período em que estive desenvolvendo meus estudos de mestrado.

Também não posso deixar de agradecer aos meus amigos Fernando Koch e Marcos dias de Assunção pelos conselhos, dicas e principalmente pela paciência que demonstraram em estar colaborando nesses estudos.

Aos meus colegas de laboratório, Carlos Oberdam Rolim, Kleber Magno Maciel, Leonardo Kunrath e Gabriel Queiroz Lana pelo companheirismo e disposição em estar sempre ajudando.

Aos meus pais, que me ensinaram a perseguir meus sonhos e batalhar por um ideal.

Aos meus sogros, pelo apoio moral nos momentos em que o desânimo estava batendo.

E principalmente a minha esposa Tatiane, que esteve em todos os momentos ao meu lado, mesmo nos mais difíceis, sendo paciente durante os momentos de crise e colaborativa na revisão desse trabalho.

SUMÁRIO

AGRADECIMENTO iii

SUMÁRIO iv

LISTA DE FIGURAS vii

LISTA DE TABELAS viii

LISTA DE ABREVIATURAS ix

RESUMO x

ABSTRACT xi

1 - INTRODUÇÃO 12

1.1 - Objetivo geral 15

1.2 - Objetivos específicos 15

1.3 - Justificativa e Trabalhos Correlatos 15

1.4 - Organização do Trabalho 18

2 - ECONOMIA GRID 19

2.1 - Conceitos sobre Economia Grid 19

2.2 - Modelos de mercado 23

2.2.1 - Mercado de commodities 24

2.2.2 - Leilão 25

2.3 - Modelo de Recurso 26

2.3.1 - Controle de Gerenciamento 27

2.3.2 - Composição de recursos 28

2.3.3 - Suporte a escalonamento 28

2.3.4 - Mecanismo de Contabilidade 29

2.4 - Definição de Recursos 29

2.4.1 - Taxonomia dos recursos 32
2.4.2 - Recursos relevantes 35
2.4.3 - Combinando Recursos 36
3 - GRID-M E ECONOMIA 38
3.1 - O Grid-M 38
3.2 - Grid-M Economy 46
3.3 - Funções de utilidade 49
3.3.1 - Processo de leilão 52
4 - MODELO DE ESCALONAMENTO PROPOSTO 54
4.1 - Agentes de Referencial 55
4.2 - Agentes Fornecedores 56
4.2.1 - Serviço de estimativa de custo. 57
4.2.2 - Serviço de teste *RunTask* 60
4.3 - Agentes Consumidores 61
5 - RESULTADOS EXPERIMENTAIS 64
5.1 - Ambiente 64
5.2 - Experimentos 66
6 - CONCLUSÃO 76
6.1 - Trabalhos Futuros 77
REFERÊNCIAS BIBLIOGRÁFICAS 79
ANEXOS 83
ANEXO 1 84
ANEXO 2 85
ANEXO 3 87
ANEXO 4 91
ANEXO 5 92

ANEXO 6 94

ANEXO 7 95

LISTA DE FIGURAS

Figura 1 - Modelo de arquitetura de um *Grid* com enfoque econômico 22
Figura 2 - Taxonomia dos Modelos Econômicos 24
Figura 3 - Taxonomia dos Modelos de Recursos. 27
Figura 4 - Interface de serviço 39
Figura 5 - Classes *Node* e *Service* 41
Figura 6 - Exemplo de extensão de um serviço 42
Figura 7 - Serviço Multiplicar 43
Figura 8 - Tupla de uma tarefa 44
Figura 9 – Parâmetros e Solicitação de execução de uma tarefa 44
Figura 10 - Parâmetros de uma tarefa 44
Figura 11 - Envio da execução da tarefa 45
Figura 12 - Retorno do resultado 46
Figura 13 - Dependência entre recursos 48
Figura 14 – Interface Utilities 50
Figura 15 - Variação da carga do recurso 51
Figura 16 - Processo de leilão 52
Figura 17 - Diagrama da plataforma computacional 54
Figura 18 - Diagrama do agente Referencial 55
Figura 19 - Estimativa de recursos 57
Figura 20 - Verificação da dependência da Classe *Metrics* 58
Figura 21 - Requisição do preço inicial do nodo 58
Figura 22 - Variação da carga 60
Figura 23 - Solicitação de orçamento 62
Figura 24 - Retorno do resultado 62
Figura 25 - Thread de execução 63
Figura 26 - Resumo gráfico das execuções 67
Figura 27 - Variação de preço no experimento 1 73
Figura 28 - Variação de preço no experimento 2 74
Figura 29 - Variação de preço no experimento 3 74

LISTA DE TABELAS

Tabela 1 - Classificação dos recursos 32
Tabela 2 - Classificação de alguns recursos..... 35
Tabela 3 – Informações sobre as execuções 64
Tabela 4 - Características dos nodos..... 65
Tabela 5 - Índice de desempenho..... 66
Tabela 6 - Resumo execuções..... 66
Tabela 7 - Tempo de execução por processador 67
Tabela 8 - Número de execuções 68
Tabela 9 - Resumos dos preços nos experimentos..... 69
Tabela 10 - Resumo do tempo para retorno dos orçamentos. 71

LISTA DE ABREVIATURAS

CPU	Central Processing Unit (Unidade Central de Processamento)
FIPA	Foundation for Intelligent Physical Agents
HTTP	Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)
IP	Internet Protocol (Protocolo de Internet)
P2P	Peer-to-Peer (Par-a-Par)
QoBiz	Quality of Business (Qualidade do negócio)
QoE	Quality of Experience (Qualidade da Experiência)
QoS	Quality of service (Qualidade de serviço)
SPEC	Standard Performance Evaluation Corporation
WBEM	Web-Based Enterprise Management
WMI	Windows Management Instrumentation

RESUMO

O escalonamento de tarefas em sistemas distribuídos é uma área que ainda há necessidade de pesquisas, tanto pelas promissoras evoluções previstas nos sistema de Grid e P2P, quanto nas dificuldades de conciliar o funcionamento de todos os tipos e configurações de nodo que podem existir nesse tipo de rede.

Dentro da natureza humana, o sistema econômico já é um modelo de negociação consolidado e fundamentado, o que lhe garante confiança em seus conceitos apresentados. Essa confiança é o que se busca para que seja utilizado no escalonamento de tarefas em um Grid computacional.

Baseado nesses fundamentos, nesse trabalho procuramos estender as pesquisas que estão sendo feitos na área de Economia Grid, que nada mais é do que a junção dos conceitos de economia com esse tipo de sistema distribuído denominado Grid. Mas especificamente, propomos uma arquitetura em que o escalonamento das tarefas seja realizado de forma justa ao desempenho de cada nodo, atribuindo mais tarefas para os nodos que possuem melhores condições de atender a requisição.

Palavras-chaves: Escalonamento; Economia Grid; Formação de Preço; Modelos econômicos.

ABSTRACT

Task scheduling in distributed systems is still an area in need of research, as much because of the promising improvements in Grid and other P2P systems as because of the difficulties in conciliating all the kinds of node configurations that are permitted in such systems.

In human affairs the economic system is an already consolidated and well founded negotiation model, which gives confidence to its concepts. This trust is what we seek for task scheduling in computational grids.

On this basis, this work explores the research being conducted on Grid Economy, which is nothing more than the linking of economy notions with grid systems. More specifically, we describe an architecture in which task scheduling is performed fairly in regards to each node's performance capabilities, assigning more tasks to nodes which are more capable of responding to requests.

1 - INTRODUÇÃO

A necessidade de computadores com alto desempenho não é recente, desde o principio da computação há a necessidade de realizar tarefas que exigem uma quantidade considerável de recursos, geralmente de processamento. Essa necessidade levou os pesquisadores e as empresas ligadas a área de computação a desenvolverem sistemas que possibilitassem satisfazer inicialmente a necessidade de processamento. A primeira opção é apresentada sobre a forma de máquinas de grandes dimensões, dispostas em salas de igual tamanho, e com altíssima capacidade de processamento.

O poder de computacional desses equipamentos é respeitado, tanto que é elaborado um ranking (TOP500.Org 2007) dos computadores com maior desempenho do mundo. Porém esse tipo de dispositivo tem o inconveniente de ser um equipamento caro, inacessível para muitas empresas que não estão ligadas a área de computação e que percebem a necessidade de ter um equipamento de alto desempenho em sua infra-estrutura.

A partir dos anos 80, a popularização do micro-computador (Tanenbaum and van Steen 2002), expandiu a quantidade de equipamentos com razoável desempenho e preço acessível. Juntamente a essa evolução, deu-se a expansão de redes de comunicação em larga escala, garantindo que os computadores fossem interligados em velocidade cada vez maior, permitindo o desenvolvimento de sistemas que possam realizar a divisão de tarefas entre nodos dispostos em locais diferentes. Iniciou-se, a partir desse momento, a expansão dos sistemas distribuídos.

Segundo Tanenbaum e van Steen (Tanenbaum and van Steen 2002), um sistema distribuído é “uma coleção de computadores independentes que aparecem para os usuários do sistema como um simples computador”. Os sistemas distribuídos são focos de várias pesquisas na área computacional e sua utilização vem crescendo em todos os campos.

Consoante com Foster (Foster 2002), os sistemas distribuídos apresentam características que permite classificá-los em três tipos: cluster, *Grid* computacional e P2P, cada um com particularidades e utilização distintas, porém com princípios de distribuição de recursos comum a todos. Dentro desses sistemas o *Grid* computacional

(Foster, Kesselman et al. 2001; Buyya 2002), juntamente com os sistemas P2P, tem se mostrado tecnologias promissoras para a utilização em nosso cotidiano.

O *Grid* computacional (mantido em inglês, por considerarmos que dentro da área de computação, o termo possui uma clara distinção em comparação ao outros significados e traduções apresentadas), inicialmente denominado de Metacomputação (Smarr and Catlett 1992; de Roure, Baker et al. 2003), consiste na utilização de nodos, geograficamente distribuídos, fornecendo algum tipo de recurso para quem necessitar desses recursos em qualquer lugar que esteja conectado.

A infra-estrutura de um *Grid* está sendo amplamente estudada, e vários trabalhos (Berman, Fox et al. 2003) concretizam os conceitos apresentados para esse tipo de sistema distribuído (Foster, Kesselman et al. 2001; Berman, Fox et al. 2003; Rajkumar Buyya 2003), inclusive convergindo suas diretrizes para dispositivos móveis e autônomos (McKnight, Anius et al. 2002; Xue, Li et al. 2003; McKnight, Howison et al. 2004; Wang, Yu et al. 2005; Ahuja and Myers 2006), com cada vez maior inserção de dispositivos considerados heterogêneos.

Com esses avanços, o foco inicial do *Grid* computacional, que era atender exclusivamente a tarefas científicas e acadêmicas (GridCafé 2007), passa a seguir outras direções e o *Grid* gradativamente está se expandido para mercados maiores, com sua utilização em empresas e instituição fora do meio acadêmico.

A evolução dos *Grid* computacionais se dá também em áreas econômicas, onde os já consolidados princípios do mercado econômicos são utilizados para atribuir novas funcionalidades e melhorar o desempenho dos serviços de um *Grid.* Basicamente, os modelos econômicos estão sendo utilizados em duas frontes. A primeira se destina a utilizar princípios econômicos para atribuir um valor financeiro aos serviços prestados por um *Grid*, esse tipo de funcionalidade é também chamado de computação utilitária. A segunda está focada na utilização desses conceitos para aprimorar as funcionalidades do *Grid* principalmente no que tange o escalonamento e balanceamento de carga.

A disponibilização de um serviço como um serviço utilitário (Rappa 2004; Bunker and Thomson 2006), já se apresenta em forma comercial, como é o caso do Sun *Grid* Compute Utility (Sun Microsystems 2007), um sistema que disponibiliza poder de processamento para a execução de certos serviços pelo custo de $1 (um dólar) por uso de um processador a cada hora. Esse tipo de serviço tenta se aproximar ao sistema que

deu origem ao nome de *Grid* computing, que seria o sistema de Distribuição de Energia Elétrica.

O sistema econômico também favorece o aprimoramento dos sistemas de balanceamento e escalonamento de tarefas. Tendo em vista que os sistemas econômicos por natureza são independentes e distribuídos (Wolski, Plank et al. 2000), a utilização desses conceitos possibilitam aos sistemas computacionais uma autonomia na execução dos serviços prestados, principalmente em *Grid*, possibilitando que a estrutura não necessite de sistemas centrais, passíveis de falhas e por conseqüente de paralisação do sistema como um todo.

Entre as duas abordagens, porém existe um ponto em comum, que é a definição de um valor para o serviço. Esse fator se torna complexo à medida que analisamos os requisitos que a função de utilidade deve satisfazer: deve ser abrangente de forma que garanta retorno financeiro pelo investimento feito na infra-estrutura e custos de manutenção, e ainda deve ser simples (Bunker and Thomson 2006) para que o usuário entenda o valor que está sendo cobrado.

Na parte de escalonamento, o algoritmo para definição de custo deve ser capaz de perceber os nuances de desempenho geral do sistema e transformar esses detalhes em valores que simbolizem o real estado do nodo.

As dificuldades apresentadas ainda são agravadas pelas diversas configurações e tipo de arquiteturas que podem ser encontradas nos mais diferentes tipos de dispositivos que fazem parte de uma infra-estrutura de *Grid*. Dessa constatação elaboramos a seguinte pergunta:

Quais recursos são importantes que sejam utilizados para a formação de preço de um serviço?

A partir dessa pergunta, temos uma base que será o direcionamento de nosso trabalho. Analisando os fatores envolvidos num sistema distribuído, é definida outra pergunta:

Como transformar a utilização de cada recurso em uma unidade comum (preço)?

Partimos dessa pergunta para a elaboração da proposta presente nesse trabalho, porém podemos visualizar que estamos sendo pontuais na definição do tema, já que a formação de preço para o consumidor pode dar-se por diferentes métricas, tanto quantitativas quanto qualitativas.

1.1 - Objetivo geral

O objetivo geral desse trabalho é desenvolver uma infra-estrutura que permita fazer a distribuição de tarefas de forma eficiente e equilibrada entre os nodos que possuem diferentes capacidades de recursos.

1.2 - Objetivos específicos

Os objetivos específicos desse são trabalhos são os seguintes:

- Estimar os recursos quantitativos que impactam no custo da execução de um serviço.
- Propor formas de quantificar os recursos que serão necessários por um serviço e definir um preço baseado nesses valores.
- Estender uma infra-estrutura de *Grid* em desenvolvimento para que suporte o escalonamento de tarefas baseada num valor único para todos os recursos e dispositivos.

1.3 - Justificativa e Trabalhos Correlatos

Desde a apresentação dos trabalhos de Ferguson (Ferguson 1989) e de Waldspurger et al (Waldspurger, Hogg et al. 1992), a aplicação de métodos econômicos tem sido amplamente utilizadas. Os modelos econômicos que atualmente são utilizados para as transações no dia-a-dia, com a já conhecida distinção entre consumidores/fornecedores ou clientes/fornecedores, tem se mostrado (Nakai 2001)

como um modelo promissor para o escalonamento de tarefas dentro de sistemas distribuídos.

Alguns fatores que levam a considerar modelos econômicos dentro de sistemas distribuídos são(Wolski, Plank et al. 2001):

- Recursos não são de graça: inicialmente, o uso de *Grid* se restringia a instituições acadêmicas e de pesquisas, por isso, o fator custo não tinha tanto peso, porém a aceitação comercial do *Grid* vai depender do retorno do investimento feito na infra-estrutura.
- O dinamismo do desempenho de um *Grid* é difícil de modelar: condições do *Grid* podem sofrer mudanças em pouco tempo, como por exemplo, a sobrecarga de um recurso. São necessários mecanismos que permite ao Grid readaptar-se a essas mudanças e manter-se em condição ótima de funcionamento. Mecanismos econômicos já são estudados há muito tempo para minimizar essas mudanças bruscas e podem ser aplicados aos sistemas de *Grid*

Se um usuário decidir que quer disponibilizar seus recursos em um *Grid* ele poderá estar recebendo um valor financeiro, baseado em quanto de recurso está sendo disponibilizado.

Dentre essas características vários conceitos econômicos estão envolvidos, a divisão dos principais módulos envolvidos é apresentada por Kenyon e Cheliotis (Cheliotis, Kenyon et al. 2004), Nesta divisão, cada um dos módulos, também chamadas de pilhas, é dividido em várias camadas, cada uma responsável por uma atividade específica. Abordaremos com maiores detalhes essa Pilha de comercialização de um *Grid* no Capítulo 2.

Para o contexto desse trabalho, a abordagem se dá na formação de preço, dentro desses aspectos alguns trabalhos se destacam pela preocupação de consolidar a utilização dos recursos na forma de um valor simplificado, dentre eles destacamos alguns na seqüência do texto.

O projeto Tycoon (Lai, Rasmusson et al. 2005), é um projeto desenvolvido nos laboratórios da HP em Palo Alto e que provê meios para fazer o escalonamento de tarefas em nodos através de comandos semelhantes a comandos de *shell*, um servidor central faz a parte de registrador de serviços, que constantemente realiza atualizações do

estado dos leiloeiros. Os leiloeiros são os intermediadores que tem quatro funções principais: gerenciar os recursos locais, coletar as requisições dos usuários, alocar os recursos dos usuários de acordo com suas requisições e anunciar a disponibilidade de recursos locais. Dentro do projeto, os nodos possuem diferentes capacidades de processamento que é considerada durante a execução. Além dos servidores de registro de serviços, há o servidor de Banco, que é responsável pelo controle financeiro do sistema e é disposto de forma única para todo o sistema.

Com foco em dispositivos móveis e com limitações de recursos, o NWSLite (Gurun, Krintz et al. 2004) preocupa-se com o balanceamento de cargas entre nodos, ponderando os fatores que são necessários para migrar uma tarefa com os benefícios que essa migração pode resultar. São considerados valores de processamento e taxas de transmissão de rede, esses valores são calculados e avaliados de forma a avaliar o beneficio da migração, o NWSLite não apresenta modelos econômicos que incentive a competição entre os nodos, porém, seus conceitos se encaixam na formação de preço para um serviço.

Reddy (Reddy 2006), em sua dissertação de mestrado, propõe três políticas de escalonamento de tarefas utilizando conceitos de economia. Dentro das simulações foram utilizados nodos com diferentes capacidades de processamento e essa diferença foi utilizada para equilibrar a distribuição das tarefas entre os nodos. As diferenças de desempenho dos nodos são mensuradas em tempo de execução (Reddy and Gupta 2006). Quando uma tarefa é executada, o tempo que a tarefa levou para ser concluída e a quantidade de ciclos que foi designado para a tarefa são os fatores que definem a capacidade de processamento do nodo. Nessa abordagem, qualquer atividade que comprometa o tempo que a tarefa demore em executar, pode estar mascarando o real desempenho do recurso.

Os trabalhos relacionados a economia em *Grid* não se resumem a somente esses acima citados, Yeo e Buyya (Yeo and Buyya 2006) faz uma comparação detalhada, inclusive com a taxonomia dos sistemas de economia em *Grid* o que limitou a menção dos trabalhos nessa dissertação é a preocupação dos trabalhos em lidar com recursos heterogêneos e baseados neles estipular um preço comum para os serviços, não que os trabalhos não mencionados não tratem desse fator, mas a profundidade abordada, em

maior ou menor grau em cada trabalho, não condiz com o contexto pretendido com essa dissertação.

1.4 - Organização do Trabalho

De forma a desenvolver de forma gradual a proposta, esse trabalho é dividido nos seguintes capítulos:

Capítulo 2 – Aborda os conceitos de economia Grid, esses conceitos serão utilizados na implementação para realizar a distribuição de tarefas entre os nodos, esse capítulo descreve modelos econômicos e a classificação desses modelos na área de sistemas distribuídos. São descritos também, os tipos de recurso comumente encontrados em dispositivos computacionais, uma classificação desses recursos e como eles são utilizados na proposta.

Capítulo 3 – É descrito o ambiente de infra-estrutura Grid-M, suas características, funcionalidades e exemplo de utilização, em breve é apresentado a proposta de extensão do projeto Grid-M de forma que possa realizar a formação de preço para um serviço e implementar os serviços necessários para a realização dos leilões que fazem o escalonamento de tarefas.

Capítulo 4 – Detalhes da implementação são expostos, com ênfase na programação e arquitetura, como foram realizados os experimentos e detalhes que foram precisos ser considerados durante o desenvolvimento.

Capítulo 5 – São apresentados os resultados obtidos na execução do sistema, comprovando as funcionalidades e atendendo as propostas expostas durante a definição do problema.

Capítulo 6 – Finalizando o trabalho, são apresentadas conclusões sobre a realização desse trabalho, bem como sugestões de trabalhos futuros que podem ser agregados ao ambiente, bem como novas características que podem ser exploradas em trabalho a parte.

2 - ECONOMIA GRID

Nesse capítulo são abordados os conceitos pertinentes a utilização de conceitos econômicos dentro de um *Grid* computacional, esses aspectos são importantes para situar o trabalho proposto dentro do que se observa sobre Economia Grid

2.1 - Conceitos sobre Economia Grid

O *Grid* tem expandido seu horizonte e aos poucos começa a se propagar fora dos meios acadêmicos, se seus conceitos já são explorados e utilizados para pesquisas científicas sem fins lucrativos, fora desse ambiente acadêmico a ordem é outra, e o *Grid* passa a sofrer pressões financeiras. Cheliotis et al (Cheliotis, Kenyon et al. 2004) foca que não basta dispor de uma infra-estrutura, com alto investimento realizado, se esse agregado de computadores não provê o devido retorno financeiro, a ordem passa a ser: "*agora você tem um Grid quem obtêm o que, quando e por quanto?"*.

Apesar dessa visão financeira de um serviço prestado por um *Grid* a utilização de conceitos econômicos não é focada apenas nesse propósito, a princípio duas direções estão sendo tomadas no que diz respeito aos modelos econômicos aplicados a sistemas distribuídos:

- Comercialização de serviços(Rappa 2004; Sun Microsystems 2007): segue a idéia de Utilidade *(Utility),* que considera um serviço como uma atividade econômica e quantificada por um valor comercial. Há uma preocupação maior na qualidade do serviço prestado(Appleby, Fakhouri et al. 2001) e menor no tempo de negociação do preço. A infra-estrutura necessária para esse tipo de serviço se torna consideravelmente maior(Buyya 2002), já que além dos componentes básicos de um sistema distribuídos, há a necessidade de desenvolver novas partes responsáveis por atividades como controle de pagamentos, leiloeiros ou até mesmo agentes reguladores.
- Alocação de Recursos(Kurose and Simha 1989 ; Parkes and Ungar 2001; Subramoniam, Maheswaran et al. 2002; Xue, Li et al. 2003): trata-se de

> aplicações pontuais; nessa modalidade computacional, os conceitos de economia são empregados para facilitar o escalonamento de tarefas entre dispositivos heterogêneos dispersos, como por exemplo, os recursos de um *Grid*. Apresentam problemas mais restritos e há preocupação com o desempenho geral do sistema e não raramente são feitas menções para sistemas com limitações temporais.

Sobre o escalonamento de tarefas a utilização de modelos econômicos provê diversas contribuições, Ferguson et al (Ferguson, Nikolaou et al. 1996) compara o modelo econômico com os sistemas computacionais distribuídos, um conjunto complexo de mecanismos é disposto ao modelo econômico com a finalidade de assegurar o controle descentralizado dos recursos, uma limitação dos atuais softwares de infra-estrutura *Grid* (Wolski, Plank et al. 2000), que não provêem mecanismos robustos e distribuídos para a alocação de recursos.

O item escalonamento sempre foi um dilema dentro de sistemas distribuídos, inclusive atualmente(Wolski, Plank et al. 2001), não existem políticas ou mecanismos para alocação e controle de recursos que podem assegurar alto desempenho e estabilidade nas configurações de um *Grid* computacional. Métodos de escalonamento distribuído são (Wolski, Plank et al. 2000) difíceis de manter, gerenciar e incertos em garantir o desempenho a medida que nodos são retirados ou incluídos.

Além desse fator, atualmente são disponíveis os mais diferentes tipos de computadores, diferenciando arquitetura, sistema operacional, capacidade dos recursos e vários outros fatores que tornam o sistema um ambiente heterogêneo e o projetista deve estar ciente de que seu programa precisa estar apto a executar nessas várias configurações de *hardware* e *software*.

Com a constatação desses problemas apresentados, a utilização de conceitos econômicos é vislumbrada como uma solução viável para ser utilizado em sistemas distribuídos, inicialmente pelo fato que os modelos econômicos já consolidados apresentam características semelhantes aos desejados num ambiente de computação distribuída, e ainda, é comum encontrar no mercado econômico, produtos e bens dos mais variados tipos e características sendo quantificado por um valor comum, no caso a moeda.

A utilização de modelos econômicos pode prover ainda (Kenyon and Cheliotis 2004): flexibilidade, eficiência, escalabilidade e *feedback* para as decisões de investimento:

- Flexibilidade: recursos podem ser obtidos quando o usuário necessitar e pode expressar sua necessidade com alto grau de detalhamento dos recursos necessários.
- Eficiência: o preço dos recursos reflete fatores como oferta e demanda, os quais podem representar a carga de um recurso.
- Escalabilidade: novas entidades consumidoras e fornecedoras de recursos podem ser inseridas e retiradas, mantendo a flexibilidade e eficiência do sistema.
- *Feedback*: preço para uso de recursos e o valor do recursos pode ser utilizado para guiar decisões de gerenciamento.

Outra vantagem que a utilização de modelos econômicos traz, é a transparência, um usuário não precisa saber exatamente como um serviço irá ser disponibilizado para ele, apenas será informado do preço para a execução, e isso vale para todos os serviços que um *Grid* pretende disponibilizar.

Essa similaridade tem impulsionado a utilização de princípios econômicos em pesquisas (Ferguson 1989 ; Wolski, Plank et al. 2000; Buyya 2002; Piro, Guarise et al. 2003; Lai, Rasmusson et al. 2005; Reddy 2006) para prover serviços computacionais para ambientes de produção onde seriam comercializados os recursos do *Grid,* a introdução de conceitos econômicos dentro de atividades computacionais, é vista com naturalidade, já que um simples parque de computadores interligados (Cotton 1975) pode ser visto como um sistema econômico em miniatura, onde as forças de oferta e demanda podem ser observados.

Cada uma dessas forças representadas por dois agentes (Cotton 1975 ; Ferguson, Nikolaou et al. 1996 ; Wolski, Plank et al. 2000; Buyya 2002; Buyya, Abramson et al. 2003) Consumidores (usuários dos recursos) e Fornecedores (proprietário dos recursos), e cada um com seus objetos diferentes. Os Consumidores (Ferguson, Nikolaou et al. 1996) procuram otimizar seus critérios de desempenho, pela obtenção dos recursos necessários sem preocupar-se com o desempenho de todo o sistema. Enquanto o

Fornecedor (Cotton 1975) tem o comportamento baseado em funções de produção (o relacionamento de fatores de entrada, como labor e capital, para produzir resultados de saída) e posição relativa no mercado.

Porém como já comentado anteriormente, agregar conceitos econômicos ao sistema distribuído é custoso, tanto devido a implementação de novos componentes ao sistema, como a demanda computacional extra, necessária para atender as novas funcionalidades da arquitetura. Para facilitar, Kenyon e Cheliotis (Kenyon and Cheliotis 2004) propõem um modelo composto de três pilhas, em cada pilha vários níveis, cada nível responsável por uma atividade dentro da arquitetura. O modelo de arquitetura pode ser visto na Figura 1:

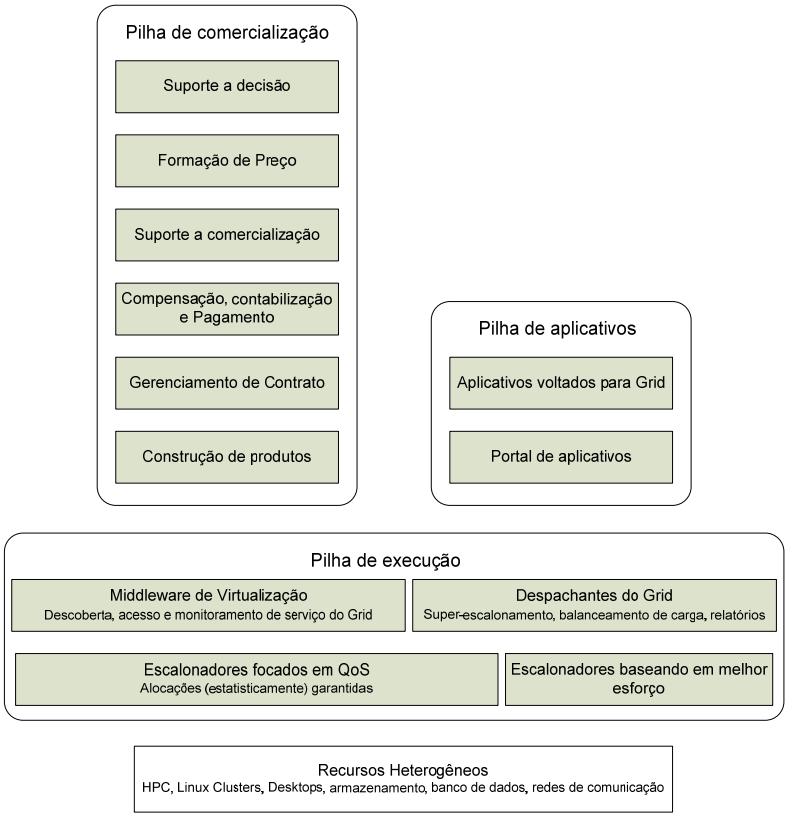

Figura 1 - Modelo de arquitetura de um *Grid* com enfoque econômico
Baseado em Kenyon e Cheliotis (Kenyon and Cheliotis 2004)

Analisando esse modelo estaremos focando nossa proposta na pilha de comercialização, mas precisamente na formação de preço.

Ainda a respeito do modelo apresentado por Kenyon e Cheliotis, verificamos que ela se refere apenas a infra-estrutura do nodo, não abrangendo a interação entre os diversos nodos que fazem parte de uma arquitetura. Para isso YEO e Buyya (Yeo and Buyya 2006) apresentam uma taxonomia sobre a arquitetura, dividida nos seguintes tópicos:

- Modelo de mercado
- Modelo de recursos
- Modelo de tarefas
- Modelo de alocação de recursos
- Modelo de avaliação

Porém para a elaboração desse trabalho estaremos focando apenas nos dois primeiros itens que são: modelo de Mercado e de Recursos.

2.2 - Modelos de mercado

Segundo Piro (Piro 2003) um modelo econômico não é determinado apenas por um conjunto de recursos e um conjunto de agentes (consumidores e produtores), mas também por um conjunto de regras que especificam a interação entre os recursos e os agentes.

A interação entre os envolvidos nesse processo pode se perfazer globalmente de dois princípios (Buyya, Abramson et al. 2002) (i) Através de conceitos de macroeconomia e (ii) de microeconomia. A macroeconomia (McConnell, Brue et al. 2003) procura examinar os princípios econômicos como um todo, ou através de subdivisões básicas ou agregadas, isto é, uma coleção de unidades econômicas tratadas como uma unidade única. Enquanto a microeconomia tem seu foco voltado para unidades econômicas específicas, estudando o comportamento de consumidores e fornecedores no âmbito local.

A partir dos princípios de microeconomia e macroeconomia, vários modelos foram propostos para o campo da computação, baseado no modelo derivado de Buyya (Buyya 2002), Shin and Buyya (Yeo and Buyya 2006), apresenta uma taxonomia dos modelos econômicos utilizados no ambiente computacional, conforme apresentado na Figura 2.

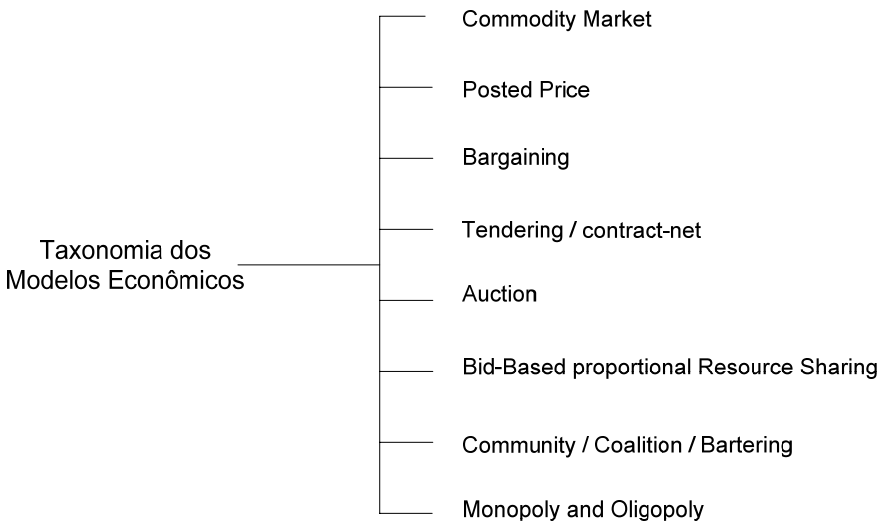

Figura 2 - Taxonomia dos Modelos Econômicos

Desses modelos, estaremos detalhando alguns deles, já que apresentam maior interesse para utilização em sistemas distribuídos: Mercado de commodities e Leilão.

2.2.1 - Mercado de commodities

Dentro de um mercado de commodities (Piro 2003), os bens e serviços disponíveis, ainda que de posse de diferentes proprietários, são consideradas como permutáveis, e os consumidores não compram um recurso em específico, mas simplesmente compram certa quantidade de commodities, que serão fornecidos por fornecedores não especificados durante a negociação. A política de preços que estabelece o valor da commodity pode ser (Buyya, Abramson et al. 2002) derivada de diversos parâmetros e pode ser dependente ou não de variações da oferta e demanda do recurso.

O esquema de preços em um mercado de commodities pode ser baseado (Buyya, Abramson et al. 2002) em:

- Taxa única
- Tempo de duração
- Oferta/demanda

- Subscrição

A definição do preço tendo em vista um único recurso não é uma prática útil dentro do mercado de commodities e praticamente merece atenção nos outros modelos de mercados utilizados para o escalonamento de tarefas em sistemas distribuídos, Piro (Piro 2003) considera que diferentes recursos computacionais, como processamento, armazenamento e rede, estão acoplados como um "recurso integrado unificado".

2.2.2 - Leilão

O modelo de leilão é um modelo de mercado que provê (Buyya, Abramson et al. 2003) uma negociação de um para muitos entre um provedor de serviço (vendedor) e muitos consumidores (compradores), chegando a um simples valor (preço).

A forma como uma oferta é feita em um leilão pode ser dada em (Klemperer 1999) dois aspectos, o leilão aberto é feito de forma que todos os participantes fiquem sabendo dos lances e podem ter uma visão na hora das negociações que estão ocorrendo, outra forma de oferta seria o leilão fechado, onde o leiloeiro recebe a oferta lacrada e os participantes não têm noção de como andam as ofertas.

Escolher uma dessas formas de leilão é definido em tempo de planejamento do leilão, tanto nos casos reais, quanto em processos computacionais, permitir que um leilão seja de forma aberta é uma forma mais transparente para os participantes, que podem estar ajustando seus preços a medida que conhecem o valor no momento do leilão, porém nesse tipo de leilão pode também comprometer o resultado esperado, a partir do momento que apenas um licitante está interessado no bem ou serviço, é óbvio que ele não irá concorrer com si mesmo em busca de uma melhor oferta para o leiloeiro

Em um leilão fechado, fica comprometido o poder de negociação, já que nenhum licitante concorrente conhece a oferta realizada, porém tem-se a possibilidade de garantir que o vencedor apresente a melhor oferta por livre opinião, podendo nesse caso garantir a melhor oferta do bem leiloado.

Quanto a forma como um leilão é realizado, podemos fazer a divisão em dois tipos (Klemperer 1999):

- Ascendentes

- Descendentes

No leilão ascendente, também conhecido por leilão Inglês, o preço é elevado seguidamente através de vários lances, até que reste somente um licitante interessado que proverá o maior valor. Esse tipo de leilão (Buyya, Abramson et al. 2003) pode ser realizado através do anúncio de preço pelo vendedor, através de chamadas de preços dos licitantes ou através de lances submetidos eletronicamente.

O tipo de leilão descendente, cuja origem (Klemperer 1999) é atribuída a vendedores de flores na Holanda, por isso também conhecido por leilão Holandês, é iniciado com um preço e sofre constantes reduções até que algum licitante aceite o valor.

Após a realização dos lances do leilão, é a hora que efetuar o pagamento do bem adquirido, nesse momento, os leilões possuem uma classificação quanto ao preço de fechamento (Klemperer 1999):

- Primeiro preço
- Segundo preço

Em um leilão com o primeiro preço de fechamento, o valor a ser pago é o melhor preço submetido, tanto num leilão ascendente, quanto num descendente. Enquanto num leilão de segundo preço, ganha o licitante que fizer o melhor lance, assim como no leilão de primeiro preço, porém, o valor a ser pago será o segundo melhor valor que foi dado lance. No leilão de segundo preço geralmente tem a natureza de ser um leilão fechado para evitar lances desleais.

2.3 - Modelo de Recurso

O modelo de recurso (Yeo and Buyya 2006) descreve as características de arquitetura de um sistema distribuído. Nesse modelo, os sistemas distribuídos são classificados conforme o controle de gerenciamento, a composição dos recursos, o suporte, a escalonamento e o mecanismo de contabilidade, conforme mostrado na Figura 3.

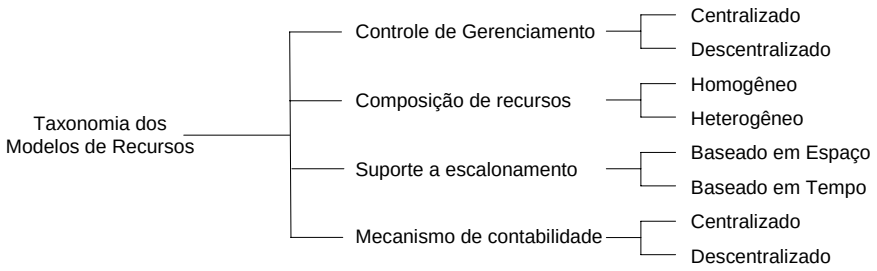

Figura 3 - Taxonomia dos Modelos de Recursos.
Fonte: YEO and Buyya (Yeo and Buyya 2006)

2.3.1 - Controle de Gerenciamento

O controle de gerenciamento descreve como um recurso é controlado e gerenciando em um sistema distribuído (Yeo and Buyya 2006). Um sistema com o controle de gerenciamento centralizado apresenta um ponto central que administra os recursos e as tarefas. Enquanto isso um sistema com controle de gerenciamento descentralizado tem mais de controle capaz de gerenciar subconjunto de recursos em um sistema distribuído.

Cada um apresenta suas vantagens e desvantagens, como é de se esperar, o controle de gerenciamento centralizado apresenta maior facilidade de implementação, porém é mais suscetível a gargalos e falhas devido a sobrecarga e mau funcionamento, em comparação ao controle de gerenciamento descentralizado, que apresenta um modelo com menores chances de paralisação geral do sistema, por possuir administradores que podem estar atuando para compensar falhas individuais no sistema.

Como ponto negativo, do controle de gerenciamento descentralizado está a dificuldade de implementação, consideravelmente maior que num controle centralizado, e ainda atenção especial deve ser desprendida quanto ao sincronismo entre os controle, o que exige protocolos e serviços de sincronização e demandam uma maior quantidade de dados trafegando via rede.

Conforme justificado por Yeo and Buyya (Yeo and Buyya 2006), o mais apropriado modelo para sistemas distribuídos baseado em modelos de mercado é o modelo descentralizado, por garantir melhor escalabilidade e confiabilidade

2.3.2 - Composição de recursos

Yeo and Buyya (Yeo and Buyya 2006) define a composição de recursos como a combinação dos recursos que são disponibilizados em um sistema distribuído. No caso de em um sistema distribuído, todos os nodos possuírem a mesma configuração e tipos de recursos, esse sistema é considerado homogêneo, esse tipo de sistema é direcionado para facilitar processamento paralelo que requerem o mesmo tipo de recurso a ser processado. A vantagem de ter uma composição homogênea de recursos é a simplicidade do código e a velocidade de execução em comparação ao sistema com recursos heterogêneos.

Em contrapartida (Yeo and Buyya 2006) o sistema com composição de recursos heterogêneos, que possuem diferentes composições de recursos e configurações, tem a vantagem de poder disponibilizar serviços que podem ser executados de forma concorrente.

A utilização de recursos heterogêneos em um sistema econômico, apresenta algumas dificuldades técnicas, principalmente no que se refere a definição de um preço comum para todos os recursos. Vários fatores influenciam em cada um dos recursos e conforme a utilização de cada um deles, a curva de variação de preços pode ser extremamente diferente. Esses recursos apresentam diferentes funções de utilidade e conseqüentemente, estimar uma função de custo único para todos os recursos tem sido um desafio para os pesquisadores que estão ligados a área.

2.3.3 - Suporte a escalonamento

O suporte a escalonamento determina (Yeo and Buyya 2006) o tipo de processamento que é suportado pelo sistema do Cluster. O escalonamento baseado em espaço permite que apenas uma simples tarefa seja executada em um período do processador, enquanto o escalonamento baseado em tempo suporta que múltiplas tarefas sejam executadas em um período do processador.

No escalonamento baseado em espaço, uma tarefa que for designada para executar em um nodo, irá utilizar todo o processamento desse nodo até que a tarefa tenha sido concluída, com isso, o tempo em que a tarefa demorará em ser executada irá ser menor,

já que não são necessários chaveamento das tarefas. As tarefas que chegarem posteriormente nesse nodo, deverão aguardar até que a tarefa que está executando libere o uso do processador, assumindo que essa abordagem, preempção não é suportada.

Em contrapartida, o escalonamento baseado em tempo, permite que tarefas sejam chaveadas e processadas concorrentemente no processador. A vantagem de se estar utilizando um sistema de escalonamento baseando em tempo é que uma tarefa pode ser alocada para utilizar o processador enquanto outra está esperando pela resposta de um dispositivo de entrada/saída. Com isso em um período de tempo maior (Yeo and Buyya 2006) tem uma maior vazão de tarefas sendo executadas.

2.3.4 - Mecanismo de Contabilidade

Os mecanismos de contabilidade (Yeo and Buyya 2006) servem para manter e armazenar informações sobre a execução de tarefas em um sistema distribuído. Essas informações podem ser utilizadas para decisões de mudança de planos de alocação no futuro.

Quanto a classificação, um mecanismo de contabilidade pode ser centralizado, quando um único nodo é responsável por manter os dados concentrados e armazenados. E um mecanismo descentralizado que provê múltiplos nodos monitorando e armazenando conjunto de informações.

Destacado por Yeo and Buyya (Yeo and Buyya 2006), ter um mecanismo de contabilidade centralizado garante que a recuperação dos dados seja mais simples, contudo é menos confiável e expansível comparado com um mecanismo de contabilidade descentralizado.

2.4 - Definição de Recursos

Arquitetura voltada a serviços têm sido considerada um importante paradigma para a disponibilização de softwares (Al-Ali, ShaikhAli et al. 2003), ainda mais presente em arquiteturas estruturadas em camadas (Broy 2003), como é o caso da arquitetura de *Grids* computacionais.

Para deixar claro, segundo Broy et al (Broy, Krüger et al. 2007), um serviço é um conjunto de funções providas por um software ou sistema (servidor) para um software ou sistema cliente, usualmente acessado através de uma interface de aplicação.

Aplicados esse conceito a um sistema de economia Grid, temos duas entidades que já foram discutidas anteriormente, porém agora contextualizadas dentre da definição de serviços:

- Fornecedores: são aqueles que são capazes de disponibilizar um conjunto de recursos, cada recurso disponibilizado apresenta uma demanda limitada e é delimitada por um conjunto de características inerente de cada recurso.
- Consumidores: são os interessados pela disponibilidade de um serviço, devem ser capazes de expressar suas necessidades e exigências, em troca da execução do serviço, recebem um custo, que pode ser financeiro, temporal ou baseado em alguma características computacional como consumo de energia (bateria no caso de dispositivos móveis).

Tanto fornecedores quanto consumidores são entidades independentes e a definição de qual papel, um dispositivo computacional fará parte, não exatamente precisa ser pré-definido no tempo de projeto. Por serem entidades dinâmicas, os papéis ainda podem se trocar durante a execução, um nodo que forneça certo serviço, pode requisitar que seja a execução de algo que ele necessite em outro nodo.

Percebe, nesse ponto que a base de toda a negociação se dá em nível de recurso. A identificação dos recursos necessários pelo cliente e a definição de custo de cada fornecedor é o que forma o mercado de negociação que será estabelecido entre ambos. Para consolidar essa definição Narayanan (Narayanan 2002) estabelece associações de um recurso *r,* com:

- Demanda instantânea $d_r(t)$, é a especificação de recurso que um usuário utilizará no espaço de tempo $t+\delta t$.
- Fornecimento instantâneo $s_r(t)$, é a capacidade de um fornecedor para prover recursos nesse mesmo período de tempo $t+\delta t$.
- Função de custo $c_r(t)$, o conjunto de recursos utilizados/providos acarreta em uma função de custo, esse custo pode ser características temporais,

como tempo para executar uma tarefa, características funcionais, como consumo de bateria, ou ainda com uma unidade de permuta universal, que seria a estipulação de uma moeda para a transação.

A relação de recursos utilizados pode ser uma função desempenhada pelos consumidores, que precisariam fazer um levantamento de suas necessidades baseados em características da tarefa que deseja executar. Ou também pode ser considerada uma tarefa do fornecedor (Yuan 2004), onde é mantido um histórico das execuções e baseado nesses valores, é realizada uma estimativa de quanto será necessário que seja utilizado de um determinado recurso para a execução da tarefa.

Cada tarefa, portanto, consistem em um conjunto de recurso *R*, desmembrados da solicitação da atividade e que irá influenciar no desempenho final do sistema, e da quantidade ω de cada recurso r_i que irá utilizar durante sua execução (Buyya, Abramson et al. 2003):

$$x = \{\omega r_1, \omega r_2, \ldots, \omega r_n\} \qquad (1)$$

Onde:

ω = representa a quantidade de recurso que será alocado

r_i = representa o recurso alocado.

Apesar de parecer uma tarefa trivial, especificar todos os possíveis recursos pode se tornar uma tarefa não computável (NP completo), devido a quantidade elevada de fatores que podem impactar na quantidade de cada recurso, e ainda a influência considerável do inter-relacionamento entre os recursos, como por exemplo, em um dispositivo móvel, a disponibilidade de ω quantidade de processamento poderá refletir no consumo de energia de forma acelerada.

Por isso, o grau de exatidão que se espera que seja fornecido do preço deve ser levado em consideração durante o projeto da infra-estrutura, para que, dependendo da situação, a estimativa dos recursos necessários poderá exigir maior quantidade de processamento que a execução do serviço em si.

2.4.1 - Taxonomia dos recursos

Segundo conceitos econômicos, os recursos naturais estão divididos em dois grandes grupos (Härting and Kofler 2006) os recursos renováveis e os não- renováveis. O primeiro se aplica a todos aqueles recursos que de uma forma ou outra podem ser regenerados, enquanto os não-renováveis são limitados pela quantidade de recursos atualmente disponíveis, isto é, são limitados pela reserva atual.

Em computação, Narayanan (Narayanan 2002) define uma analogia dos recursos que podem ser disponibilizados pelo computador com os recursos naturais:

Tabela 1 - Classificação dos recursos

Renováveis	Baseados em Tempo
	Baseados em Espaço
Não- Renováveis	Esgotáveis

2.4.1.1 - Recursos Baseados em Tempo

São recursos que possuem as características de temporalidade, isto é, apresenta duas condições distintas de utilização ao longo do tempo, utilização nula e utilização na sua totalidade disponível. São recursos que se encaixam nessa classificação a utilização do processador e da rede.

Portanto os recursos baseados em tempo apresentam definidos pela seguinte expressão:

$$s_r(t) = \begin{cases} R_t(t) & \textit{Se a aplicação utilizar o recurso} \\ 0 & \textit{Caso contrário} \end{cases} \qquad (2)$$

Onde:

$s_r(t)$ = quantidade recursos disponibilidade para o consumidor no tempo t.

$R_t(t)$ = total de recurso disponível no momento t.

A demanda por esse tipo de recurso pressupõe que no momento t, toda a capacidade do recurso será disponibilizada para o requisitante:

$$d_r(t) = \begin{cases} \infty \; se \; a \; aplicação \; utiliza \; o \; recurso \\ 0 \; caso \; contrário \end{cases} \qquad (3)$$

Segundo Narayanan (Narayanan 2002), o fornecimento de recursos baseados em tempo $R_r(t)$, desconsiderando o custo para chaveamento, pode ser aproximado com um modelo GPS (General Processor Share):

$$\sum_{todas \; aplicações} s_r(t) = R_r(t) \qquad (4)$$

Devemos lembrar que diversos fatores inerentes foram omitidos nos cálculos apresentados para simplificar a definição de custo, como por exemplo, a latência em uma transmissão de dados e tempos de esperar por E/S em um processador, porém na prática, valores elevadores desses parâmetros podem ocasionar distorção no resultado.

2.4.1.2 - Recursos baseados em espaço

Refere-se aos recursos que se utilizarão determinado espaço lógico em determinado espaço de tempo. Um exemplo clássico dessa classificação é o espaço em disco.

Segundo Narayanan (Narayanan 2002), há três tipos de objetos que poderão estar contido num espaço:

- Objetos permanentemente armazenados em disco: estão armazenados em disco e não serão utilizados para qualquer cálculo de estimativa de fornecimento ou demanda.
- Cópias cachê de objetos: estão originalmente armazenadas em servidores remotos e são copiadas para a máquina local para processamento.
- Objetos temporários: são criados para a execução de um serviço de depois da conclusão, são apagados.

A quantidade que pode ser solicitada será limitada pela capacidade de fornecimento.

A definição da capacidade disponível pelo fornecedor é definida por:

$$d_r(t) = \sum_{X \in temp(t)} size(X) \qquad (5)$$

Onde:

t*emp(t)* é o conjunto de objetos temporários pertencentes a uma aplicação no tempo *t*.

Por limitações de espaço, no caso de uma memória secundária e de desempenho em uma memória principal, Narayanan (Narayanan 2002), define o custo para a execução de um serviço baseado na equação (6):

$$c_r(t) = \begin{cases} \propto \ se\ d(t) > s(t) \\ s(t)\ caso\ contrário \end{cases} \qquad (6)$$

Onde:

s(t) é a disponibilidade do fornecedor de recursos.

d(t) é a demanda por esse tipo de recurso por parte do consumidor

Percebe-se que o custo para armazenar os dados é igual ao custo do fornecimento para o caso do fornecimento ser maior que a demanda, uma situação que exigiria um armazenamento de dados maior que a capacidade do dispositivo, por exemplo, seria uma situação que não poderia ser atendida.

2.4.1.3 - Recursos Esgotáveis

E por fim, são os recursos por assim considerar não-renováveis, sob a óptica da computação, um recurso típico que se encaixa nessa classificação é a bateria, existe uma quantidade limitada de energia que o dispositivo pode estar requerendo, após esse prazo, é necessária a troca ou recarga da bateria.

Segundo Narayanan (Narayanan 2002), para caracterizar o fornecimento de energia é necessário informações sobre o nível de recurso $E_r(t)$, caso o dispositivo esteja sendo carregado no momento, teremos a taxa de recarga $R_r(t)$, e o nível possível de

recurso $M_r(t)$, que seria a capacidade de fornecimento de energia. Ainda deve ser considerado que o recurso pode apresentar um limitador $A_r(t)$, no caso da bateria, o máximo que um cliente pode drenar de corrente elétrica.

Com esses parâmetros, o fornecimento de recursos esgotáveis é definido:

$$s_r(t) = (E_r(t),\ R_r(t),\ M_r(t),\ A_r(t)) \qquad (7)$$

2.4.2 - Recursos relevantes

Com a classificação dos recursos computacionais, podemos definir uma lista de recursos que seriam interessantes que fossem tratados no nível de formação de preço, dentro esses recursos Narayanan (Narayanan 2002), define:

Tabela 2 - Classificação de alguns recursos

Recurso	Classificação	Unidade de demanda	Unidade de fornecimento
CPU	Baseado em tempo	Ciclos	Ciclos/s
Energia	Esgotável	Joules	Joules
Taxa de transmissão de rede	Baseado em tempo	Bytes	Bytes/s
Taxa de recepção de rede	Baseado em tempo	Bytes	Bytes/s
Memória física	Baseado em espaço	Bytes	Bytes
Leitura em Disco	Baseado em tempo	Bytes	Bytes/s
Gravação em Disco	Baseado em tempo	Bytes	Bytes/s
Espaço em Disco	Baseado em espaço	Bytes	Bytes

Devemos destacar que os recursos apresentados na Tabela 2 se referem a recursos baseados em características físicas dos dispositivos computacionais, num ambiente de *Grid* podem ser abordadas outras características baseadas em características lógicas, como métricas de software.

Ainda ressaltamos que a lista de recursos não se esgota nessa tabela, devido a grande quantidade de serviços que podem estar envolvidos num ambiente de *Grid* a heterogeneidade de dispositivos envolvidos e a criação de novas tecnologias, pode se fazer necessário a classificação de novos recursos.

Para exemplificar essa situação, imaginamos um serviço de impressão, uma impressão pode possuir características, como demora de impressão, que é um recurso baseado em tempo e a quantidade de tinta utilizada que é um recurso esgotável.

2.4.3 - Combinando Recursos

A possibilidade de que um serviço requeira apenas um tipo de recurso é remota, em grande parte das requisições, o serviço irá utilizar de vários recursos de forma conjugada, como por exemplo, processamento e comunicação de dados através da rede de dados.

Com isso, podemos especificar os recursos necessários para um serviço através da declaração dos recursos utilizados $r_1(T), r_2(T), \dots, r_n(T)$

Para cada recurso que fará parte da solicitação, deverá ser especificado um valor de demanda, que seria a quantidade de recurso utilizado, esse quantificador é uma tentativa de expressar a real quantidade de recursos que um serviço irá necessitar. Aplicando esse valor ao conjunto de recursos utilizados obteremos a quantidade de recursos que um serviço irá necessitar

$$d_n(T) = \omega R_n(T) \qquad (8)$$

Onde:

ω = representa a quantidade de recurso *n* necessário durante o tempo de execução *T*

Até esse ponto, obtemos um vetor de recursos utilizados pelo serviço, representado por $\overline{D}$, contendo a relação dos recursos e a suas respectivas demandas. Esse vetor é criado pelo consumidor, no momento que necessita do fornecimento de um serviço, ou caso o fornecedor, tenha essa capacidade, ela decompõe a solicitação e estipula os devidos valores.

Não faz parte desse trabalho, detalhar o processo como ocorre o processo de definição da demanda pelos consumidores, já que envolve técnicas de predição de demanda que fugiria do escopo.

Criada a devida lista $\overline{D}$, o nodo fornecedor deverá montar a relação de preços para os recursos, representados pelo vetor $\bar{P}$. Cada recurso é representado dentro do vetor por p sendo que:

$$\forall p \in \bar{P}, p \geq 0 \qquad (9)$$

Com isso, a função de custo para um serviço C_S descrito na equação (10) sintetiza o custo como sendo a somatória dos preços individuais de todos os recursos pela sua quantidade de recurso necessária.

$$C_s = \sum_i^n D_i p_i \qquad (10)$$

Por fim, ressaltamos que essa é uma equação genérica para a função de utilidade, já que os valores de demanda e preço podem sofrer alterações durante a execução, para evitar variações abruptas no valor do preço estaremos demonstrando posteriormente uma função de suavização, que permite minimizar variações temporárias do preço.

Para a demanda seria em função da grande quantidade de fatores que pode influenciar nos valores pré-estabelecidos, para esse caso, seria interessante manter um histórico das execuções e baseado nesses valores estipular um desvio padrão para os valores solicitados pelo cliente.

3 - GRID-M E ECONOMIA

Para a elaboração desse trabalho, utilizamos o sistema de Grid-M, um ambiente de *Grid* desenvolvido no Laboratório de Redes e Gerência (LRG 2007) da Universidade Federal de Santa Catarina. O sistema foi utilizado pela familiaridade com o sistema e a disponibilidade do código fonte, que permite que sejam feitas modificações em sua estrutura de forma simples. O código fonte pode ser encontrado no site do projeto (Grid-M 2007), a descrição e a justificativa do projeto base foram apresentadas por Rolim (Rolim 2007) em sua dissertação de mestrado.

A princípio, o Grid-M é uma infra-estrutura centrada na utilização de dispositivos móveis (Rolim 2007), porém devido a sua flexibilidade, o ambiente pode ser executado em qualquer meio computacional que suporte a tecnologia empregada no desenvolvimento. Nesse capítulo faremos uma descrição do ambiente utilizado, para que possamos depois apresentar as alterações realizadas que permitem ao Grid-M dispor do serviço de economia.

3.1 - O Grid-M

O Grid-M foi desenvolvido com enfoque em dispositivos móveis e nômades, que apresentam limitações de desempenho, geralmente dispositivos alimentados por bateria e com características de mobilidade presente em seu projeto. Esses tipos de dispositivos possuem tendência de crescimento em poder computacional, recursos e utilização que permitem prever que num futuro próximo, dispositivos móveis terão grande utilização no nosso dia-a-dia (McKnight, Anius et al. 2002; McKnight, Howison et al. 2004; Wang, Yu et al. 2005; Ahuja and Myers 2006)

Partindo desse principio, Rolim (Rolim 2007) apresenta o Grid-M, uma infra-estrutura de *Grid* e um conjunto de API que permitem o desenvolvimento de um sistema para a distribuição de tarefas entre diferentes nodos participantes de uma rede de computadores.

O sistema é orientado a serviços, sendo composto por um conjunto de serviços que são utilizados para as funções básicas do sistema e uma interface que permite a

criação de novos serviços baseados nessa interface, ao quais serão mais bem detalhados em breve.

Ao inicializar o sistema os nodos vizinhos são devidamente registrados em uma base dados local, esse sistema de repositório é atualizado através de mecanismos de anunciação. Quando uma tarefa é solicitada, ocorre o processo de busca que faz uma estimativa dos nodos disponíveis e busca por um nodo que possa executar a tarefa solicitada.

Em cada nodo, a base de dados local é composta por um conjunto de nodos vizinhos e seus serviços, a partir do momento que um nodo precisa executar uma atividade em um nodo remoto, essa tabela é verificada e são localizados os nodos fornecedores que tem esse serviço disponível.

Cada nodo fornecedor é composto por um conjunto de serviços $S=\{s_1, s_2, ..., s_n\}$, e cada serviço s_i dentro desse conjunto possui um comportamento dado pela função (11):

$$F : I \rightarrow \rho(O) \qquad (11)$$

Essa função nada mais é do que a representação de uma interface com um comportamento pré-estabelecido de uma função que o nodo irá executar, conforme demonstrado graficamente na Figura 4.

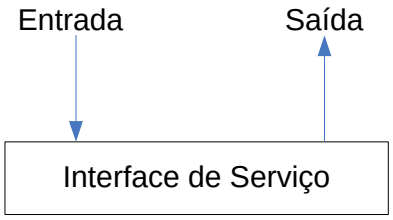

Figura 4 - Interface de serviço

Todos os nodos que estão presentes no sistema de *Grid* e disponibilizam algum tipo de serviço, apresentam pelo menos uma interface de serviço, para esse caso, o nodo contém um conjunto de serviços ou domínio, que define as atividades que ele pode estar realizando, nesse caso, o domínio do nodo é representado por (12).

$$Dom(n) = \{S_n \mid S_n \neq \{\emptyset\}\, e\, (\forall s)(s \in S_n \rightarrow F(s) \neq 0)\} \qquad (12)$$

Onde S_n é o conjunto de serviços prestados pelo nodo n, esse conjunto deve possuir serviços que sejam executados, parte-se do princípio que todo o serviço s contido no conjunto de tarefas S produz algum tipo de resultado.

Quanto ao sistema Grid-M, todo o dispositivo que queira fazer parte da infra-estrutura deve implementar a classe *Node*, ela é composta por um conjunto básico de serviços que são utilizados para as funções essenciais do sistema, portanto o *Dom()* de uma instância da classe *Node*, é composto inicialmente pelos seguintes serviços:

- ***Stop***: interrompe a execução dos serviços no nodo.
- ***Get-statistics***: retorna resultados estatísticos da execução do nodo.
- ***Ping***: serviço utilizado apenas para retornar uma mensagem indicando que o nodo está ativo.
- ***Register***: registrado os nodos que fazem parte da estrutura, é necessários que todos os nodos enviem uma requisição desse serviço, alertando aos outros nodos de sua existência na rede.
- ***Get-delayed***: tarefas podem ser temporizadas para executar posteriormente, essa mensagem verifica se a tarefa temporizada já concluiu, caso positivo retorna o resultado, caso negativo retorna uma mensagem de que o serviço ainda está ocupado processando.
- ***Route-table***: recebe uma lista de nodo e endereços e atualiza a base local com esses dados.
- ***Add-route***: adiciona um nodo e seu caminho na base local de roteamento.
- ***Set-default-route***: define a rota padrão.
- ***Discovery***: verifica se o nodo implementa um serviço em específico.
- ***Check-attached-services***: retorna os serviços disponíveis no nodo.
- ***Check-attached-sensors***: retorna os sensores disponíveis no nodo.

Conforme detalhado no início do capítulo, o domínio de serviços de um nodo pode ser estendido, recebendo novos serviços, conforme pode ser verificado na Figura 5, a classe *Node* dispõe do método *addService()*, esse método apresenta como argumento *serviceName* e *service*¸ a variável *serviceName* é do tipo String e é utilizada

para definir o nome de como o serviço irá ser localizado, enquanto a variável *service* é uma instância da classe *Service* e contêm os métodos necessários para a devida execução do serviço. Essas duas classes formam a base do Grid-M, e podem ser obtidos maiores detalhes de suas implementação no site do Grid-M (Grid-M 2007).

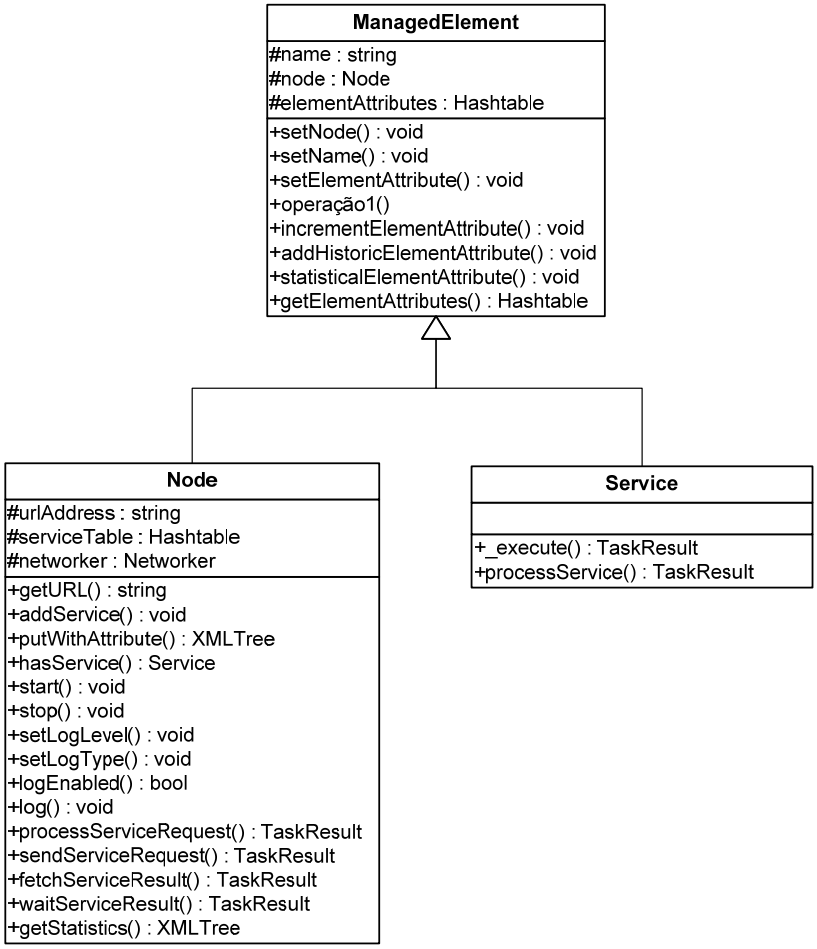

Figura 5 - Classes *Node* e *Service*

Depois de criada a instância da classe *Node* e adicionado os serviços necessários, o agente anuncia sua criação na infra-estrutura e passa a fazer parte do ambiente, interagindo com outros agentes na mesma situação que ele.

Para exemplificar a extensão de um serviço, a Figura 6 faz uma demonstração de um nodo com a capacidade de prover o serviço *MathService*, nessa figura, percebemos a instanciação da classe *Node* na primeira linha, onde é chamado o construtor *Node()* que recebe duas *Strings* como argumentos, a primeira se refere ao nome como o nodo vai ser

conhecido dentro da infra-estrutura, por conveniência, nos exemplos está se utilizando nomes simples como “node-1”, esse nome deve ser único dentro da infra-estrutura, pois será através dele que o nodo será localizado. Numa estrutura com grande quantidade de agentes, fica inviável fornecer nomes de forma manual para todos os nodos, para isso pode utilizar valores como o número IP ou o nome do computador, como foi o caso da implementação da proposta desse trabalho, onde foi utilizado a função *java.net.InetAddress.getHostName()*.

```
    Node node1 = new Node("node-1", "http://localhost:8001"); //
Instancia o nodo no endereço localhost, porta 8001

    NetInterface httpNetInterface = new HTTPServerInterface(node1,
8001); // Cria o servidor HTTP

    node1.setNetInterface(httpNetInterface); // Seta o servidor HTTP
para esse nodo

    node1.addService("multiplicar", new MathService()); // Adiciona o
serviço de multiplicação através da instanciação da classe
MathService()

    node1.start(); // Inicia a execução do node1
```

Figura 6 - Exemplo de extensão de um serviço

Na figura anterior foi visto a extensão do nodo, adicionando o serviço *MathService()*, essa classe é apresentada na Figura 7, a classe não está totalmente implementada e constitui-se apenas do método responsável pela execução do serviço, definido como *_execute*, nesse método os parâmetros passados pela função sãos objetos do tipo *Node* e *Task*. Esses parâmetros enviam informações sobre o nodo que disparou a execução do serviço e as informações sobre a execução da tarefa, como por exemplo, os dados que o fornecedor deve considerar e o tipo do serviço.

```
protected TaskResult _execute(Node nodo, Task tarefa) {
    long resultado = 0;
    ArrayList ListaDeValores = new ArrayList();

    XMLTree parametros = task.recebeParametros(); // Recebe os
parâmetros que foram enviando na tarefa
    while (parametros)
      Decomponha a Relação em parâmetros em uma Lista

    if (tarefa.serviço == "multiplicar”) { // Verifica o tipo do
serviço
      while(existir parametros)
            multiplique os parametros
    }
      Retorne o valor calculado
  }
```

Figura 7 - Serviço Multiplicar

Ainda podemos observar na Figura 7, que os parâmetros da tarefa são transformados numa lista, que posteriormente serão multiplicados, conforme equação (13). O produto é convertido novamente em um parâmetro, dessa vez de uma instância da classe *TaskResult* e retornará essa instância para o nodo que disparou a execução do serviço.

$$Resultado = \prod_{valor}^{parâmetros} valor \qquad (13)$$

Conforme a extensão do domínio, cada nodo pode ter serviços diferenciados de outros nodos. Porém a função de solicitação do serviço não é diferenciada para cada serviço, dentro do Grid-M a solicitação de execução da tarefa *T(y)* demonstrada por um tupla que permite a interação entre o requisitante e o executor, esse conjunto de dados é demonstrado na expressão (14):

$$T(y) = \{n_i, s_i, P\} \qquad (14)$$

Onde:

n_i = é o nodo que irá executar a tarefa, sendo que $n_i \in N$

s_i = nome do serviço que o nodo local solicita que seja executado, sendo que o serviço s_i deve fazer parte do $Dom(n_i)$

P = é o conjunto de parâmetros previamente definidos em nível de projeto para cada tipo de serviço *s* e utilizado para definir condições e métricas de execução do serviço.

Em forma de código de programação a tupla que é utilizada para a definição de uma tarefa é apresentada na **Erro! Fonte de referência não encontrada.**

```
Task task = new Task("node-1", "multiplicar", Parameters);
```

Figura 8 - Tupla de uma tarefa

O conjunto de parâmetros é a forma como um serviço tem sua definição repassada para o fornecedor, garantindo que o serviço seja executado conforme interesse do cliente. A Figura 9 demonstra a definição de um conjunto de parâmetros, percebe que para esse serviço, um grupo de valores que serão multiplicado são enviados para o nodo executor.

```
XMLTree parametros = new XMLTree();// Cria a árvore de parâmetros
parametros.add("value", "10"); // Adiciona os parâmetros.
parametros.add("value", "2");
parametros.add("value", "4");
parametros.add("value", "5");
```

Figura 9 – Parâmetros e Solicitação de execução de uma tarefa

O conjunto de parâmetros *P* é representado por um mapa com suas chaves e valores convertido no padrão de mensagens definida pela Foundation for Intelligent Physical Agents (FIPA) ACL (Agents 2001) e pode ser observado na Figura 10.

```
<Parameters>
      <value>10</value>
      <value>2</value>
      <value>4</value>
      <value>5</value>
</Parameters>
```

Figura 10 - Parâmetros de uma tarefa

Após definido o conjunto de parâmetros, esses farão parte da tupla responsável por definir quem, qual serviço e que parâmetros são utilizados para o processamento,

restando ao nodo apenas enviar a tarefa e aguardar o resultado da execução. A Figura 11 demonstra o envio da tarefa e o aguardo do retorno através da variável *result*.

```
// Executa a tarefa
TaskResult result = node1.sendServiceRequest(task);
```

Figura 11 - Envio da execução da tarefa

O envio da tarefa no Grid-M pode ser feita de duas formas, a primeira delas é através da definição direta do nodo que irá receber a tarefa, a segunda é deixado a cargo de o sistema decidir, especificando o campo do nodo através do símbolo * (asterisco), dessa forma a tarefa será encaminhada para o primeiro nodo que estiver apto a executar-la.

Dentro dos experimentos, foi utilizada a especificação do nodo diretamente, isso para garantirmos que a distribuição das tarefas corresponderia ao que o sistema de escalonamento tinha decidido em tempo de execução.

Após o recebimento de uma tarefa, a função do nodo executor é aplicar a equação (1) ao conjunto de dados da entrada (3) e gerar uma saída $\rho(O)$ adequada ao esperado. A função $\rho(O)$ nada mais é do que um serviço s do conjunto S que pode ser prestado pelo nodo, essa função representa a transformação dos dados de entrada I em dados de saída O e a real definição dessa função dependerá do contexto de cada nodo.

A saída de dados, assim como a entrada, deve seguir um padrão, portanto a saída O dentro do Grid-M é chamada *TaskResult* e tem a sintaxe muito semelhante a apresentada pela entrada de dados, a equação (15) apresenta o conjunto de dados retornados pelo nodo em resposta de uma tarefa.

$$Tr(z) = \{T, r, P\} \qquad (15)$$

Onde:

$Tr(z)$ = resposta de uma tarefa executada pelo nodo z

T = tarefa original que ocasionou a execução da tarefa.

r = código de retorno.

P = parâmetros, assim como no envio de execução, pode ocorrer o retorno de informações via corpo do resultado, para isso esse campo é utilizado, com o mesmo formato encontrando na solicitação da Tarefa.

O campo de resposta *r* apresenta uma condição em que o nodo foi executado, assim como numa resposta do protocolo HTTP (W3C 2007), o código de retorno apresenta através de um valor decimal a condição em que a tarefa esteve durante a sua execução.

Dentro da classe de serviço *MathService()*, após o processamento da tarefa, a tupla de retorno, receba os argumentos conforme demonstrada na equação (15) e implementados dentro do Grid-M conforme exemplificado na Figura 12.

```
      TaskResult retorno = new TaskResult(tarefa, RESULT_OK, null);
// Cria uma tarefa resultado que será retornada ao solicitante

      retorno.adicionaParametros("result", "" + resultado); //
Adiciona o resultado

      return retorno; // Retorna a instancia de TaskResult
```

Figura 12 - Retorno do resultado

Definida pela função *ρ(O)*, a execução da tarefa é uma parte importante do Grid-M, pois é através dela que são estendidas as funcionalidades básicas do sistema. Partindo do pressuposto que todas as demais funcionalidades do sistema são extensíveis, é necessário definir uma interface padrão que guie o desenvolvimento de todas as novas funcionalidades. A partir da próxima sessão estaremos abordando as extensões realizadas na infra-estrutura do Grid-M é que são necessários para a proposta desse projeto.

3.2 - Grid-M Economy

Para um sistema de *Grid* os dispositivos envolvidos são denominados nodos ou nós e sua finalidade é fornecer serviços para os solicitantes que requisitaram uma determinada quantidade de recursos. Em um sistema econômico computacional o processo é o mesmo, porém com diferença que os nodos que executam os serviços são

denominados fornecedores e os solicitantes consumidores, isso em virtude da utilização desses termos no modelo econômico tradicional.

A analogia é a mesma tanto em um sistema de *Grid* quanto em um sistema econômico, o fornecedor é detentor de um conjunto de recursos r_i, representados pelo conjunto R e detalhado na equação (16):

$$R = \{r_1, r_2, \dots, r_n\} \qquad (16)$$

Cada recurso disponibilizado pelo fornecedor apresenta características distintas, isso é percebível quando exemplificamos que os recursos podem ser memória, processador, taxas de transmissão de dados e outros recursos presentes em um dispositivo computacional, sabemos que cada um deles apresenta diferentes unidades de medição, como MB (megabytes) para memória, ciclos de processamento para o processador, Mbps (mega bits per second) para a transmissão de dados no sentido de envio ou recebimento, porém cada um apresenta um quantificador de quanto recurso é disponibilizado pelo fornecedor.

O conjunto de quantificadores é utilizado em dois sentidos, pelo fornecedor esses valores são indicadores de reserva de recursos e também como indicador do preço final que será cobrado do consumidor, a equação (17) demonstra o conjunto de quantificadores.

$$W = \{\omega_1, \omega_{2,} \dots, \omega_n\} \qquad (17)$$

Entre o conjunto de recursos e o conjunto de quantificadores, há uma relação de bijeção, isto é, deve existir uma relação única entre os valores do conjunto de recursos com os valores do conjunto dos quantificadores. Da relação entre eles obtemos a função de utilidade, que é definida por McConnell et al (McConnell, Brue et al. 2003), como o grau de satisfação ou prazer disponibilizado para o usuário, essa função atribui um conjunto de preços para os recursos, demonstrado na equação (18):

$$P = \{p_1, p_2, \dots, p_n\} \qquad (18)$$

Baseado nesses valores, a formulação do preço do serviço, a partir desse ponto pode ser facilmente calculada, somando o custo individual de cada recurso, conforme mostrado na equação (19):

$$P = \sum_i p_i \omega_i \qquad (19)$$

A simplicidade de definir uma fórmula, não necessariamente representa a simplicidade de se definir o custo de um recurso. O fator da demanda de um recurso representa a capacidade de disponibilidade desse recurso para o sistema, o mercado tem a função de atribuir o devido valor ao recurso baseado nesse fator e o consumidor vai influenciar na flutuação do preço baseando na sua requisição pelo serviço.

Para a elaboração desse trabalho não nos preocupamos com a interdependência entre os recursos, porém ela existe e a influência que ela pode apresentar ao dispositivo pode ser considerável. Em seu trabalho, Narayanan (Narayanan 2002) apresenta o gráfico de relacionamento entre os mais comuns recursos disponíveis em dispositivos móveis, esse gráfico, por motivo de análise é reproduzido na Figura 13.

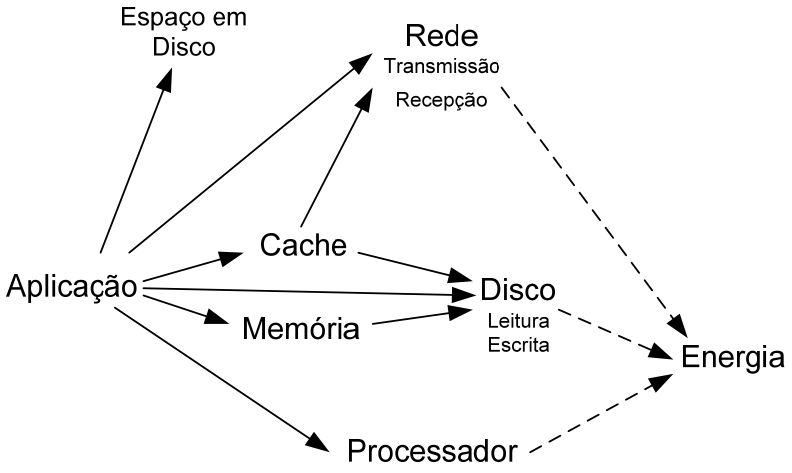

Figura 13 - Dependência entre recursos
Baseado em: Narayanan (Narayanan 2002)

A influência que cada um dos recursos sobre os demais dependentes pode ocasionar a sobreposição de custos sobre um recurso em particular, como pode acontecer com uma requisição de processamento, numa situação puramente teórica, a

tarefa apenas utilizaria processamento e desse processamento seria consumido uma determinada quantidade de energia, porém na prática isso não é o que se observa, uma requisição de processamento requer consumo de memória, leitura e escrita em disco e pode acontecer à interação com outros dispositivos, aliado a isso, uma aplicação real demanda um consumo considerável de energia elétrica, o que se trata de uma utilidade e cobra um preço pelo consumo.

Pela complexidade acima demonstrada, preferimos desconsiderar a dependência entre os recursos presentes em um dispositivo computacional, porém como será comentando nas considerações finais, esse desafio será deixado como uma sugestão de trabalhos futuros.

3.3 - Funções de utilidade

Para apresentar o valor final do custo de um serviço, a equação (19) descreve como sendo a somatória dos valores individuais de cada recurso. O desafio de definir um custo para cada serviço é ponderar o peso que cada recurso tem no preço final.

Conforme demonstrado no capítulo 2, os recursos foram classificados em dois tipos, os renováveis e os esgotáveis, cada um com funcionalidades parecidas, porém para cada um dos recursos de cada grupo, é necessário um estudo próprio.

Devido a limitações temporais, nesse trabalho estaremos apenas demonstrando a utilização do processador para definir o custo, de forma que durantes os experimentos buscamos realizar a execução de tarefas que exigissem uma grande quantidade de processamento e que não tivesse impacto na utilização de outros recursos, como por exemplo, acesso a memória principal ou a memória secundária.

Apesar disso, há a preocupação em definir um ambiente extensível, que permitisse agregar a função de utilidade de outros recursos. Para isso foi definido uma interface que deverá ser utilizada nas classes que queiram ser integradas ao sistema. A Figura 14 demonstra essa classe e os campos que são obrigatórios que sejam implementados. Vale ressaltar que no contexto de compilação, o sistema faz a verificação das classes que implementam a função de utilidade se essas apresentam rigorosamente os campos da interface *Utilities*.

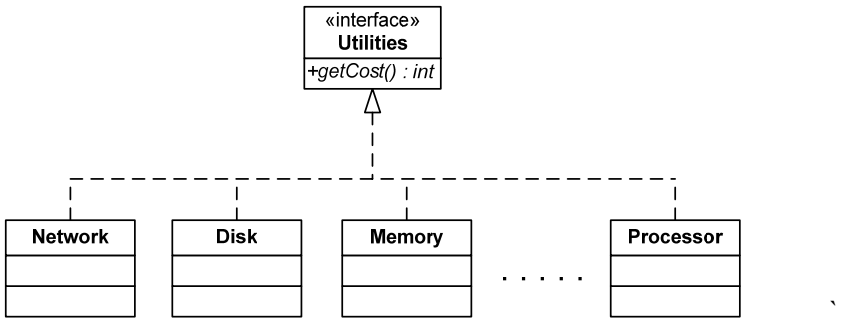

Figura 14 – Interface Utilities

Para calcular o custo do processador, foi implementado uma função de utilidade baseada em Ferguson (Ferguson 1989), essa equação, demonstrada no (20), foi adaptada para representar a heterogeneidade que existe entre os diferentes tipos de processadores.

$$price_{processor} = p_0 + k \times Load \qquad (20)$$

Onde:

k = constante que define a amplitude da variação do preço inicial em razão da carga do recurso

$Load$ = Carga do recurso.

p_0 = preço inicial

A função da carga do recurso representa as variações de utilização do recurso que está sendo monitorado. A carga é representada pela equação (21):

$$Load = \left(\frac{Resource_{load} - T}{T}\right) \qquad (21)$$

O valor de T irá influenciar o valor central em que a carga do recurso irá oscilar, essa variação será em função da variação de carga do recurso e os valores irão se situar entre 1,0 e -1,0. Essa variação será amplificada pela constante K, o que será responsável por definir a variação do preço inicial do recurso.

Para exemplificar essa variação, a Figura 15 simula diferentes índices de carga e a influência que esses valores ocasionam na variação da carga, o valor definido de *T* é 50, esse valor será utilizado posteriormente nas simulações e ele foi escolhido por representar o ponto central do recurso.

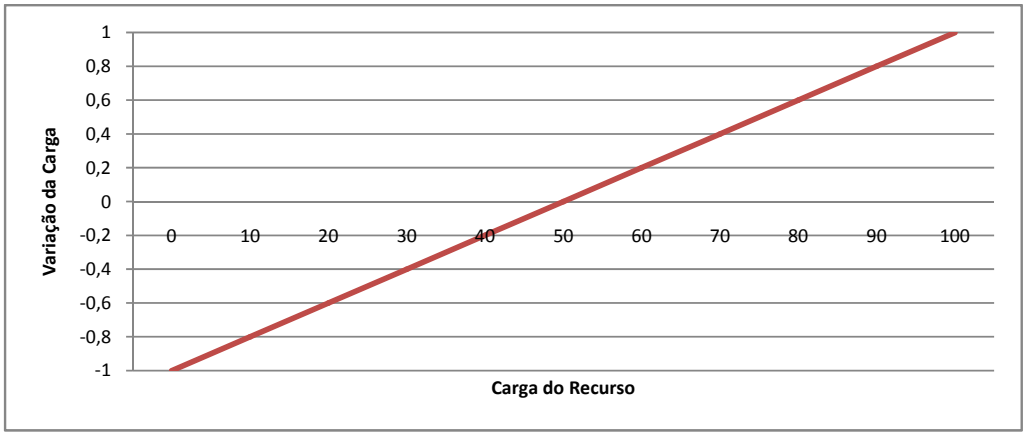

Figura 15 - Variação da carga do recurso

Para o preço inicial é proposto uma variação dos conceitos apresentados por Ferguson (Ferguson 1989) e Piro (Piro, Guarise et al. 2003), que estipulam um valor escolhido numa faixa de valores. Nesse trabalho será utilizada uma estrutura que provê a seleção dos valores iniciais baseado em índices de desempenhos bastante conhecidos, os índices da SPEC (SPEC 2007), um consórcio de empresa, sem fim lucrativo, que tem como objetivo criar um *benchmark* que represente o desempenho de sistema com transparência e imparcialidade.

Para a utilização no nosso sistema, foram selecionados diversos *benchmarks* e agrupados em um banco de dados, no momento que um nodo fornecedor era executado, sua primeira atividade era enviar uma requisição para os nodos agentes que contêm a base de benchmark e selecionar o índice que melhor representa o desempenho do processador.

Num sistema mais amplo pode-se estar utilizado esse valor como uma variável macroeconômica, refletindo a variação do preço inicial em função de perspectiva da evolução tecnológica, alterando o preço conforme a chegada de novos processadores e desatualizações do encontrados no momento.

3.3.1 - Processo de leilão

Devidamente atribuído os preços para o serviço, os nodos fornecedores ficam aguardando até que a requisição de um orçamento seja feito. A requisição de um serviço é disparada pelo cliente, nesse momento ocorre o processo de leilão holandês, porém os lances não são publicados abertamente para todos os licitantes por uma questão de tráfego de rede, que poderia sobrecarregar o sistema.

Nesse processo de leilão, conforme pode ser observado na Figura 16, inicia-se com a consulta do consumidor (i), que envia uma requisição para uma lista de fornecedor que tem cadastrado em sua tabela local de nodos, através do processo de formação de preço descrito anteriormente, apresenta (ii) sua proposta ao consumidor, o consumidor armazena todas as propostas retornadas e escolhe a menor proposta de custo (iii). O próximo passo é enviar a tarefa para o nodo vencedor que terá que executar a tarefa acordada.

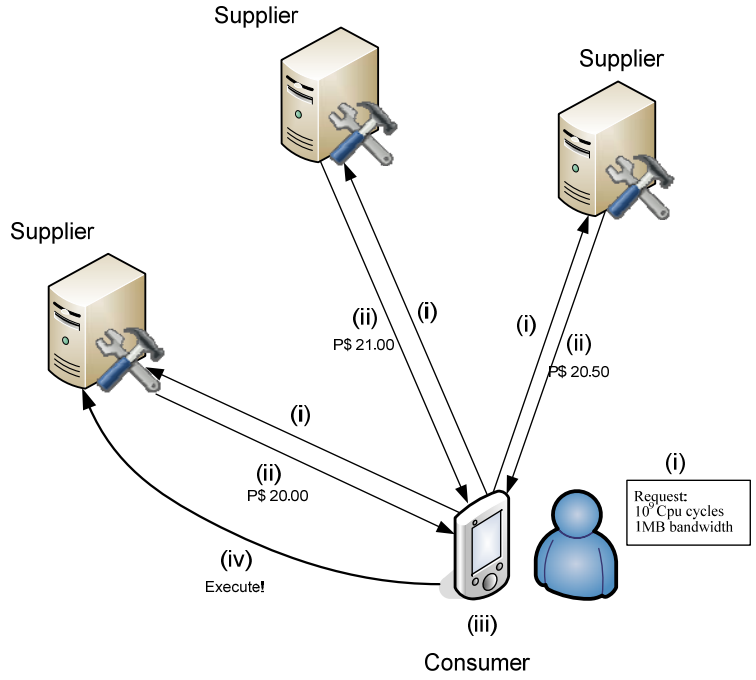

Figura 16 - Processo de leilão

A possibilidade de requisitos que são enviados para os fornecedores é grande, até agora comentamos apenas em recursos, porém nada impede que outras métricas sejam utilizadas para fazer a estimativa de custo para um serviço, Moorsel (van Moorsel

2001), por exemplo demonstra os conceitos de QoE e QoBiz, que são métricas de qualidades referente a Experiência e a Qualidade de Negócio.

Juntando esses conceitos, ao já conhecido termo QoS, pode-se oferecer uma estimativa de custo baseada também em indicadores de qualidade, aproximando a estimativa de custo a um serviço real, onde além do produto, o usuário tem a possibilidade de escolher a qualidade desejável. Porém a possibilidade de utilização desses conceitos é pertinente para um trabalho futuro dentro de sistemas de economia *Grid*.

4 - MODELO DE ESCALONAMENTO PROPOSTO

O desenvolvimento da proposta, conforme mencionado é baseada no Grid-M, a estrutura do Grid-M é detalhada no trabalho apresentado por Rolim (Rolim 2007). A partir da base desse sistema serão apresentadas modificações e expansões necessárias para o funcionamento da proposta desse trabalho.

Agentes são responsáveis por desempenhar atividades específicas dentro do sistema, esses agentes foram agrupados em três grupos, graficamente representado na Figura 17, o primeiro é grupo de agentes que é encarregado de definir o preço inicial para o segundo grupo de agentes, que são os Fornecedores, os responsáveis pela execução das tarefas solicitadas pelo terceiro grupo de nodos, os consumidores.

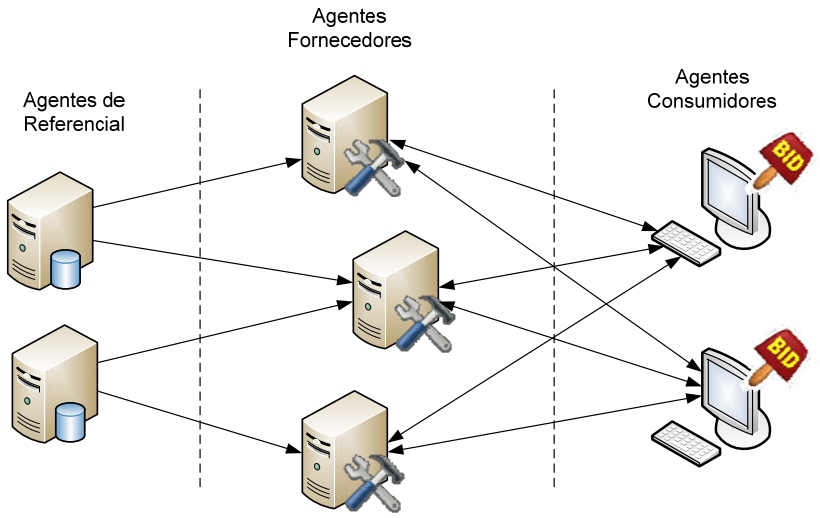

Figura 17 - Diagrama da plataforma computacional

Cada grupo de agentes serão detalhados nos próximos tópicos, abrangendo cada funcionalidade e como foram implementados.

4.1 - Agentes de Referencial

Os agentes de referencial são dispositivos especiais, encarregados de disponibilizar os índices que servem como preço inicial para os agentes Fornecedores. Além disso, em nossa implementação, eles foram utilizados como agentes responsáveis por manter uma lista dos agentes Fornecedores, disponibilizando essa lista para os clientes no momento em que forem iniciar a execução de um serviço e atualizado a cada 10 minutos.

Por implementar a classe *Node*, os serviços básicos são os mesmos descritos no capítulo anterior sobre o Grid-M, para a aplicação desejada, o agente referencial precisava executar serviços de consulta e atualização do banco de dados. A Figura 18 apresenta uma descrição visual do agente Referencial, nela percebemos a utilização da classe *Node* como base para o agente, são utilizadas duas classes estendidas da classe abstrata *Service*, *SQLReference* e *SQLUpdate*. O código fonte da classe Referencial é apresentado no Anexo 1.

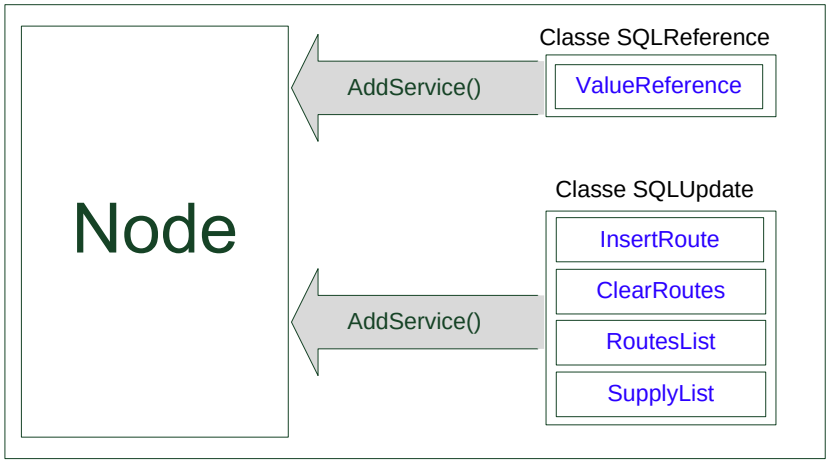

Figura 18 - Diagrama do agente Referencial

A classe *SQLReference*, transcrita no Anexo 2, implementa o serviço *ValueReference*, serviço esse responsável por disponibilizar o índice de desempenho do processador para os agentes Fornecedores. Dentro do ambiente de teste proposto, a solicitação do índice de benchmark era feita uma única vez durante toda a vida do agente fornecedor.

Isso levou-nos a entender que o agente que implementa o serviço *ValueReference* não seria um nodo crítico para o desempenho geral do sistema, já que durante estabelecimento da infra-estrutura dos nodos Fornecedores é que esse agente terá uma demanda maior.

Por esse motivo, optou-se por utilizar o mesmo nodo para implementar uma série de serviços responsáveis por manter uma lista dos nodos próximos, utilizada pelos clientes durante o processo de solicitação da execução. A classe *SQLUpdate*, detalhada no Anexo 3, apresenta os seguintes serviços:

- **InsertRoute**: recebe o nome, IP e tipo do dispositivo que está sendo integrado ao ambiente, quanto ao tipo, o nodo pode ser um *Supplier* (Fornecedor) ou *Consumer* (Consumidor).
- **ClearRoutes**: no final da simulação a lista de nodos é limpa para garantir a não interferência entre uma simulação e outra.
- **RoutesList**: retorna uma lista contendo o nodo e o IP de todos os nodos que estão cadastrados no banco de dados.
- **SupplyList**: retorna uma lista de nodos marcados como sendo do tipo *Supplier*

4.2 - Agentes Fornecedores

Os agentes fornecedores, ou também chamados de nodos executores, são os dispositivos que executam as tarefas dentro do *Grid.* São eles, os dispositivos que devem existir em quantidade consideravelmente maior que os agentes de referencial, já que o desempenho do sistema se baseia na distribuição de tarefas entre esses agentes.

No anexo 4 é apresentada a classe Fornecedor. Assim como os outros agentes, a classe Fornecedor instância *Node*, uma classe da infra-estrutura do Grid-M responsável por estabelecer as funções básicas de um agente. Além disso, o agente Fornecedor estende suas funcionalidades através da implementação de duas classes derivadas da classe *Service*: (i) *CostEstimation* e (ii) *RunTask*.

4.2.1 - Serviço de estimativa de custo.

O serviço de estimativa de custo é o serviço que implementa a equação (19) apresentada no capítulo 3, e que consiste na soma dos custos individuais para os diversos recursos de um dispositivo com os devidos pesos atribuídos para cada recurso.

Derivada da classe *Service* e denominada *PriceService*¸ essa classe implementa o serviço de estimativa de custo. Dentre suas funcionalidades está o orçamento de um serviço baseado nos valores estimado de recursos, que um cliente solicita em tempo de execução através de um instância da classe *XMLTree* enviada junto com a tarefa. A Figura 19 demonstra um exemplo de uma solicitação que é gerada pelo cliente, a qual serve para definir um orçamento para a execução do serviço.

```
<Parameters>
 <Processor>
       <Cycles>50000</Cycles>
 </Processor>
 <Network>
       <SendBytes>50000</SendBytes>
       <ReceiveBytes>10000</ReceiveBytes>
       <Latency>100</Latency>
 </Network>
 <Memory>
       <Size>5000</Size>
 </Memory>
 <Disk>
       <Space>15000</Space>
 </Disk>
</Parameters>
```

Figura 19 - Estimativa de recursos

Ressaltamos que no desenvolvimento desse projeto não foram previstos o tratamento de todos os recursos solicitados no exemplo. Além disso, na execução não verificamos se os valores solicitados são realmente compridos. Numa infra-estrutura em que há a preocupação com os retornos financeiros, é fundamental que sejam auditadas as execuções de forma a punir casos de execuções especulativas.

Porém, durante todo o desenvolvimento, houve a preocupação em facilitar novas funcionalidades ao sistema, para isso há a garantia de expansibilidade através da criação de uma classe derivada da classe abstrata *Metrics* e verificada durante a execução através do pedaço de código apresentado na Figura 20.

```
if (c.getSuperclass().getName().equals(Metrics.class.getName())){
      return (Metrics)c.getConstructor(new
Class[]{XMLTree.class}).newInstance(new Object[]{parameters});
}
```

Figura 20 - Verificação da dependência da Classe *Metrics*

Para cada recurso, uma classe implementa a função de utilidade, para a implementação proposta nessa trabalho, apenas a classe que se refere ao processamento será abordada.

Ao iniciar uma requisição em que o cliente solicita o preço de processamento, a classe *Processor* é invocada e o preço inicial do nodo é definido através de consulta ao agente Referencial. A Figura 21 apresenta um fragmento do código fonte da classe *Processor*,

```
String strCpuModel = CPUInformation.GetExtendedProcessorName();

XMLTree xmlParameters = new XMLTree("Reference"); //Verify the name
xmlParameters.add("processor", strCpuModel);

Task task = new Task("Referencial ", "ValueReference", parameters);

//Send task
TaskResult result = Fornecedor.supplier.sendServiceRequest(task);
XMLTree parameters = result.getParameters();

PriceZero = Integer.parseInt(parameters.getText("SpecResult"));
kFator    = Integer.parseInt(parameters.getText("kFator"));
```

Figura 21 - Requisição do preço inicial do nodo

A partir da definição do preço inicial, o nodo inicia um processo monitorando a utilização do processador, durante a execução da classe *Fornecedor*, esse processo irá coletar informações de processamento que servirão para formar a variação de preço do serviço.

Se considerarmos apenas a utilização atual do processador, teríamos um problema de oscilações abruptas na variação de preço do serviço, essa variação irá influenciar no preço do serviço e conseqüentemente, ocasionar problema ao não representar a real

utilização do processador, por isso optou-se por utilizar uma função de suavização (Torvalds 1992; Narayanan 2002).

A desvantagem de se utilizar essa função é a necessidade de executar mais um processo em segundo plano, o que em dispositivos com recursos limitados, pode comprometer o desempenho.

A função escolhida para a suavização é baseada no comando Unix *loadavg* e composta conforme equação (22).

$$N_{i+1=}\alpha N_i + (1-\alpha)n_i \qquad (22)$$

Onde:

N_i = é o valor *i* da medição suavizada.

n_i = consiste da medição da utilização do processador no momento do cálculo da função

α = é a função de declinação da curva.

Para recuperar informações sobre a utilização atual do processador, utilizamos o sistema de gerenciamento do Windows (Corporation 2007), o WMI, um conjunto de instrumentação baseada no WBEM responsável por gerenciar informações de hardware e software do dispositivo monitorado.

A função de declinação é responsável pela suavização das variações que ocorrem devidas as flutuações de carga no dispositivo, a função é definida como (23):

$$\alpha = e^{\frac{-t_p}{T_s}} \qquad (23)$$

Onde:

t_p = é o intervalo da amostragem. Na simulação foi utilizado o tempo entre amostras de 1 segundo.

T_s = constante que define o tempo de declinação.

A constante que define o tempo de declinação T_s, é o fator que irá influenciar na variação da suavização, valores maiores ocasionarão curvas menos acentuadas e

amostras de processamento precisam manter um valor por longos períodos para que tenha influência na curva, valores menores ocasionarão transições mais acentuadas e valores alternados da amostragem podem provocar picos também na curva suavizada.

Por padrão, o comando *loadavg* apresenta os valores de 1, 5 e 15 minutos para a constante de declinação, optamos pelo valor de 5 minutos por representar adequadamente a variação de carga no período que tínhamos estipulado para teste.

A Figura 22, faz uma apresentação gráfica de uma situação encontrada nos experimentos realizados, onde pode ser observado a medição instantânea do processamento sem qualquer tratamento e as variações da curva de suavização com diferentes valores de declinação.

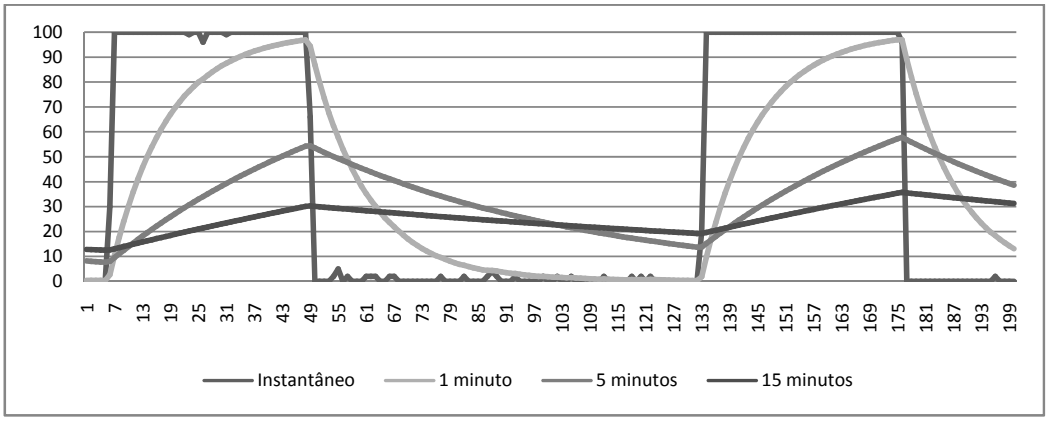

Figura 22 - Variação da carga

O algoritmo utilizado para o sistema foi baseado no código fonte do programa implementado em Linux (Torvalds 1992) e o código fonte dessa classe se encontra no anexo 6.

4.2.2 - Serviço de teste *RunTask*

Para simular uma carga computacional, foi escolhido um algoritmo para estimar o valor do número PI, a escolha se deu considerando a demanda de processamento que esse tipo de algoritmo exige e a possibilidade de alterar a quantidade de processamento necessário, bastante para isso, alterar a quantidade de execuções necessárias.

A aproximação do PI, apresentada na equação (24), foi proposta por Lehmer (Lehmer 1938) e sua complexidade computacional é O(n). A escolha dessa equação se deu pela afinidade em localizar material de referência e implementação.

$$\pi \cong \sum_{i=1} \frac{-16^{i+1}}{(2i-1)5^{2i-1}} - \sum_{i=1} \frac{-4^{i+1}}{(2i-1)239^{2i-1}} \qquad (24)$$

Estipulando valores elevados de repetição, conseguimos simular aplicações de alta granularidade (Dantas 2005), apesar de que o algoritmo ainda exige uma quantidade de memória que não foi considerada na execução da tarefa.

Para a implementação do algoritmo, foi utilizado como base o código fonte apresentado em (Codecodex 2007), que apesar de demonstrar o código na mesma linguagem em que foi desenvolvido esse trabalho, foram necessárias modificações de forma a tornar a implementação adequadas as necessidades do projeto. O código final, utilizado na arquitetura é apresentado no Anexo 7.

4.3 - Agentes Consumidores

Os consumidores são os licitantes responsáveis pelas requisições de tarefas. Para isso, o nodo deve executar uma seqüência de passos, que levarão a execução correta pelos Agentes fornecedores.

Uma das atividades que devem ser planejada antes da solicitação do orçamento é estimar os recursos e quantidade necessários para a execução do serviço, essa atividade não é uma tarefa trivial, vários fatores podem influenciar a quantidade de recursos que uma tarefa pode necessitar. Por isso, nesse projeto não estaremos abordando as técnicas necessárias para decompor uma tarefa, estaremos apenas estimando uma quantidade recursos necessário e alterando o funcionamento da tarefa de forma a refletir os valores estimados.

Em seguida, o nodo licitante, deve disparar as solicitações de orçamento para os nodos que estão presentes em sua lista de executores. O nodo executor tem como função então, interpretar essa lista de recursos e baseada nela gerar o custo para a execução do serviço, retornando o valor para o nodo licitante. A Figura 23 apresenta um fragmento

do código executado pelo cliente responsável por ler a lista de nodos disponíveis e enviar uma tarefa de requisição de estimativa de Custo.

```
while (houver nodos a serem consultados) {
      String node = (Recebe o nome do nodo);
      String url = (Recebe o endereço do nodo);
      cliente.addRoute(node, url );

      Task task = new Task(node, "CostEstimation", parameters);
      TaskResult result = node0.sendServiceRequest(task);
}
```

Figura 23 - Solicitação de orçamento

O nodo recebe os orçamentos gerados pelos agentes executores, para evitar demoras excessivas, principalmente no caso de nodos geograficamente distantes, é definido um tempo limite para a solicitação, o qual caso seja ultrapassado, o nodo é descartado dessa licitação.

A medida que o cliente recebe os orçamentos dos agentes Fornecedores , o nodo tem a tarefa de escolher o menor valor entre todos os lances recebidos, o trecho de código responsável por desempenha essa atividade é apresentado na Figura 24.

```
XMLTree resultParameters =
result.getParameters("ServiceReply").get("CostOfService");

int tmp = Integer.parseInt(resultParameters.getText("Price"));

if(tmp < price){
      price = tmp;
      executor = result.getOriginator();
      ipExecutor = url;
}
```

Figura 24 - Retorno do resultado

A modalidade de leilão fechado foi escolhida para que não ocorra sobrecarga da rede por tráfego de solicitações excessivas. Caso ocorra empate entre os lances retornados, optou-se por implementar uma política que prevalecesse o primeiro lance recebido, por mais que se imagine que um dos orçamentos fosse recebido simultaneamente, o fato de o recebimento estar sincronizado, impede que o nodo licitante receba as respostas no mesmo intervalo de tempo.

Após eleito o nodo fornecedor com menor preço, é criada uma tarefa através de um thread, conforme pode ser observado na Figura 25, esse thread recebe o endereço e o nome do nodo fornecedor e envia a tarefa, aguardando a resposta da solicitação. Foi utilizado esse mecanismo para garantir que diversas execuções fossem enviadas de forma a não bloquear a execução de tarefas simultâneas.

```
ExecutePI task1 = new ExecutePI(executor, node0,bidNumber );
Thread t = new Thread(task1);
```

Figura 25 - Thread de execução

Para cada bloco de simulações, o cliente é responsável pela quantidade de execuções, para isso, a quantidade é controlada através de uma variável já apresentada na figura anterior e denominada *bidNumber*.

5 - RESULTADOS EXPERIMENTAIS

Esse capítulo apresenta os experimentos que foram desenvolvidos e os resultados obtidos. Basicamente, foram executadas três séries com diferentes configurações, conforme apresentado na Tabela 3.

Tabela 3 – Informações sobre as execuções

	Experimento 1	Experimento 2	Experimento 3
Casas decimais do PI	100.000	300.000	500.000
Intervalo entre solicitações	5 segundos	5 segundos	30 segundos
Limite de solicitações	200	200	100

A primeira diferença entre as execuções é a quantidade de casas decimais do PI, esse valor é utilizado na execução da tarefa no agente fornecedor de cada solicitação e representa a quantidade de processamento necessário, numa relação diretamente proporcional da quantidade de casas decimais com o número de execuções do processador.

Em seguida, temos o intervalo entre as execuções feitas pela agente consumidor. A cada momento um novo thread será executado, realizando os processos de leilão e execução de mais um orçamento. Para o experimento 3, decidiu-se aumentar o intervalo entre as solicitações, pois a quantidade de cálculo que era executada ocasionava lentidão excessiva no tempo de resposta dos agentes.

O número de solicitações define a quantidade máxima de solicitações que o agente consumidor solicitará para os fornecedores, a partir do número estipulado, o consumidor pára de enviar solicitações e aguarda o recebimento dos resultados das tarefas que ainda estão pendentes, somente depois disso que a simulação encerra-se.

5.1 - Ambiente

Para a realização dos testes foi utilizado um ambiente com computadores de arquitetura PC e com diferentes características de processador, a Tabela 4 apresenta a relação dos nodos utilizados com o seu respectivo processador e o endereço IP, os dois

pontos separam o endereço da porta de escuta em que os agentes foram executados. Todos os nodos utilizam o sistema operacional Windows XP® e dispõem de 512 MB de memória RAM.

Tabela 4 - Características dos nodos

Nome	Processador	Endereço IP: Porta
node 01	AMD Athlon(tm) 64 Processor 3500+	172.16.5.130:10000
node 04	AMD Athlon(tm) 64 Processor 3500+	172.16.5.241:10000
node 10	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.150:10000
node 11	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.167:10000
node 13	AMD Athlon(tm) 64 Processor 3500+	172.16.5.101:10000
node 15	AMD Athlon(tm) 64 Processor 3500+	172.16.5.67:10000
node 16	AMD Athlon(tm) 64 Processor 3500+	172.16.5.239:10000
node 18	AMD Athlon(tm) 64 Processor 3500+	172.16.5.106:10000
node 19	AMD Athlon(tm) 64 Processor 3500+	172.16.5.190:10000
node 21	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.233:10000
node 25	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.53:10000
node 27	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.70:10000
node 28	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.212:10000
node 29	AMD Athlon(tm) 64 Processor 3500+	172.16.5.133:10000
node 31	Intel(R) Celeron(R) CPU 2.53GHz	172.16.5.192:10000
node 44	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.92:10000
node 61	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.126:10000
node 70	Intel(R) Pentium(R) 4 CPU 2.80GHz	172.16.5.146:10000

Resumindo a tabela apresentada, percebemos que são disponibilizados três tipos de processadores, cada um com suas características diferentes, a Tabela 5 apresenta um resumo da quantidade de nodos utilizados, cada tipo de processador e o respectivo índice de desempenho, esse valor foi retirado da base de benchmarks da SPEC (SPEC 2007).

Tabela 5 - Índice de desempenho

Quantidade	Modelo	Índice de desempenho
8	AMD Athlon(tm) 64 Processor 3500+	14
9	Intel(R) Pentium(R) 4 CPU 2.80GHz	10,4
1	Intel(R) Celeron(R) CPU 2.53GHz	11,5

Através do índice de desempenho de cada processador, percebemos uma vantagem entre o processador Pentium 4 em comparação ao Celeron, uma diferença maior ainda é apresentada entre o processador Athlon e os demais. Esses valores são atribuídos as características construtivas de cada processador, o qual esse trabalho não contempla o detalhamento.

Para não ocorrer diferenças de desempenho e funcionalidade foi padronizada a versão 1.5.0 release 12 do Java para todos os nodos, inclusive nodos cliente e banco de dados de benchmark.

5.2 - Experimentos

Os experimentos foram divididos em três grupos, a Tabela 6 resume os resultados obtidos na execução, a primeira coluna representa o tempo médio de todas as execuções, a segunda coluna o menor tempo que um nodo levou para executar e a terceira, o maior tempo.

Tabela 6 - Resumo execuções

	Média	Menor	Maior
Experimento 1	88,90	73,23	133,86
Experimento 2	235,16	165,14	427,63
Experimento 3	628,27	464,44	1098,67

A diferença entre os tempos em cada experimento atribui-se a diferença entre os processadores e também a carga de processamento em que o nodo apresenta no momento, pois, além das tarefas já executadas pelo Sistema operacional, o próprio

sistema de monitoramente introduzia uma demanda computacional para realizar a coleta e análise do processamento.

A Figura 26 apresenta graficamente os valores da Tabela 6; percebe-se que apesar da equação definida ter sido considerada com aumento linear da sua complexidade em função do número de execuções, graficamente não se comprova essa consideração, suspeita-se que em virtude da grande quantidade de memória utilizada para armazenar o valor de PI, o tempo de leitura/escrita na memória pode ter sido responsável pela diferença.

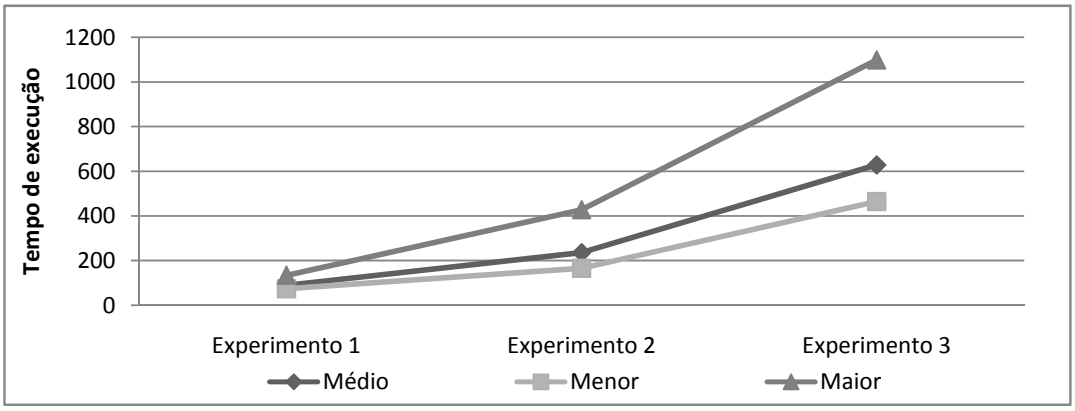

Figura 26 - Resumo gráfico das execuções

Tabela 7 faz a separação dos tempos médio, menor e maior que os nodos levaram para executar o serviço, os resultados foram agrupados pelo tipo de processador para que tenhamos noção de desempenho do nodo com processador de maior desempenho.

Tabela 7 - Tempo de execução por processador

	Médio	**Menor**	**Maior**
Experimento 1			
AMD Athlon(tm) 64 Processor 3500+	74,86	73,23	98,30
Intel(R) Celeron(R) CPU 2.53GHz	79,85	77,81	93,97
Intel(R) Pentium(R) 4 CPU 2.80GHz	102,37	97,55	133,86
Experimento 2			
AMD Athlon(tm) 64 Processor 3500+	213,23	165,14	427,63
Intel(R) Celeron(R) CPU 2.53GHz	234,48	173,81	311,48

Intel(R) Pentium(R) 4 CPU 2.80GHz	254,74	218,91	371,75
Experimento 3			
AMD Athlon(tm) 64 Processor 3500+	526,05	464,44	862,53
Intel(R) Celeron(R) CPU 2.53GHz	620,91	487,80	786,44
Intel(R) Pentium(R) 4 CPU 2.80GHz	719,95	610,70	1098,67

Percebe-se que apesar da freqüência do processador Pentium 4 ser maior do que a do processador Celeron, na prática, o que se observou é que o Celeron teve respostas mais rápidas na execução da tarefa. Acreditamos que esse desempenho seja resultado da pequena cache que o Celeron possui o qual minimiza o tempo de acesso aos dados na memória.

A divisão da quantidade de execuções em cada um dos experimentos é mostrada na Tabela 8, nessa tabela os nodos estão ordenados conforme o processador que possuem, os marcados com o símbolo "*" são os nodos equipados com processador AMD Athlon64 Processor 3500+; o nodo marcado com "#" representa o processador Intel(R) Celeron(R) CPU 2.53GHz e finalmente, os nodos marcados com "&" representam os processadores Intel Pentium 4 2.80GHz.

Tabela 8 - Número de execuções

Nodo	Experimento 1	Experimento 2	Experimento 3
node 01*	20	16	8
node 04*	19	17	7
node 13*	17	18	5
node 15*	19	16	8
node 16*	18	16	5
node 18*	19	15	6
node 19*	16	16	7
node 29*	18	19	6
node 31#	9	8	4
node 10&	6	7	3
node 11&	6	6	7
node 21&	5	6	4

node 25[&]	5	8	5
node 27[&]	4	7	4
node 28[&]	4	5	8
node 44[&]	3	4	4
node 61[&]	6	7	2
node 70[&]	4	6	2
Total	**198**	**197**	**95**

Conforme mencionado no capítulo anterior, o preço enviado para um nodo é composto por um preço inicial, adicionado do índice de variação de carga no processador, essa variação é suavizada para que não ocorram oscilações bruscas no valor do serviço.

Outro detalhe que a Tabela 8 apresenta é a distribuição da quantidade de execução em virtude do índice de desempenho dos nodos, nodos com melhores índices tiveram maior quantidade de execuções, isso garante que teremos uma distribuição de carga uniforme entre as mais diversas capacidades de processamento.

Enquanto isso no experimento 3, praticamente não houve diferença significativa na quantidade de execuções. Essa pouca diferença é atribuída à necessidade de processar uma tarefa com considerável requerimento de processador, o que exigia um grande tempo do nodo processando e por conseqüente elevação do preço dos nodos, refletindo em uma maior quantidade de solicitações de orçamento.

O número de execuções que cada nodo executou nos experimentos tem influência no preço, há tendência de que o maior número de execuções recaia para os nodos com maior capacidade de processamento. Essa tendência vai influenciar o preço da execução do nodo, que conforme podemos observar na Tabela 9, tende a um equilíbrio no preço médio das execuções.

Tabela 9 - Resumos dos preços nos experimentos

		Experimento 1	Experimento 2	Experimento 3
Node 01	Média	435,97	928,80	3774,45
	Mínimo	27,00	101,00	1168,00
	Máximo	1387,00	1301,00	6668,00
Node 04	Média	559,02	910,83	3673,60

	Mínimo	27,00	41,00	568,00
	Máximo	1707,00	1241,00	6468,00
Node 10	Média	410,20	991,67	3487,58
	Mínimo	1,00	1,00	103,00
	Máximo	2001,00	1681,00	5703,00
Node 11	Média	405,74	972,95	3437,39
	Mínimo	1,00	1,00	103,00
	Máximo	2041,00	1501,00	5603,00
Node 13	Média	585,06	929,58	3611,51
	Mínimo	27,00	101,00	968,00
	Máximo	1747,00	1361,00	6468,00
Node 15	Média	594,35	895,03	3397,83
	Mínimo	27,00	221,00	768,00
	Máximo	1747,00	1301,00	6068,00
Node 16	Média	611,55	909,00	3130,79
	Mínimo	27,00	221,00	368,00
	Máximo	1827,00	1181,00	5368,00
Node 18	Média	594,56	956,20	3053,59
	Mínimo	27,00	221,00	168,00
	Máximo	1827,00	1661,00	5268,00
Node 19	Média	510,14	985,15	3384,81
	Mínimo	27,00	161,00	868,00
	Máximo	1787,00	1901,00	6068,00
Node 21	Média	2338,79	971,54	3078,08
	Mínimo	1,00	1,00	303,00
	Máximo	208401,00	1681,00	5103,00
Node 25	Média	352,69	1117,92	3389,25
	Mínimo	1,00	121,00	503,00
	Máximo	2161,00	2281,00	6003,00
Node 27	Média	281,57	1118,48	3487,29
	Mínimo	1,00	121,00	103,00
	Máximo	2001,00	2281,00	6103,00
Node 28	Média	279,99	942,19	3022,98
	Mínimo	1,00	61,00	103,00
	Máximo	1961,00	1681,00	5003,00
Node 29	Média	493,19	996,20	3320,66
	Mínimo	27,00	221,00	568,00
	Máximo	1747,00	2021,00	5368,00
Node 31	Média	356,20	1030,71	95,80
	Mínimo	19,00	28,00	33,00
	Máximo	1739,00	1588,00	153,00

Node 44	Média	335,70	1135,99	3285,60
	Mínimo	1,00	1,00	103,00
	Máximo	2521,00	2161,00	6003,00
Node 61	Média	534,16	1012,03	3601,75
	Mínimo	1,00	1,00	103,00
	Máximo	2481,00	1741,00	5803,00
Node 70	Média	277,85	956,31	3255,19
	Mínimo	1,00	1,00	103,00
	Máximo	2081,00	1681,00	5403,00

Uma das particularidades considerada durante os experimentos foi a medição do tempo em que um orçamento era respondido, Havia a preocupação em verificar o tempo necessários para que um nodo recebe-se uma requisição de orçamento, realiza-se a devida coleta de dados de processamento e retornar-se a informação solicitada.

Percebemos na Tabela 10, que em alguns casos, ocorreram valores expressivamente elevados dentro do campo máximo, em todos os casos verificados, os momentos que ocorrem os maiores tempos de respostas são aqueles em que o nodo está executando uma atividade de uma requisição anterior.

Tabela 10 - Resumo do tempo para retorno dos orçamentos.

		Experimento 1	**Experimento 2**	**Experimento 3**
Node 01	Média	698,52	1735,07	749,07
	Mínimo	15,00	15,00	15,00
	Máximo	4032,00	17797,00	2407,00
Node 04	Média	398,79	1345,20	1720,99
	Mínimo	15,00	15,00	15,00
	Máximo	1875,00	9703,00	8875,00
Node 10	Média	315,70	634,81	748,95
	Mínimo	15,00	15,00	15,00
	Máximo	2625,00	4687,00	2438,00
Node 11	Média	276,15	789,12	1074,24
	Mínimo	15,00	15,00	15,00
	Máximo	1938,00	6984,00	4109,00
Node 13	Média	381,12	1114,02	737,11
	Mínimo	15,00	15,00	15,00
	Máximo	1188,00	13125,00	2422,00
Node 15	Média	383,38	1673,34	987,99
	Mínimo	15,00	15,00	15,00

	Máximo	1750,00	17031,00	5672,00
Node 16	Média	352,93	1059,85	1010,67
	Mínimo	15,00	15,00	15,00
	Máximo	1890,00	7812,00	5891,00
Node 18	Média	368,41	1801,49	1738,96
	Mínimo	15,00	15,00	15,00
	Máximo	1843,00	26141,00	9844,00
Node 19	Média	316,61	1063,27	1314,99
	Mínimo	15,00	15,00	15,00
	Máximo	2687,00	6469,00	8563,00
Node 21	Média	208,78	417,77	1102,45
	Mínimo	15,00	15,00	15,00
	Máximo	1922,00	4203,00	4328,00
Node 25	Média	210,28	1120,02	4191,65
	Mínimo	15,00	15,00	15,00
	Máximo	2093,00	11375,00	25422,00
Node 27	Média	200,78	568,55	2306,64
	Mínimo	15,00	15,00	15,00
	Máximo	1968,00	6656,00	12265,00
Node 28	Média	177,86	620,72	752,51
	Mínimo	15,00	15,00	15,00
	Máximo	1766,00	6422,00	2532,00
Node 29	Média	313,09	1251,39	1742,55
	Mínimo	15,00	15,00	15,00
	Máximo	1953,00	13000,00	8609,00
Node 31	Média	314,87	1184,63	1872,64
	Mínimo	15,00	15,00	15,00
	Máximo	4312,00	13313,00	14578,00
Node 44	Média	185,32	803,50	887,50
	Mínimo	15,00	15,00	15,00
	Máximo	2110,00	8422,00	10344,00
Node 61	Média	222,95	590,49	687,63
	Mínimo	15,00	15,00	15,00
	Máximo	1359,00	4328,00	2062,00
Node 70	Média	173,05	410,80	694,46
	Mínimo	15,00	15,00	15,00
	Máximo	1953,00	3782,00	2157,00

Nos três experimentos realizados monitoramos todos os nodos, primeiro para conhecermos a realidade de execuções e de variação de preço, porém percebemos segundo as tabelas apresentadas, que apenas selecionando nodos com os três tipos

diferentes de processadores teríamos uma amostragem do funcionamento do sistema como um todo.

Porém, se permitíssemos que apenas três nodos executassem o sistema de monitoramento, poderíamos gerar um desequilíbrio nos resultados comparando com os outros nodos. Com disso, é considerável apresentarmos os resultados obtidos das variações encontradas.

A Figura 27, apresenta as variações encontradas no experimento 1, vale lembrar que foram escolhidos três nodos que representassem três diferentes tipos de processador.

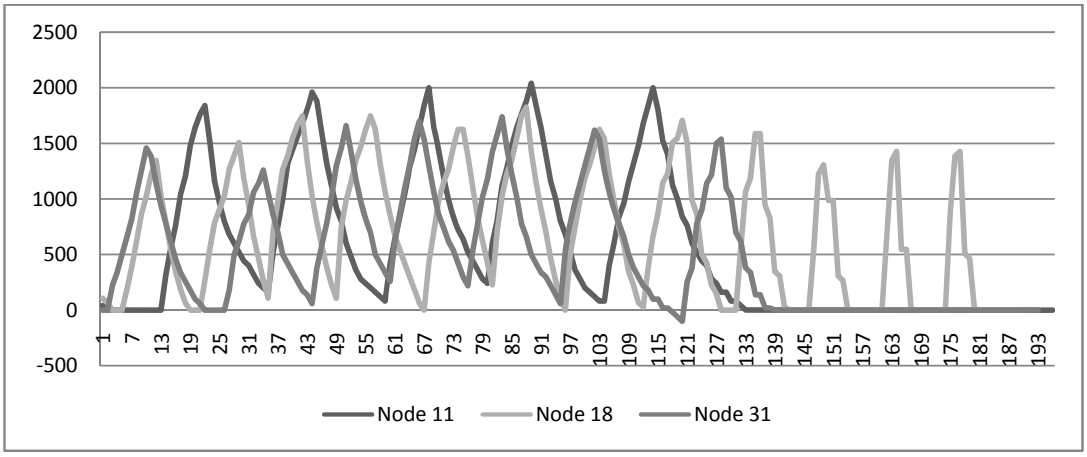

Figura 27 - Variação de preço no experimento 1

No gráfico do experimento 1, podemos perceber variações bruscas nos valores de preço encontrados se comparado ao gráfico dos experimentos 2 e 3, atribuímos esse resultado ao tamanho reduzido da tarefa que ocasionava pequenas oscilações apenas.

As variações ficam menos visíveis a partir do experimento 2, demonstrado na Figura 28. Com o aumento de carga no nodo, as variações são menos freqüente, podemos ainda observar tendência de equilíbrio no final da simulação, o qual tende a se estender se as simulações estivessem se estendido.

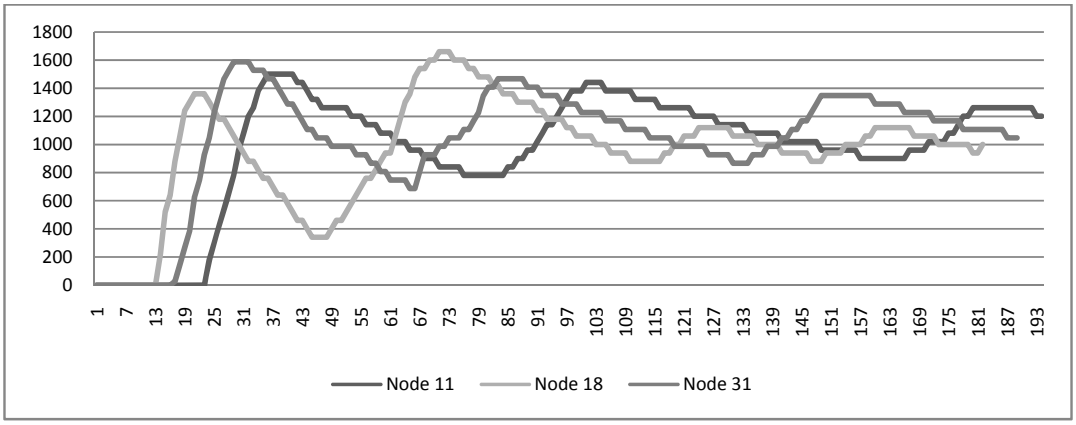

Figura 28 - Variação de preço no experimento 2

E finalmente, na variação de preços do Experimento 3, demonstrada na Figura 29, podemos perceber a mesma tendência de equilíbrio entre os três diferentes tipos de dispositivos, com o diferencial de que as variações se dão em valores mais elevadores, devido exclusivamente ao extenso processamento necessários para realizar o serviço.

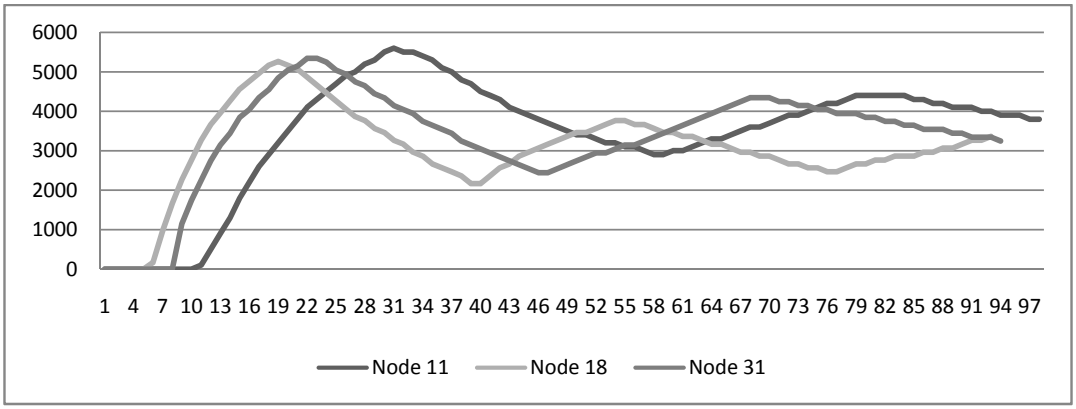

Figura 29 - Variação de preço no experimento 3

Nesses experimentos, onde percebermos que os computadores estavam divididos em três blocos principais de computadores, divididos de acordo com o tipo de processador, e que os softwares em todos eles, com exceção dos drives, eram iguais, mesmo assim, ocorrem pequenas diferenças em relação aos resultados. Não esperávamos resultados iguais, mas podemos perceber que dentro da área de computação uma grande quantidade de fatores pode influenciar a execuções.

O estudo desses fatores não cabe dentro desse trabalho, mas deixamos em aberto o tópico para que possamos aprofundar nossos conhecimentos na área de Ciência de computação, uma área que tem cada vez mais presença em nossas vidas.

6 - CONCLUSÃO

No momento de pesquisa desse trabalho, percebemos a carência que existe em softwares de escalonamento para sistemas distribuídos que permitam fazer uma distribuição de tarefa efetiva entre os nodos, percebemos um agravante no momento que consideramos nodos heterogêneos.

O projetista tem que se preocupar em atender toda a diversidade de tipos de dispositivo que pode estar inserido em um sistema distribuído, porém, geralmente essa preocupação se dá apenas nesse nível, o grau de dificuldade é substancialmente incrementando se tivermos que ponderar as diferentes características de desempenho que podem fazer parte desse conjunto de equipamentos.

Com o objetivo de colaborar nesse segmento, resolvemos utilizar conceitos de economia, que já estão consolidados dentro da interação humana, um dos fatores que levaram a utilizar esses conceitos é a similaridade apresentada com os sistemas distribuídos. Apresentamos alguns modelos de mercado, como ocorre o método de escalonamento e uma taxonomia dos sistemas existentes.

Em seguida foram detalhados alguns conceitos pertinentes a formação de preço em um nodo baseado em seus recursos computacionais. Levamos em consideração os recursos separadamente, não considerando a influência que um recurso pode desencadear em relação aos outros recursos.

Baseado nesses conceitos, apresentamos a proposta desse trabalho, que é a realização do escalonamento baseado em um fator comum (preço). Para isso nos limitamos a utilizar as características da CPU, considerando as diferentes capacidades de processamento como um fator de influência no preço.

Após termos realizados o detalhamento técnico da nossa proposta, partimos para a infra-estrutura da simulação, partimos do princípio que os nodos são idôneos, não consideramos fatores como segurança e autenticidade das mensagens que eram trocadas entre os nodos.

Nosso ambiente de teste utilizou-se de uma quantidade mediana de nodos, porém não consideramos situações em que os nodos podem estar situados em locais afastados geograficamente. Isso poderia influenciar no resultado de nossos experimentos, pelo fato que o modelo de mercado escolhido realiza licitações para escolha do fornecedor, o

tempo que seria necessário para o envio e recebimento dos resultados poderia degradar o desempenho do sistema.

Apesar disso, os resultados foram animadores, o sistema distribuiu as tarefas de acordo com a capacidade de processamento de cada processador. Percebemos ainda, que o preço do serviço de cada nodo tem tendência a chegar a um equilíbrio, esse equilíbrio é baseado na distribuição das tarefas nos nodos, equilibrando a utilização de processamento de cada nodo.

Como legado, deixamos a base de conhecimento desse tipo de escalonador e a extensão da infra-estrutura de Grid utilizada, a qual permite que novos recursos possam ser quantificados para a formação do preço do serviço, como por exemplo, o uso da memória principal, espaço em disco e quantidade de dados trafegados na rede.

6.1 - Trabalhos Futuros

Durante a elaboração desse trabalho, foi necessário abdicar de certos tópicos, para que não perdêssemos o andamento de desenvolvimento do assunto principal, também durante o desenvolvimento percebíamos que novas funcionalidades poderiam ter sido incluídas, essas idéias foram sintetizadas e servem como futuras direções para que sejam incrementadas ao sistema proposto:

- Estender a função de utilidade para os recursos de comunicação de dados, memória principal e espaço em disco: a infra-estrutura do sistema de economia para o Grid-M foi desenvolvido pensando em futuras extensões para os recursos que poderiam estar sendo mensurados. Dessa forma é possível abranger novos recursos e utilizar os já comumente conhecidos para definição do preço. Como trabalhos futuros, pretendemos utilizar essa expansibilidade para definir equações de utilidade para a memória principal, espaço em disco e comunicação de dados, esse último com maior ênfase.
- Realizar experimentos com nodos com maior disparidade de recursos: apesar da quantidade de máquina envolvida ser considerável, os nodos apresentam processadores com poucas diferenças de desempenho e

estavam conectados a uma rede de comunicação local de alta velocidade, experimentos devem ser executados em ambientes com maior disponibilidade de nodos, inclusive com a sugestão de utilizar dispositivos sem fio, e também com nodos espalhados através de várias organizações.

- Utilizar métricas de qualidade do serviço como fatores que influenciam no preço: num sistema de Economia Grid, os recursos são peças chaves do sistema, porém para o usuário, não basta apenas ter o recurso disponível, ele preza pela qualidade do que está contratando, por isso, uma arquitetura que quantifique índices de qualidade e represente esses valores de forma monetária é uma proposta possível para o trabalho exposto, o principal problema dessa proposta é a dificuldade em utilizar grande parte dos indicadores e convertê-los numa moeda única.
- Controle da execução: no processo de licitação, os nodos fornecem seus valores e o cliente assume que esse processo é confiável, não é feita uma verificação posterior a execução se o acordo foi cumprido, um mecanismo Regulador seria uma alternativa, impondo regras e cobrando os nodos que descumprirem acordos acertados.
- Mecanismos de contabilização: o trabalho proposto não contém nenhuma forma de compensação ou desconto pela execução do serviço, a princípio pelo escopo do trabalho que foi focado na parte de escalonamento, porém é uma necessidade futura que seja implementado um sistema de contabilização e que realmente efetue o sistema de cobrança e proventos.

REFERÊNCIAS BIBLIOGRÁFICAS

Agents, F. f. I. P. (2001, 2002/12/03). "FIPA ACL Message Structure Specification." Retrieved 26 de dezembro de 2007, 2007, from http://www.fipa.org/specs/fipa00061/.

Ahuja, S. P. and J. R. Myers (2006). "A Survey on Wireless Grid Computing " The Journal of Supercomputing **Volume 37**(Issue 1): 3-21.

Al-Ali, R. J., A. ShaikhAli, et al. (2003). "Supporting QoS-based discovery in service-oriented Grids." 17th International Symposium on Parallel and Distributed Processing: 22-26.

Appleby, K., S. Fakhouri, et al. (2001). "Oceano - SLA based management of a computing utility." International Symposium on Integrated Network Management Proceedings: 855-868.

Berman, F., G. Fox, et al. (2003). Grid computing : making the global infrastructure a reality, John Wiley & Sons Inc.

Broy, M. (2003). Service-Oriented Systems Engineering: Modeling Services and Layered Architectures. Formal Techniques for Networked and Distributed Systems, Springer Berlin / Heidelberg. **Volume 2767/2003**.

Broy, M., I. H. Krüger, et al. (2007). "A formal model of services." ACM Transactions on Software Engineering and Methodology **Volume 16**(Issue 1).

Bunker, G. and D. Thomson (2006). Delivering Utility Computing - Business-driven IT Optimization, John Wiley & Sons Ltd.

Buyya, R. (2002). Service-Oriented Grid Architecture for Distributed Computational Economies. School of Computer Science and Software Engineering. Melbourne, Monash University**:** 180.

Buyya, R., D. Abramson, et al. (2002). "Economics Paradigm for Resource Management and Scheduling in Grid Computing " Concurrency and Computation: Practice and Experience **Vol. 14**: 1507-1542.

Buyya, R., D. Abramson, et al. (2003). "Economic models for resource management and scheduling in Grid computing." The Journal of Concurrency and Computation: Practice and Experience (CCPE): 1507 - 1542.

Cheliotis, G., C. Kenyon, et al. (2004). Grid Economics: 10 Lessons from Finance Peer-to-Peer Computing: Evolution of a Disruptive Technology. R. Subramanian and B. Goodman. Hershey, PA, USA., Idea Group Publisher.

Codecodex. (2007, 2007/11/21). "Digits of pi calculation." Retrieved 26 de dezembro de 2007, 2007, from http://www.codecodex.com/wiki/index.php?title=Digits_of_pi_calculation.

Corporation, M. (2007). "WMI: Introduction to Windows Management Instrumentation." Retrieved 26 de dezembro de 2007, 2007, from http://www.microsoft.com/whdc/system/pnppwr/wmi/WMI-intro.mspx.

Cotton, I. W. (1975). "Microeconomics and the Market for Computer Services." ACM Computing Surveys (CSUR) **Volume 7**(Issue 2): 95 - 111

Dantas, M. (2005). Computação Distribuída de Alto Desempenho, Axcel Books do Brasil.

de Roure, D., M. A. Baker, et al. (2003). The evolution of the Grid. Grid Computing – Making the Global Infrastructure a Reality. F. Berman, G. Fox and A. J. G. Hey, John Wiley & Sons, Ltd.

Ferguson, D. F. (1989). "The application of microeconomics to the design of resource allocation and control algorithms." ACM Transactions on Software Engineering and Methodology 166

Ferguson, D. F., C. Nikolaou, et al. (1996). "Economic models for allocating resources in computer systems." World Scientific Publishing Co., Inc. : 156 - 183

Foster, I. (2002). What is the Grid? A Three Point Checklist, Argonne National Laboratory & University of Chicago**:** 4.

Foster, I., C. Kesselman, et al. (2001). "The Anatomy of the Grid." To appear: International Jornal of Supercomputer Applications: 25.

Grid-M. (2007). "Grid-M : Middleware para Integrar Dispositivos Móveis, Sensores e Grids." Retrieved 10 de janeiro de 2008, from http://grid.lrg.ufsc.br/.

GridCafé. (2007). "Grid projects in the world." Retrieved 08 de janeiro de 2008, from http://gridcafe.web.cern.ch/gridcafe/gridprojects/grid-tech.html.

Gurun, S., C. Krintz, et al. (2004). "NWSLite: a light-weight prediction utility for mobile devices." International Conference On Mobile Systems, Applications And Services: 2 - 11.

Härting, R. and W. Kofler (2006). Economics of Resources. Environmental Economics**:** 1-23.

Kenyon, C. and G. Cheliotis (2004). Grid resource commercialization: economic engineering and delivery scenarios. Grid resource management: state of the art and future trends. Norwell, MA, USA, Kluwer Academic Publishers**:** 465 - 478

Klemperer, P. (1999). "Auction Theory: A Guide to the Literature." Journal of Economic Surveys **Volume 13**(Issue 3): 227-286.

Kurose, J. F. and R. Simha (1989). "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems " IEEE Transactions on Computers **Volume 38**(Issue 5): 705 - 717.

Lai, K., L. Rasmusson, et al. (2005). "Tycoon: An implementation of a distributed, market-based resource allocation system." Multiagent and Grid Systems **1** (3): 169 - 182

Lehmer, D. H. (1938). "On Arccotangent Relations for PI." The American Mathematical Monthly **45**: 8.

LRG. (2007). "LRG - Laboratório de Redes e Gerência." Retrieved 28 de fevereiro de 2008, from http://www.lrg.ufsc.br/.

McConnell, C. R., S. L. Brue, et al. (2003). Microeconomics, The McGraw−Hill Companies.

McKnight, L. W., D. Anius, et al. (2002). "Virtual Markets in Wireless Grids: Peering Policy Obstacles." 30th Research Conference on Communication, Information and Internet Policy.

McKnight, L. W., J. Howison, et al. (2004). "Wireless Grids: Distributed Resource Sharing by Mobile, Nomadic, and fixed Devices." IEEE Internet Computing **Volume 8**(Issue 4): 24- 31.

Nakai, J. (2001). Pricing Computing Resources: Reading Between the Lines and Beyond. Technical Report - NAS-01-010, NASA Ames Research Center. NASA Advanced Supercomputing Division.

Narayanan, D. (2002). Operating system support for mobile interactive applications Computer Science Department. Pittsburgh, PA, Carnegie Mellon University. **PhD thesis:** 209.

Parkes, D. C. and L. H. Ungar (2001). "An auction-based method for decentralized train scheduling." Fifth International Conference on Autonomous Agents: 43 - 50

Piro, R. M. (2003). Simulation of Economy-based Load Balancing in Computational Grid for Large-Scale Scientific Applications. Istituto Nazionale di Fisica Nucleare. Torino, University of Torino. **Master:** 144.

Piro, R. M., A. Guarise, et al. (2003). "An economy-based accounting infrastructure for the datagrid." Fourth International Workshop on Grid Computing: 202- 204.

Rajkumar Buyya , D. A., Jonathan Giddy , Heinz Stockinger (2003). "Economic models for resource management and scheduling in Grid computing." Concurrency and Computation: Practice and Experience **14**(13-15): 1507 - 1542.

Rappa, M. A. (2004). "The utility business model and the future of computing services." IBM Systems Journal **Volume 43**(Issue 1): 32 - 42

Reddy, S. R. (2006). Market Economy Based Resource Allocation in Grids. Institute of Technology. Kharagpur, School of Information Technology. **Master:** 72.

Reddy, S. R. and A. Gupta (2006). Auction Based Resource Allocation in Grids. Distributed Computing and Networking. Berlin Springer Berlin / Heidelberg. **Volume 4308/2006:** 145-156.

Rolim, C. O. (2007). Uma arquitetura para submissão de aplicações de dispositivos móveis e embarcados para uma configuração de grade computacional. Centro Tecnológico. Florianópolis, Universidade Federal de Santa Catarina. **Mestrado:** 110.

Smarr, L. and C. E. Catlett (1992). "Metacomputing." Communications of the ACM **Volume 35**(Issue 6): 44 - 52.

SPEC. (2007, 2007/12/14). "SPEC's Benchmarks and Published Results." Retrieved 26 de dezembro de 2007, 2007, from http://www.spec.org/benchmarks.html.

Subramoniam, K., M. Maheswaran, et al. (2002). "Towards a micro-economic model for resource allocation in Grid computing systems." IEEE International Symposium on High Performance Distributed Computing **Volume 2**: 782 - 785

Sun Microsystems, I. (2007). "Network.com." Retrieved 29 de dezembro de 2007, from http://www.network.com/.

Tanenbaum, A. S. and M. van Steen (2002). Distributed Systems: Principles and Paradigms, Prentice Hall.

TOP500.Org. (2007). "Top500 Supercomputing Site." Retrieved 28 de dezembro de 2007, from http://www.top500.org/.

Torvalds, L. (1992). "Kernel internal timers." Retrieved 26 de dezembro de 2007, 2007, from http://lxr.linux.no/linux/kernel/timer.c#L864.

van Moorsel, A. (2001). "Metrics for the Internet Age: Quality of Experience and Quality of Business." To be published in the Fifth Performability Workshop.

W3C. (2007, 2007/10/24). "HTTP - Hypertext Transfer Protocol." Retrieved 26 de dezembro de 2007, 2007, from http://www.w3.org/Protocols/.

Waldspurger, C. A., T. Hogg, et al. (1992). "Spawn: A Distributed Computational Economy." IEEE Trans. on Software Engineering **Volume 18**(Issue 2): 103-117.

Wang, Z., B. Yu, et al. (2005). Wireless Grid Computing over Mobile Ad-Hoc Networks with Mobile Agent. First International Conference on Semantics, Knowledge and Grid. Beijing, China.

Wolski, R., J. S. Plank, et al. (2000). G-Commerce – Building Computational Marketplaces for the Computational Grid. Technical Report UT-CS-00-439, University of Tennessee.

Wolski, R., J. S. Plank, et al. (2001). "Analyzing Market-Based Resource Allocation Strategies for the Computational Grid " International Journal of High Performance Computing Applications **Volume 15**: 258-281.

Xue, Y., B. Li, et al. (2003). Price-based Resource Allocation in wireless ad hoc networks. Eleventh International Workshop on Quality of Service. Berkeley, CA, USA, Springer Berlin. **Volume 2707/2003:** 79

Yeo, C. S. and R. Buyya (2006). "A taxonomy of market-based resource management systems for utility-driven cluster computing." Software - Practice & Experience **Volume 36**(Issue 13): 1381 - 1419.

Yuan, W. (2004). GRACE-OS: An Energy-Efficient Mobile Multimedia Operating System. Department of Computer Science. Urbana-Champaign, University of Illinois. **Ph.D:** 158.

ANEXOS

ANEXO 1

```
public class Referencial {
	private static final String port = "11000";

	Referencial(String nodeName, String nodeUrl ){

		Service sqlmanager = new SQLUpdate();

		// Node definition
		Node reference = new Node(nodeName, nodeUrl);
		reference.setLogLevel(0);
		NetInterface httpNetInterface = new
HTTPServerInterface(reference, Integer.parseInt(port));
		httpNetInterface.setSnoopMode(true);

		// Add the services
		reference.setNetInterface(httpNetInterface);
		reference.addService("ValueReference", new SQLReference());
		reference.addService("InsertRoute", sqlmanager);
		reference.addService("ClearRoutes", sqlmanager);
		reference.addService("RoutesList", sqlmanager);
		reference.addService("SupplyList", sqlmanager);

		// Inilializing the node
		reference.start();

	}
}
```

ANEXO 2

```
public class SQLReference extends Service {
	// JDBC driver name and database URL
	private static final String JDBC_DRIVER="com.mysql.jdbc.Driver";
	private static final String DATABASE_URL =
		"jdbc:mysql://192.168.0.100/db";
	private static float SpecResult = 0;
	private static float kFactor = 1;
	private static float definedPrice = 0;

	protected TaskResult _execute(Node node, Task task) throws
Exception {

		XMLTree parameters = task.getParameters().get("Reference");
		//Get parameters from XMLTree
		PriceSqlQuery (parameters.getText("processor"));

		TaskResult tr = new TaskResult(task, TaskResult.RESULT_OK,
null);
		tr.addParameters("Result", "" + SpecResult);
		tr.addParameters("KFactor", "" + kFactor);

		return tr;
	}

	public void PriceSqlQuery(String CpuModel){
		Connection connection = null; // manages connection
		Statement statement = null; // query statement

		String strCpuModel = CpuModel.replaceAll("\\(\\w*\\)",
"").replaceAll("\\W", " ");
		String sqlQuery = "SELECT kfator, DefinedPrice, Benchmark,
Result, Processor FROM table.cpulist, table.economy WHERE processor
LIKE \"" + strCpuModel.replaceAll("Processor", "").replace(' ', '%')+
"%\"";

		try{
			Class.forName( JDBC_DRIVER );
			connection = DriverManager.getConnection(
DATABASE_URL, "user", "pass" );
			statement = connection.createStatement();
			ResultSet resultSet =
statement.executeQuery(sqlQuery);

			int numberOfRows = 0;
			String Benchmark;

			while(resultSet.next()){
				numberOfRows = 1;
				kFactor = resultSet.getFloat("kFator");
				definedPrice= resultSet.getInt("DefinedPrice");
				Benchmark = resultSet.getString("Benchmark");
				SpecResult  = resultSet.getFloat("Result");
```

```
                        String Processor =
resultSet.getString("Processor");

                  }
                  if (numberOfRows == 0){
                        JOptionPane.showMessageDialog(null,"Not found
Benchmark to " + strCpuModel );
                        System.exit(0);
                  }

            }
            catch ( SQLException sqlException ){
                  sqlException.printStackTrace();
                  System.exit( 1 );
            }
            catch ( ClassNotFoundException classNotFound ) {
                  classNotFound.printStackTrace();
                  System.exit( 1 );
            }
            finally
            {
                  try  {
                        statement.close();
                        connection.close();
                  }
                  catch ( Exception exception ){
                        exception.printStackTrace();
                        System.exit( 1 );
                  }
            }
      }
}
```

ANEXO 3

```
public class SQLUpdate extends Service {
	// JDBC driver name and database URL
	static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
	static final String DATABASE_URL =
"jdbc:mysql://192.168.0.100/db";
	Connection connection = null;

	protected TaskResult _execute(Node node, Task task) throws
Exception {
		XMLTree parameters = task.getParameters().get("Reference");
//Get parameters from XMLTree

		if (task.getService().equals("InsertRoute"))
			return this.insertNodeRoute(parameters, task);
		else if (task.getService().equals("ClearRoutes"))
			return this.cleanRoutes(task);
		else if (task.getService().equals("RoutesList"))
			return this.routeList(task);
		else if (task.getService().equals("SupplyList"))
			return this.supplyList(task);
		else
			return new TaskResult(task,
TaskResult.RESULT_NOT_UNDERSTOOD, null);

	}
	private Statement connectDB(){

		try{
			Class.forName( JDBC_DRIVER ); // load database driver
class
			connection = DriverManager.getConnection(
DATABASE_URL, "user", "pass" );
			return connection.createStatement();
		} // end catch
		catch ( ClassNotFoundException classNotFound ) {
			classNotFound.printStackTrace();
			System.exit( 1 );
		}
		catch ( SQLException sqlException ){
			sqlException.printStackTrace();
			System.exit( 1 );

		} // end catch
		return null;
	}

	private TaskResult insertNodeRoute(XMLTree parameters, Task
task){
		String node, route, type;
		Statement connector = null;
		// Get the node parameters to be insertting into the
database
		node = parameters.getText("NodeName");
		route = parameters.getText("NodeRoute");
```

```
            type = parameters.getText("NodeType");

            String sqlInsert= "INSERT INTO table.routelist VALUES
(NULL, "+"\""+ node + "\", "+"\""+ route +"\", \""+ type +"\" )" ;

            try   {
                  connector = connectDB();
                  // query database
                  int rows = connector.executeUpdate(sqlInsert);
                  connector.close();

            } // end try
            catch (SQLException e) {
                  e.printStackTrace();
                  System.exit( 1 );
            }
            finally // ensure statement and connection are closed
properly
            {
                  try {
                        connector.close();
                        connection.close();
                  } // end try
                  catch ( Exception exception )
                  {
                        exception.printStackTrace();
                        System.exit( 1 );
                  } // end catch
            } // end finally
            return new TaskResult(task, TaskResult.RESULT_OK, null);
      }

      private TaskResult cleanRoutes(Task task){
            Statement connector = null;
            String sqlDelete= "DELETE FROM table.routelist";

            try   {
                  connector = connectDB();
                  // query database
                  int rows = connector.executeUpdate(sqlDelete);
                  connector.close();

            } // end try
            catch (SQLException e) {
                  e.printStackTrace();
                  System.exit( 1 );
            }
            finally // ensure statement and connection are closed
properly
            {
                  try {
                        connector.close();
                        connection.close();
                  } // end try
                  catch ( Exception exception )
                  {
                        exception.printStackTrace();
                        System.exit( 1 );
                  } // end catch
```

```
            } // end finally
            return new TaskResult(task, TaskResult.RESULT_OK, null);
      }

      private TaskResult routeList(Task task){
            Statement connector = null;

            String sqlSelect2=”SELECT node, route FROM
table.routelist";

            XMLTree routes = new XMLTree("RouteList");
            try   {
                  connector = connectDB();
                  // query database
                  ResultSet resultSet =
connector.executeQuery(sqlSelect2);
                  // process query results

                  while(resultSet.next()){
                        XMLTree node = new XMLTree("Node");
                        node.add(resultSet.getString("node"),
resultSet.getString("route"));
                        routes.add(node);
                  }
                  connector.close();

            } // end try
            catch (SQLException e) {
                  e.printStackTrace();
                  System.exit( 1 );
            }
            finally {
                  try {
                        connector.close();
                        connection.close();
                  } // end try
                  catch ( Exception exception )
                  {
                        exception.printStackTrace();
                        System.exit( 1 );
                  } // end catch
            } // end finally
            return new TaskResult(task, TaskResult.RESULT_OK, routes);
      }

      private TaskResult supplyList(Task task){
            Statement connector = null;

            String sqlSelect2= "SELECT  node, route FROM
table.routelist WHERE type = 'supply'"  ;

            XMLTree routes = new XMLTree("SupplyList");
            try   {
                  connector = connectDB();
                  // query database
                  ResultSet resultSet =
connector.executeQuery(sqlSelect2);
                  // process query results
```

```
                  while(resultSet.next()){
                        XMLTree node = new XMLTree("Node");
                        node.add(resultSet.getString("node"),
resultSet.getString("route"));
                        routes.add(node);
                  }
                  connector.close();

            } // end try
            catch (SQLException e) {
                  e.printStackTrace();
                  System.exit( 1 );
            }
            finally {
                  try {
                        connector.close();
                        connection.close();
                  } // end try
                  catch ( Exception exception )
                  {
                        exception.printStackTrace();
                        System.exit( 1 );
                  } // end catch
            } // end finally
            return new TaskResult(task, TaskResult.RESULT_OK, routes);
      }

}
```

ANEXO 4

```
public class Fornecedor {
	public static String nodeName;
	public static String nodeUrl;
	public static final String port = "10000";
	public static Node supplier ;
	// Set Supplier
	Fornecedor(){
		supplier = new Node(nodeName, nodeUrl);
		supplier.setLogLevel(0);
		NetInterface httpNetInterface = new
HTTPServerInterface(supplier, Integer.parseInt(port));
		httpNetInterface.setSnoopMode(true);
		supplier.setNetInterface(httpNetInterface);
		supplier.addService("CostEstimation", new PriceService());
		supplier.addService("PI", new PIService());
		supplier.start();
	}

	public static void main(String args[]){
		java.net.InetAddress i;
		try {
			i = java.net.InetAddress.getLocalHost();
			// Getting the name and IP address of the device
			nodeName = "Supplier-" + i.getHostName();
			nodeUrl = "http://" + i.getHostAddress() + ":"+ port;
		} catch (UnknownHostException e) {
			e.printStackTrace();
			System.exit(1);
		}
		new Fornecedor();
	}
}
```

ANEXO 5

```
public class PriceService extends Service {
	private final static String PACKAGE =
Metrics.class.getPackage().getName();

	protected TaskResult _execute(Node node, Task task) throws
Exception {
		if (task.getService().equals("CostEstimation")) {

			long price = 0;
			//Get parameters from XMLTree
			XMLTree parameters = task.getParameters();
			Iterator it =
parameters.get("ServiceRequest").getSubTrees();
			System.out.println("Request cost of service from " +
task.getOriginator());
			while (it.hasNext()) {
				int tmp = 0 ;
				XMLTree value = (XMLTree) it.next();
				tmp = getCostOfService(value.getTag(),
value.get("Metrics") );
				if (tmp < 0)
					tmp = 0;
				price+=tmp;
			}
			return new TaskResult(task, TaskResult.RESULT_OK,
buildReply(price));
		}
		return new TaskResult(task,
TaskResult.RESULT_NOT_UNDERSTOOD, null);
	}

	public int getCostOfService(String metricName, XMLTree
Parameters){
		Metrics m = stringToClass(metricName,Parameters );
		if (m != null){
			return m.getCost();
		}
		return 0;
	}

	public Metrics stringToClass(String className, XMLTree
parameters) {
		try {
			Class c = Class.forName(PACKAGE + "." + className);
			if
(c.getSuperclass().getName().equals(Metrics.class.getName())){
				return (Metrics)c.getConstructor(new
Class[]{XMLTree.class}).newInstance(new Object[]{parameters});
			}
			throw new ClassCastException();
		}catch ( ClassNotFoundException classException ) {
			System.err.println("Class not found: " + className );
		}
```

```
//          handle exception instantiating factory
            catch ( InstantiationException exception ) {
                  System.err.println("Instantiation Exception for
class:" + className );
            }
            // handle exception if no access to specified Class
            catch ( IllegalAccessException accessException ) {
                  System.err.println("Illegal Access Exception for
class:" + className );
            }
            catch ( ClassCastException castException ) {
                  System.err.println("Class " + className + " is not a
subclass of Metrics" );
            }
            catch (Exception e) {
                  e.printStackTrace();
            }
            return null;
      }
      private XMLTree buildReply(long price){
            XMLTree parameters = new XMLTree("ServiceReply"); //Verify
the name
            XMLTree xt1 = new XMLTree("SystemInfo");
            parameters.add(xt1);

            XMLTree xt2 = new XMLTree("CostOfService");
            xt2.add("Price", Long.toString(price));
            parameters.add(xt2);

            return parameters;
      }
}
```

ANEXO 6

```
public class LoadAvg{
	private final static int HZ =		100;
	private final static int FSHIFT =	 11;
	private final static int FIXED_1 =	 (1<<FSHIFT);
	private final static int LOAD_FREQ =(5*HZ);
	private final static int EXP_1 =	 1884;
	private final static int EXP_5 =	2014;
	private final static int EXP_15 =	 2037;

	private static void CALC_LOAD(int indice, int exp, int n){
		avenrun[indice] *= exp;
		avenrun[indice] += n*(FIXED_1-exp);
		avenrun[indice] >>= FSHIFT;
	}

	public void LoadAvg() {
		int activeTasks;
		int count = LOAD_FREQ;
		count -= ticks;
		if (count < 0) {
			activeTasks = count_active_tasks();
			do {
				CALC_LOAD(0, EXP_1, activeTasks);
				CALC_LOAD(1, EXP_5, activeTasks);
				CALC_LOAD(2, EXP_15, activeTasks);
				count += LOAD_FREQ;
			} while (count < 0);
		}
	}
}
```

ANEXO 7

```
private static final int SCALE = 10000;
private static final int ARRINIT = 2000;

private String calculePi(int digits){
      pi = new StringBuffer();
      arr = new int[digits + 1];
      maxArr = digits;
      for (i = 0;i <= maxArr; ++i)
            arr[i] = ARRINIT;
      for (i = maxArr; i > 0; i-= 14) {
            sum = 0;
            for (j = i; j > 0; --j) {
                  sum = sum*j + SCALE * arr[j];
                  arr[j] = sum % (j*2-1);
                  sum /= (j*2-1);

      }
            pi.append( carry + sum/SCALE);
            carry = sum % SCALE;
// Criação da tarefa que será executada
}
      return pi.toString();
}
```