Felipe César Pereira

IEC - Uma Arquitetura XML para Computação Colaborativa P2P

IM – NCE – UFRJ – Mestrado em Informática

Éber Assis Schmitz
Ph.D., Imperial College – Londres – 1980

Felipe Maia Galvão França
Ph.D., Imperial College – Londres – 1994

Rio de Janeiro
2005

IEC - Uma Arquitetura XML para Computação Colaborativa P2P

Felipe César Pereira

TESE SUBMETIDA AO CORPO DOCENTE DO NÚCLEO DE COMPUTAÇÃO E ELETRÔNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS DE INFORMAÇÃO.

Aprovada por:

_____

Prof. Éber Assis Schmitz, Ph.D.

_____

Prof. Felipe Maia Galvão França, Ph.D.

_____

Prof. Carlo Emmanoel Tolla de Oliveira, Ph.D.

_____

Profª Luigi Carro, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

JANEIRO DE 2005

FICHA CATALOGRÁFICA

P436 Pereira, Felipe César.
IEC – uma arquitetura XML para computação colaborativa P2P / Felipe César Pereira. – Rio de Janeiro, 2005.
xii, 80.; il.

Dissertação (Mestrado em Informática) – Universidade Federal do Rio de Janeiro, Instituto de Matemática, Núcleo de Computação Eletrônica, 2004.

Orientadores: Éber Assis Schmitz; Felipe Maia Galvão França

1. P2P – Teses. 2. GRID – Teses. 3. Sistemas Distribuídos – Teses. 4. Balanceamento de Carga – Teses I. Eber Assis Schmitz (Orient.). II. Felipe Maia Galvão (Orient.). III. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Núcleo de Computação Eletrônica. IV. Título

CDD

Agradecimentos

Aos meus pais, Helcio Bittencourt Pereira e Lucia Helena Pena Pereira, que sempre me apoiaram e me deram a base para que chegasse a esse ponto.

À minha mulher, Daniela Caroline Pereira da Silva, que sempre me deu força para que eu seguisse com o trabalho, aceitando ficar sozinha em diversas ocasiões para que pudesse terminar a dissertação.

Ao professor Éber Schmitz por ter orientado esse trabalho, sempre participando com sugestões e críticas para melhorá-lo cada vez mais, além do incentivo dado durante a graduação para meu ingresso no mestrado.

Ao professor Felipe França, meu co-orientador, que ajudou no fechamento do escopo e na formalização do trabalho, além das inúmeras idéias e críticas que contribuíram enormemente com o resultado final do trabalho.

A professora Maria Luiza que sempre incentivou seus alunos a continuarem estudando, em especial, na realização do mestrado.

Aos colegas da Embratel, em especial ao meu antigo e novo coordenador, Luiz Henrique e João Mendes, que sempre me ajudaram a conciliar o trabalho e o estudo, permitindo que eu flexibilizasse meu horário para que pudesse ter reuniões com meus orientadores durante a semana.

A todos os funcionários do MOT/NCE pelos finais de semana que utilizamos o laboratório para realização dos experimentos.

Em especial, aos amigos que sempre incentivaram, dando força para que eu prosseguisse e chegasse ao fim dessa etapa em minha vida.

Sumário

Lista de Siglas

API	APPLICATION PROGRAMMING INTERFACE
COM	COMPONENT OBJECT MODEL
CPU	CENTRAL PROCESSING UNIT
DTD	DOCUMENT TYPE DEFINITION
IP	INTERNET PROTOCOL
MIT	MASSACHUSETTS INSTITUTE OF TECHNOLOGY
OGSA	OPEN GRID SERVICE ARCHITECTURE
P2P	PEER TO PEER
QoS	QUALITY OF SERVICE
RCPS	RESOURCE CONSTRAINED PROJECT SCHEDULING
RCPSP	RCPS PROBLEM
SLA	SERVICE LEVEL AGREEMENT
SOAP	SIMPLE OBJECT ACCESS PROTOCOL
UDDI	UNIVERSAL DESCRIPTION, DISCOVERY AND INTEGRATION
WSDL	WEB SERVICES DESCRIPTION LANGUAGE
WWW	WORLD WIDE WEB
XML	EXTENDED MARKUP LANGUAGE

Índice de Figuras

Índice de Tabelas

Capítulo 1 - Introdução

1.1 – Apresentação

Com o crescente poder computacional disponível nas estações pessoais de trabalho, torna-se interessante utilizar tal capacidade para a realização de tarefas complexas que, quando possíveis de serem executadas, só rodam em máquinas de grande poder computacional, extremamente caras.

Este trabalho introduz uma nova arquitetura colaborativa P2P (*peer to peer*), a IeC (Infra-estrutura Colaborativa), visando sua utilização em estações de trabalho disponíveis em corporações e residências, com o propósito de se conseguir um considerável potencial computacional com um baixo custo, e que seja de fácil utilização e implantação por usuários não especialistas.

O IeC disponibiliza uma API para que desenvolvedores de aplicações P2P possam programar suas aplicações sem se preocupar com problemas de infra-estrutura, como localização do recurso desejado, tolerância a falhas e balanceamento de carga.

O uso de um mecanismo simples de escalonamento distribuído para o balanceamento de carga nos nós computacionais participantes é avaliado em termos da escalabilidade da arquitetura proposta e da qualidade do balanceamento. Tal foi obtido através de simulações usando-se como benchmark problemas do tipo RCPS (*Resource Constrained Project Scheduling*), buscando-se a solução ótima, através da enumeração de todos as possibilidades de escalonamento, e da solução por heurística, usando-se simulações tipo Monte Carlo.

1.2 – Motivação

A principal motivação para esse trabalho foram os projetos SETI@home [1] e The United Devices Cancer Research Project [2]. No primeiro, voluntários do mundo inteiro cedem seus PCs para ajudar na análise dos espectros captados pelo telescópio Arecibo, localizado em Porto Rico, procurando eventuais transmissões de civilizações inteligentes extraterrestres. No segundo, computadores ao redor do mundo emprestam CPU para ajudar na pesquisa de novas drogas contra o câncer. Em ambos os projetos,

estações de trabalho juntas conseguem poder computacional maior ou comparável a supercomputadores, com um custo menor. No SETI@home, atualmente consegue-se 15 TeraFLOPs a um custo de US$500.000,00. Um pouco mais do que o supercomputador IBM's ASCI White, que alcança 12 TeraFLOPs e custa US$110.000.000,00 [1].

Disponibilizar potência computacional a um baixo custo, e com alta escalabilidade, é uma importante área de estudo, visto que cada vez mais existe potência computacional espalhada em computadores pessoais, *personal digital assistants (*PDAs*)*, celulares e estações de trabalho. Tal potência pode ser utilizada para baratear pesquisas importantes para toda a sociedade, como cura de doenças graves, ou em um novo modelo econômico, onde qualquer indivíduo pode conectar seu computador ou PDA e ter potência computacional em um modelo de cobrança tal como outros bens de consumo, como luz, água, gás e energia elétrica.

1.3 – Objetivo

Diversas aplicações P2P já foram construídas, porém, cada uma implementa seus próprios protocolos e serviços de infra-estrutura para rodarem no ambiente P2P. O trabalho desenvolvido aqui possui os seguintes objetivos: criar uma infra-estrutura *peer-to-peer* (P2P) de fácil utilização, baseada em padrões de mercados como COM e XML, permitindo a construção e o estudo de aplicações colaborativas de forma simples; realizar uma avaliação preliminar desta arquitetura. Tal infra-estrutura fornece, através de uma API, os serviços necessários para que aplicações P2P possam ser desenvolvidas sem que os seus desenvolvedores precisem lidar com as particularidades desse ambiente. Uma especial atenção foi dada ao escalonador da arquitetura, visto que o sucesso de aplicações colaborativas depende desta ser escalável.

1.4 – Do que este trabalho não trata

A arquitetura desenvolvida não tem ambição de implementar todos aspectos necessários para o funcionamento das aplicações colaborativas em um ambiente não controlado como, por exemplo, segurança e tratamento contra sabotagem.

A avaliação realizada é preliminar, portanto, não abrange todos as classes de aplicações e situações que um ambiente P2P pode ser submetido, e nem avalia todas as funcionalidades da arquitetura.

1.5 – Contribuições do trabalho

Como resultado do trabalho, foi codificada uma arquitetura P2P baseada em XML que oferece os serviços básicos para que aplicações colaborativas P2P possam ser desenvolvidas sem que o desenvolvedor precise lidar com os serviços de infra-estrutura. Apresentamos uma análise dessa arquitetura no artigo "Uma Arquitetura XML para Computação Colaborativa P2P" [3], publicado no WSCAD2004 – Quinto Workshop em Sistemas Computacionais de Alto Desempenho. A arquitetura criada serviu também como plataforma para o estudo publicado no artigo "Uma Abordagem *Branch and Bound* para o RCPSP em um Ambiente de Computação Colaborativa" submetido e aceito para publicação no *XII Congreso Latino Iberoamericano de Investigación de Operaciones* (CLAIO 2004) [4].

1.6 – Estrutura do trabalho

O texto a seguir está organizado da seguinte forma, o Capítulo 2 apresenta os conceitos de GRID e P2P. O Capítulo 3 fala sobre as aplicações colaborativas, e dos modelos e características utilizados para distribuí-las em Grids e ambientes P2P. O Capítulo 4 apresenta a arquitetura IeC, descrevendo em detalhes todas as suas funcionalidades e módulos. O Capítulo 5 descreve o experimento utilizado para testar o IeC, o Capítulo 6 avalia-o, e o Capítulo 7 conclui o trabalho, sugerindo tópicos para trabalhos futuros.

Capítulo 2 – Arquiteturas distribuídas

Este capítulo trata dos dois principais modelos utilizados para se disponibilizar recursos computacionais distribuídos geograficamente. Tanto Grids como P2P possuem ambições semelhantes, porém, com caminhos evolucionários diferentes [18].

2.1 – GRID

O aumento da popularidade da Internet junto com o aumento computacional dos computadores e de redes rápidas, como bens de consumo de baixo custo, estão mudando a forma com que fazemos computação [16]. Essas novas tecnologias possibilitam o agrupamento de uma variada quantidade de recursos distribuídos geograficamente, tais como supercomputadores, sistemas de armazenamento, fontes de dados, dispositivos especiais, e serviços, que podem ser utilizados como um único recurso. Esse novo paradigma está sendo chamado de *"grid" computing*. O Grid tem a principal função de coordenar esse conjunto virtual de recursos em larga escala [6].

O Grid é análogo, em alguns aspectos, a infra-estrutura para geração e distribuição de energia elétrica por volta de 1910 [5]. Nesta época, a geração de energia era possível e novos dispositivos dependentes da eletricidade estavam sendo construídos. A verdadeira revolução não foi a eletricidade, mas sim, a usina elétrica e linhas de transmissão. Juntos, tais tecnologias permitiram o acesso da população a um conjunto de serviços confiáveis e de baixo custo, derivados da eletricidade.

Da mesma forma, o termo Grid Computing refere-se à infra-estrutura que tornará possível o acesso, de uma forma confiável e com um baixo custo, aos recursos computacionais mencionados.

"Grid computing está no caminho certo para resolver um problema chave em nosso mundo da computação distribuída: a descoberta e o uso coordenado dos serviços distribuídos, que podem ser implementados por recursos dinâmicos e voláteis." [7]

2.1.1 Ambientes distribuídos convencionais x Grids

Uma aplicação distribuída convencional assume um conjunto de nós computacionais que formam uma máquina paralela virtual. O conjunto consiste de PCs, estações de trabalho, e possivelmente supercomputadores, aos quais o usuário possui

acesso (login e senha válidos) a todos eles. Além disso, o conjunto virtual de nós pode ser considerado estático, pois este muda muito raramente. O tamanho de tais sistemas é da ordem de 10-100 nós. [6]

Em contraste, grids computacionais são baseados em uma grande quantidade de recursos compartilhados. Grids assumem um conjunto virtual de recursos e não de nós computacionais. Todos esses recursos, tipicamente, existem dentro de nós que estão geograficamente espalhados, e abrangem diversos domínios administrativos. Nos grids, esse conjunto de recursos é dinâmico e variado, pois podem ser adicionados e retirados a qualquer momento, de acordo com a vontade de seus donos, e sua performance, ou carga, muda freqüentemente no decorrer do tempo. Por todos esses motivos, o usuário tem muito pouco ou nenhum conhecimento a priori sobre o tipo, estado e atributos dos recursos que constituem o conjunto [6]. A tabela abaixo apresenta as principais diferenças entre os dois sistemas:

Tabela 1 - Comparação entre ambientes distribuídos convencionais e Grids [6].

Ambientes distribuídos convencionais	***Grids***
Um conjunto virtual de nós computacionais.	Um conjunto virtual de recursos.
O usuário tem acesso a todos os nós do conjunto.	O usuário tem acesso ao conjunto, mas não aos *sites* de forma individual.
Acesso a um nó significa acesso a todos os recursos do nó.	Acesso a um recurso pode ser restrito.
O usuário está ciente das capacidades e atributos dos nós.	O usuário possui pouco conhecimento sobre cada *site.*
Os nós pertencem a um único domínio confiável.	Recursos abrangem diversos domínios confiáveis.
Elementos no conjunto são da ordem de 10-100, mais ou menos estático.	Elementos no conjunto são da ordem de 1000-10000, dinâmicos.

2.1.2 – Impactos do GRID

A história das redes de computadores mostra que transformações da ordem de grandeza como a apresentada pela tecnologia de grid computing possibilitam aplicações

revolucionárias, que fazem uso da nova tecnologia, gerando, como conseqüência, motivação para um sucessivo melhoramento tecnológico [5].

Existem evidências que outra revolução tecnológica está por vir. A capacidade dos computadores e das redes continuam aumentando em ritmo acelerado. Anos de pesquisa em metacomputação criaram conhecimentos sólidos em novas aplicações que fazem uso de grande computação e de redes rápidas. O tempo parece oportuno para uma transição dos dias heróicos da metacomputação para uma maior integração entre os diversos grids através de interfaces padronizadas. Tal transição tornará as atuais aplicações que rodam em grids em rotina, e permitirá aos programadores explorarem teraFLOPs de computação e petabytes de espaço para armazenamento, interconectados por redes de gigabits de banda, levando-nos a uma nova geração de aplicações [5].

Como exemplo de tais aplicações, podemos citar aplicações financeiras, que poderão utilizar análises baseadas em simulações de Monte Carlo para calcular o futuro valor dos investimentos realizados, utilizando os teraFLOPs de computação fornecida pelo grid. Urbanistas e arquitetos poderiam submeter seus projetos a computadores gráficos poderosos e base de dados presentes no grid, com finalidade de criar realidades virtuais baseadas nestes projetos, explorando questões como: qualidade acústica, consumo de energia, e eficiência da iluminação [5].

2.1.3 – Estilos de GRID

Podemos dividir o universo dos Grids, conforme o modo que estes gerenciam os recursos computacionais existentes, tais como: escalonamento dos processos, largura de banda fornecida para acesso a rede e capacidade de disco. Krauter em seu estudo sobre a taxonomia dos Grids, realizou a seguinte divisão [8]:

- Grids computacionais – são sistemas que fornecem uma capacidade computacional maior, para uma determinada aplicação, do que qualquer máquina que pertence ao sistema. Pode ser subdividido em: supercomputação distribuída, cujo objetivo é diminuir o tempo de processamento de uma única aplicação; ou de alta vazão, que possui a finalidade de realizar uma maior quantidade de trabalhos possíveis em um determinado intervalo de tempo [5] [8];
- Grid de dados – são sistemas que fornecem infra-estrutura para consolidação de nova informação a partir de repositórios de dados, como, por exemplo, data

warehouses. Provêem mecanismos de segurança e redundância, para armazenamento de dados em sites distribuídos geograficamente [5] [7] [8];

- Grid de serviços – são sistemas que fornecem serviços que não são providos por nenhuma máquina individualmente. Pode ser dividido em colaborativo, sobre demanda e multimídia. O colaborativo conecta pessoas e aplicações a um espaço de trabalho. O sobre demanda, agrega recursos do Grid dinamicamente para prover novos serviços. O multimídia fornece infra-estrutura para aplicações multimídias de tempo real [8].

2.1.3.1 Organização das máquinas

Conforme a organização das máquinas no Grid, podem-se utilizar estruturas centralizadas e descentralizadas. Nas centralizadas, uma única máquina, ou um conjunto designado, é responsável por fazer a distribuição das tarefas no sistema, não sendo adequadas, pois não provêem a escabilidade necessária para suportar a natureza do Grid, que pode ter o número de máquinas integrantes reduzidos ou ampliados a qualquer momento. As estruturas descentralizadas, adequadas aos sistemas Grid, podem ser divididas nas seguintes categorias [8]:

- Plana – Todas as máquinas se comunicam com qualquer outra sem a necessidade de uma intermediária;
- Hierárquica – As máquinas são agrupadas por níveis, onde as de um determinado nível somente se comunicam com as do nível imediatamente inferior ou superior, ou com as de mesmo nível;
- Celular – Nesta estrutura, as máquinas são distribuídas em células. Dentro de uma mesma célula, a comunicação entre é feita de forma plana. Com outras células o envio de mensagens é intermediado por máquinas designadas como elementos de fronteira, que são as únicas máquinas visíveis para outras células. Estas podem se organizar de forma plana, ou hierárquicas.

2.1.3.2 Organização dos recursos

Um aspecto importante do Grid é como os seus recursos, que constituem um conjunto global de recursos nomeados (*namespaces*), são organizados. Isto irá influenciar nos protocolos utilizados para gerenciamento dos recursos e na estratégia utilizada para descobri-los [8].

Em uma abordagem baseada em esquema, os dados relacionados aos recursos são descritos em uma linguagem descritiva contendo suas restrições de integridade. Em alguns sistemas, a linguagem descritiva está associada a uma linguagem de consulta [8].

Na abordagem orientada a objetos, as operações disponíveis para os recursos também são descritas como parte do modelo de recursos [8].

As seguintes organizações de namespaces são encontradas nos Grids [8]:

- Relacional – os recursos são divididos por relações, utilizando conceitos dos gerenciadores de banco de dados relacionais para indicar diversas relações entre tuplas;
- Hierárquica – organiza os recursos através de hierarquias, tipicamente arrumados em estruturas de redes físicas, ou lógicas;
- Relacional/ hierárquica (híbrida) – as relações são organizadas por hierarquia, com finalidade de distribuí-las nas diversas máquinas do Grid;
- Baseada em grafo – Usa nós e ponteiros para representar a associação dos recursos em um grafo. Tipicamente utilizado em uma abordagem orientada a objetos.

2.1.3.3 Qualidade de serviço

Grids também podem fornecer aspectos de qualidade de serviço (QoS, *Quality of Service*) para diversos aspectos, tais como: largura de banda, armazenamento e computação. Quando existe QoS para um grid, é necessário estabelecer os níveis de serviços acordados (SLA, *Service Level Agreement*) para mensurar a qualidade desejada. Reserva de recursos é considerado o atributo fundamental para QoS, portanto, Grids que fornecem níveis de QoS na submissão de trabalhos, porém, não fornece meios avançados para reserva de recursos, fornecem uma solução parcial para QoS. Existem duas partes para o controle de QoS: controle de admissão, e contrato. Na primeira, o Grid deve verificar se o QoS solicitado poderá ser entregue. No segundo, se este não viola o SLA definido. QoS para os Grids é dividido em [8]:

- Relaxado – O Grid fornece controle de amissão de QoS, porém, não é capaz de controlar contratos de SLA;
- Estrito – Todos os nós do Grid conseguem gerenciar os contratos de SLA.

2.1.3.4 Descoberta e disseminação dos recursos

A descoberta e a disseminação dos recursos são funcionalidades complementares, onde a descoberta é o ato de uma aplicação procurar os recursos necessários para seu funcionamento, e a disseminação, ocorre quando um recurso tenta localizar aplicações que podem se beneficiar dele. O método utilizado para disseminação e descoberta dos recursos estará diretamente associado à forma como eles estão organizados. A disseminação ocorre através da atualização dos descritores dos recursos, e a descoberta ocorre através de consultas a esses descritores. Conforme a forma de atualização e consulta podemos agrupar estilos comuns a diversos Grids. [8]

Quanto à descoberta, temos [8]:

- Consultas (*Queries*) – A aplicação realiza consulta na base de dados que descreve os recursos na rede. A base de dados pode estar organizada de forma distribuída ou centralizada, caracterizando duas possibilidades de consultas;
- Agentes – Nessa abordagem, programas agentes navegam pelos nós a procura dos recursos desejados, ou monitoram passivamente qualquer alteração da configuração dos recursos.

Quanto à disseminação, temos [8]:

- Batch – De tempos em tempos, máquinas do grid enviam informações sobre seus recursos para o Grid;
- Online – Sempre que ocorre uma mudança de recursos em uma máquina, esta atualiza imediatamente a base de recursos do Grid.

2.1.3.5 Escalonamento

O escalonador de tarefas de um Grid tem o papel de definir quando, onde e em que ordem os trabalhos submetidos ao Grid irão rodar. Pode ser organizado de três principais maneiras [8]:

- Centralizado – Existe apenas um controlador para escalonar tarefas no sistema inteiro. Essa abordagem possui algumas vantagens como: facilidade de gerenciamento e distribuição, e possibilidade de alocar os trabalhos nas máquinas onde os recursos necessários estão localizados (co-alocação de recursos). Porém, esta abordagem não é escalável, não possui tolerância a falhas, e não permite a utilização de várias políticas de escalonamento;

- Hierárquico – Os controladores são organizados de forma hierárquica. Essa organização mantém as vantagens do centralizado e resolve os problemas de tolerância à falha e escalabilidade, porém, não resolve o problema de não permitir múltiplas políticas de escalonamento;
- Descentralizado – Resolve problemas como tolerância a falha, escalabilidade, permissão de uso de várias políticas, e permite autonomia aos *sites*, porém, insere novos desafios, tais como a dificuldade de gerência, e co-alocação de recursos.

Devido ao grande volume de máquinas envolvidas, é inviável conhecer o estado de cada máquina no Grid, por tanto, para realizar o escalonamento das tarefas, é necessário estimar tal estado. A forma como tal estimativa é feita, pode ser dividida em [8]:

- Sem histórico – A estimativa é feita apenas a partir do trabalho sendo executado e informações sobre a situação dos recursos. Podem ser utilizados heurísticas, modelos de precificação, e aprendizado de máquina (*machine learning*) para tal;
- Com histórico – Utiliza informações históricas como, por exemplo, rodadas anteriores de uma mesma aplicação, para realizar a estimativa. Pode ser utilizado heurísticas e modelos de distribuição de probabilidade para tal.

Uma característica importante ao escalonador é a possibilidade de re-escalonar os trabalhos. Essa possibilidade é feita para atender a política vigente, podendo ser para aumentar a vazão de trabalhos, aumentar a utilização dos recursos, ou qualquer outra métrica. As formas de re-escalonar são [8]:

- Batch/ Periódico – De tempos em tempos, o escalonador avalia a execução dos trabalhos e re-organiza a fim de atender as métricas mencionadas;
- Online – Qualquer evento do sistema, ou requisição de um recurso, dispara o processo de re-organização.

O processo periódico potencialmente implementa uma melhor eficiência na utilização dos recursos, porém, não consegue implementar QoS estrito, visto que no intervalo de re-escalonamento, pode ocorrer uma violação de SLA [8].

2.1.4 Principais GRIDs

Segue abaixo uma tabela contendo os principais Grids com suas características, conforme a taxonomia apresentada [8] na seção anterior.

Tabela 2 - Principais Grids e suas escolhas de arquitetura [8].

Sistema	*Tipo de Grid*	*Recursos*	*Escalonador*
2K	Serviço sobre demanda	Modelo de objetos extensível, namespace baseado em grafo, QoS relaxado para rede, descoberta por agentes, disseminação Online	Hierárquico para recursos de rede e descentralizado para outros recursos
AppLes	Computacional alta vazão	Modelo de recursos provido pelo Globus, Legion ou NetSolve	Hierárquico. Estimativa utilizando histórico e heurística. Re-escalonamento Online
Bond	Serviço sobre-demanda Plano	Modelo de objetos extensível, namespace baseado em grafo, QoS estrito, descoberta por agentes e disseminação periódica.	Descentralizado. Estimativa utilizando histórico e modelos de precificação. Re-escalonamento Online
Condor	Computacional Plano	Modelo baseado em esquema extensível, namespace híbrido, sem QoS, descoberta através de consultas centralizadas, disseminação periódica	Centralizado.
Darwin	Serviço multimídia Hieráquico	Modelo de esquema fixo. Namespace baseado em grafo, QoS estrito	Hierárquico
European Data Grid	Data Hierárquico	Modelo de esquema extensível, namespace hierárquico, QoS relaxado, descoberta através de consultas distribuídas, disseminação periódica	Hierárquico
Globus	Diversos Celular hierárquico	Modelo de esquema extensível, namespace hierárquico, QoS relaxado, descoberta por consultas distribuídas, disseminação periódica	Descentralizado. Provido externamente pelo AppLes e Nimrod/G
Javelin	Computacional Hierárquico	Modelo de objetos fixos, namespace baseado em grafo, QoS relaxado, descoberta através de consultas distribuídas, disseminação periódica	Descentralizado
GOPI	Serviço multimídia Plano	Modelo de objetos extensíveis, namespace baseado em grafo, QoS estrito	Descentralizado

Legion	Computacional Hierárquico	Modelo de objetos extensíveis, namespace baseado em grafo, QoS relaxado, descoberta através de consultas distribuídas, disseminação periódica	Hierárquico
MOL	Computacional Hierárquico celular	Modelo de esquema extensível, namespace hierárquico, descoberta através de consultas distribuídas, disseminação periódica	Descentralizado
NetSolve	Computacional Hierárquico	Modelo de esquema extensível, namespace hierárquico, QoS relaxado, descoberta através de consultas distribuídas, disseminação periódica	Descentralizado
Nimrod/G	Computacional alta vazão Celular hierárquico	Modelo de esquema extensível, namespace hierárquico, QoS relaxado, descoberta através de consultas distribuídas, disseminação periódica	Hierárquico e descentralizado. Estimativa baseada em histórico, utilizando modelos de precificação
Ninf	Computacional Hierárquico	Modelo de esquema fixo, namespace relacional, sem QoS, descoberta através de consultas centralizadas, disseminação periódica	Descentralizado
PUNCH	Computacional Hierárquico	Modelo de esquema extensível, namespace híbrido, QoS relaxado, descoberta através de consultas distribuídas, disseminação periódica	Hierárquico e descentralizado. Estimativa baseada em histórico, utilizando aprendizado de máquina

2.1.5 – Open Grid Service Architecture

A arquitetura *Open Grid Service Architecture* (OGSA) [12] [7], em essência, une Web Services [12] com os protocolos de Grid, realizando um grande progresso na definição de interfaces para os serviços fornecidos pelos Grids. A arquitetura é construída sobre padrões de mercado como SOAP, WSDL e UDDI, possibilitando que o Grid forneça Web Services de forma padronizada.

2.2 – P2P

A computação *peer-to-peer* — P2P — pode ser definida como uma classe de aplicações que, baseadas em um controle descentralizado, i.e., distribuído entre todos as máquinas participantes, usufruem de recursos de armazenamento, processamento, conteúdo, e presença humana disponíveis nas fronteiras da Internet [18]. Segundo o estudo de taxonomia de Krauter [8], P2P pode ser classificada como uma forma de organização plana das máquinas, em que todas se comunicam com qualquer outra (overlay) sem a necessidade de uma máquina servidor intermediária. Aplicações como o Gnutella [19] popularizaram o P2P na WWW onde, dado que operar em um ambiente de conectividade instável e endereços IPs imprevisíveis, projetos P2P são, geralmente, independentes de DNS e de servidores centrais. O Gnutella é um sistema P2P puro, ou seja, com controle totalmente descentralizado em que todos os nós são iguais para o sistema, destinado a troca de arquivos pela Internet.

O *survey* "Peer-to-peer computing" [17] é a principal fonte das informações contidas nesse capítulo.

2.2.1 Objetivos

P2P é freqüentemente confundido com outros termos, tais como computação distribuída tradicional, grid computing, e ad-hoc networking. Para melhor definir P2P, essa seção introduz os objetivos do P2P, que são:

- **Redução do custo de compartilhamento** – Sistemas centralizados que servem muitos clientes, tendem a ter um custo extremamente alto no nó central, o servidor. Uma arquitetura P2P pode ajudar a espalhar esse custo entre todos os *peers* como, por exemplo, o espaço para armazenamento dos arquivos compartilhados. Em sistemas como o Napster [20] e o Gnutella , o espaço é fornecido pelos *peers*;
- **Aperfeiçoar a escalabilidade/ confiabilidade** – Com a carência de uma autoridade central para os nós, aperfeiçoar a escalabilidade e a confiabilidade é um importante objetivo. Como resultado, inovações em algoritmos na área de descoberta e disseminação de recursos [27] é uma importante área de pesquisa;
- **Agregação de recursos e interoperabilidade** – Numa abordagem descentralizada, cada nó no sistema traz consigo uma certa quantidade de recursos como potência computacional, ou espaço em disco. Aplicações que se beneficiam dessa enorme

quantidade de recursos, tais como simulações com uso intensivo de computação, ou sistemas de arquivos distribuídos, naturalmente dependem de uma estrutura P2P que seja capaz de agregar tais recursos;

- **Incremento de autonomia** – Em muitos casos, usuários de sistemas distribuídos relutam em permanecer em provedores de serviços centralizados. Eles preferem que dados e trabalhos de seu interesse sejam feitos localmente. Sistemas P2P suportam esse nível de autonomia simplesmente porque requerem que o nó local trabalhe a favor de seus usuários. Os vários sistemas de compartilhamento de arquivos existentes como o Napster, Gnutella, e FreeNet [21] são exemplos deste princípio. Em todos os casos, os usuários são capazes de pegar arquivos que não poderiam estar disponibilizados em um servidor central devido a restrições de licença. Entretanto, indivíduos autônomos rodando seus próprios servidores estão aptos para compartilhar os arquivos porque é mais difícil achá-los do que a um operador de um servidor central;
- **Anonimato/ privacidade** – Relacionada à autonomia está a noção de anonimato e privacidade. Um usuário pode não querer que ninguém ou nenhum provedor de serviço conheça sobre ele ou seu envolvimento com o sistema. Com um servidor central, é difícil ter certeza do anonimato porque, tipicamente, o servidor será capaz de identificar o cliente, pelo menos pelo endereço da Internet. Implementanto uma estrutura P2P em quais atividades são feitas localmente, usuários podem evitar ter de prover qualquer informação sobre eles mesmos para outra pessoa. FreeNet é um exemplo de como o anonimato pode ser construído em aplicações P2P. Este usa um esquema que repassa mensagens para ter certeza que o requerente do serviço não possa ser facilmente rastreado.
- **Dinamismo** – Sistemas P2P assumem que o ambiente computacional é altamente dinâmico. Isto é, recursos, tais como nós computacionais, entrarão e deixarão o sistema continuamente. Quando uma aplicação pretende rodar em um ambiente altamente dinâmico, o P2P se encaixa perfeitamente. Programas de comunicação instantânea são usados para informar aos usuários quando as pessoas, as quais eles desejam se comunicar, estão disponíveis. Sem esse suporte, usuários precisariam testar a disponibilidade das pessoas, enviando mensagens periódicas para estas. Do mesmo modo, aplicações computacionais distribuídas como o SETI@Home precisam se adaptar a mudança dos participantes. Estas têm de reenviar trabalhos computacionais para outros participantes para ter certeza que estes não foram

perdidos se participantes antigos deixaram a rede enquanto realizavam um passo computacional.

- **Habilitando comunicação ad-hoc e colaboração** – Relacionado ao dinamismo está a noção de suporte à ambientes ad-hoc. Chama-se ambiente ad-hocs aqueles onde os membros vão e voltam de acordo com sua localização física ou em seus interesses correntes. Aplicações desse tipo novamente se encaixam na estrutura P2P, pois, estas normalmente não são construídos em cima da infra-estrutura estabelecida.

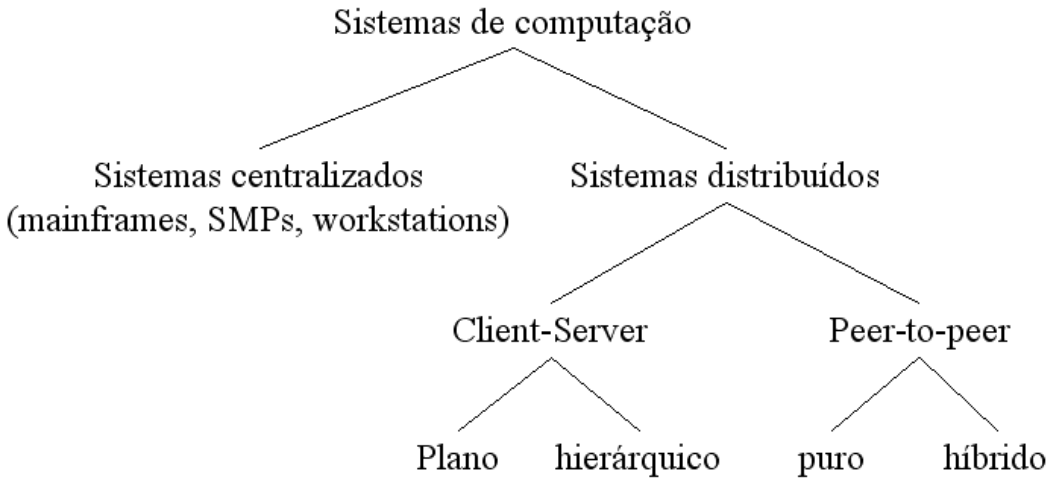

Figura 1 - Taxonomia dos sistemas de computação na perspectiva P2P [17].

2.2.2 Taxonomia

Na perspectiva P2P, a taxonomia dos sistemas de computação é apresentada na Figura 1. Todos os sistemas distribuídos podem ser classificados no modelo *client-server*, ou no modelo P2P. O primeiro pode ser plano, onde todos os clientes apenas se comunicam com um único servidor central (possivelmente replicado para ampliar a confiabilidade), ou pode ser hierárquico, ampliando a escalabilidade. No modelo hierárquico, os servidores de um nível agem como clientes dos servidores do nível superior. O modelo P2P pode ser puro ou híbrido. Em um modelo puro, não existe um servidor central. Gnutella e Freenet são exemplos de modelos puros. No modelo híbrido, um nó no sistema primeiro conecta-se a um servidor a fim de buscar meta informação, como a identidade de um nó que possua a informação desejada, ou verificar as credenciais de segurança. Após esse passo, uma comunicação P2P é feita. Napster é

um exemplo do modelo híbrido. Também existem soluções intermediárias em que supernós contêm informações que outros não possuem. Outros nós procuram tais informações nos supernós. O KaZaa [22] utiliza essa abordagem.

Tabela 3 - Comparação de soluções. Células mais escuras representam os pontos mais fortes [17].

Requerimentos	Tipo de Sistema		
	Centralizado	**Cliente-Servidor**	**Peer-to-Peer**
Descentralização	Não disponível	Alta	Muito alta
Comportamento ad-hoc	Não disponível	Médio	Alto
Custo para autonomia	Muito alto	Alto	Baixo
Anonimato	Não disponível	Médio	Muito alto
Escalabilidade	Baixa	Alta	Alta
Performance	Individualmente alta	Médio	Individualmente baixo
	Coletivamente baixa		Coletivamente alta
Tolerância à falhas	Individualmente alta	Média	Individualmente baixo
	Coletivamente baixa		Coletivamente alta
Auto-organização	Médio	Médio	Médio
Transparência	Baixa	Média	Média
Segurança	Muito alta	Alta	Baixa
Interoperabilidade	Padronizada	Padronizada	Em progresso

A Tabela 3 apresenta uma tabela comparativa entre os sistemas centralizados, clientes-servidores e P2Ps, indicando os pontos fortes e fracos em cada tipo de sistema.

2.2.3 Descentralização

O modelo P2P questiona a sabedoria de processar dados apenas em servidores centrais, acessando seu conteúdo via protocolos do tipo *request-response* [17]. No modelo cliente-servidor tradicional, a informação é concentrada nos servidores centrais e distribuída para os clientes pela rede. Os clientes agem principalmente como interface

homem-máquina. Tais sistemas centralizados são ideais para alguns tipos de aplicações e tarefas. Por exemplo, o gerenciamento de credenciais de segurança é feito de forma mais fácil em um servidor central. Entretanto, a topologia dos sistemas centralizados inevitavelmente causa gargalos e desperdício de recursos. Embora o custo e a performance dos hardwares venham melhorando, repositórios centrais ainda são mais caros de se manter.

Um dos maiores poderes da idéia da descentralização é que cada nó é um igual participante para o sistema. Isto torna difícil a implementação do modelo P2P na prática, porque não existe nenhum elemento central que enxergue todos os nós na rede e os recursos que eles provêem. Este é o motivo que leva a maioria dos sistemas P2P optarem por abordagens híbridas. Nos sistemas completamente descentralizados como o Freenet e o Gnutella, os nós conectam-se a rede estabelecendo conexão à pelo menos um nó que já se encontra na rede. Portanto, os novos nós precisam ter um ponto de partida, o que é geralmente feito mantendo-se uma lista com o endereço dos nós freqüentemente conectados. Após conectar a um nó do sistema, o novo nó pode encontrar outros e armazená-los localmente.

Um imediato benefício da descentralização é o aumento da escalabilidade. Como não existem pontos centrais, e todos os nós são iguais para o sistema (nas abordagens puras), todo os nós dividem o trabalho de coordenação e sincronização entre os nós, logo, não existem gargalos. Entretanto, não se pode atingir uma boa escalabilidade com o ônus da degradação de outras funcionalidades. Uma forma de evitar este problema é utilizar uma abordagem híbrida, mantendo-se algumas das funcionalidades do sistema centralizadas. Alguns sistemas P2P para requisitar um recurso devem enviar cegamente uma mensagem para os outros nós, esperando receber alguma resposta. Isto pode tornar o tempo para se localizar um recurso demasiadamente grande. Além disso, a busca pode falhar mesmo quando o recurso procurado existe, tornando o comportamento do sistema não determinístico. Portanto, técnicas para busca de recursos em um ambiente completamente descentralizado [27] [28] de forma eficiente são extremamente importantes.

Capítulo 3 – Aplicações colaborativas

Idealmente, as aplicações para rodarem em sistemas P2P, ou em GRIDs, devem ser paralelas, divididas em partes totalmente independentes, e possuírem unidades de trabalho de tamanho razoável, ou seja, granularidade grossa. Tais características visam a garantir a eficiência do sistema como um todo.

Na maior parte das vezes, os recursos utilizados pela aplicação estão em máquinas distintas, conectadas em redes que não foram desenvolvidas para apoiar o processamento paralelo, tais como a internet, que possui largura de banda limitada e alta latência. Portanto, o uso de comunicação excessiva deve ser evitado, pois, seu uso pode fazer com que o sistema gaste mais tempo na comunicação do que no trabalho efetivo nos processos, ou seja, operações de envio de trabalhos para máquinas na rede e troca de mensagens entre estas devem ser usadas com cautela a fim de não se degradar à performance do sistema. Os processos devem possuir uma baixa sincronização, ou nenhuma, e terem um tamanho que faça com que o tempo de execução destes torne o tempo de comunicação gasto, desprezível.

3.1 Master-Worker

Pelas razões anteriormente mencionadas, o paradigma Master-Worker [9] [17] [24] é ideal para as aplicações desenvolvidas para rodarem nos ambientes P2P e Grids. O paradigma consiste em duas entidades, o mestre e os trabalhadores. O mestre é responsável em decompor o problema em vários pedaços, distribuir estes entre o conjunto de trabalhadores, garimpar os diversos resultados e, finalmente, uni-los para alcançar a solução do problema. Cada trabalhador recebe mensagens contendo a tarefa que deve ser feita, realiza o trabalho, e envia uma mensagem contendo o trabalho feito.

Uma estratégia utilizada por Lourenço [25], na busca de soluções ótimas para problemas RCPS, foi tornar os trabalhadores em mestres e re-dividir o trabalho recebido entre outros trabalhadores, o que pode ocorrer de forma recorrente de acordo com a necessidade de se diminuir o tamanho do grão. Desta forma é criada uma estrutura hierárquica de Master-Worker (Figura 3). Quando o trabalhador que se tornou mestre recebe as respostas de seus trabalhadores, combina-as para formar sua resposta para o serviço solicitado por seu mestre, e a envia.

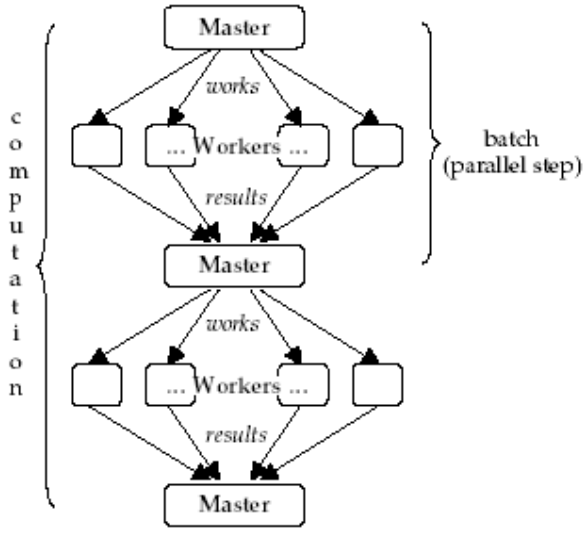

Figura 2 - O modelo Master-Worker de computação [9].

3.2 Exemplo de aplicações Master-Worker

3.2.1 Força bruta

Qualquer aplicação que requeira força bruta e cujo espaço de busca possa ser dividido pode se beneficiar de Grids e ambientes P2P através do paradigma Master-Worker. Em tais aplicações pode-se dividir o espaço de busca em diversas partes, e instruir os trabalhadores a procurarem a solução na parte recebida. O trabalhador que encontrar a resposta para a busca envia-a para o mestre. Desta forma, consegue-se varrer o espaço de busca de forma paralela.

Aplicações que se encaixam nesta categoria possuem inúmeras vantagens. Primeiro, são simples de serem implementadas porque para a maioria as aplicações, o mestre não tem de fazer nada além de dividir o espaço de busca e esperar que algum trabalhador ache a solução. Quando o espaço a ser procurado pode ser parametrizado, como, por exemplo, enviando-se a posição inicial e final, a comunicação entre os mestres e os trabalhadores é pequena, possibilitando que aplicações desse tipo sejam extremamente eficientes.

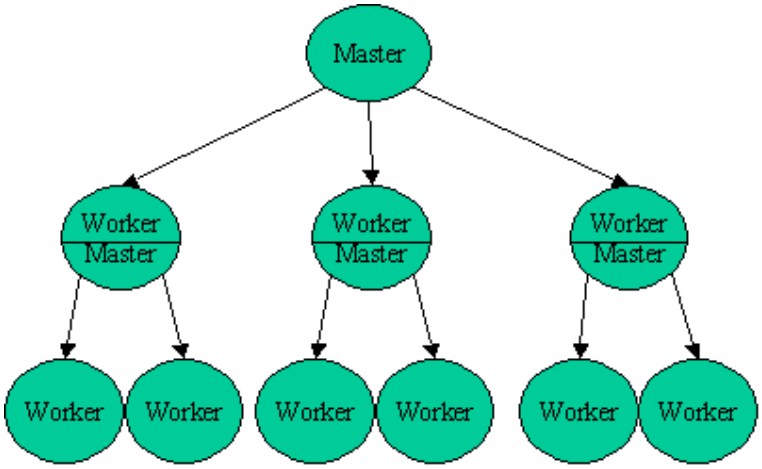

Figura 3 - Master-Worker com hierarquia.

3.2.2 Algoritmos genéticos

Outro tipo de aplicação que pode ser beneficiada pelo paralelismo do Grid são as que utilizam algoritmos genéticos [9] [26]. Estes algoritmos resolvem buscas e problemas de otimização através do princípio de seleção natural. Funcionam mantendo uma população de cromossomos, dos quais cada um representa uma possível solução do problema, e aplicam operações a essa população. Como resultado, obtém-se a simulação do processo evolutivo.

Tabela 4 - Algoritmo genético básico.

Início	Gera um conjunto aleatório de *n* cromossomos (possíveis soluções para o problema) para a população.
Aptidão	Avalia a aptidão *f(x)* de cada cromossomo *x* da população.
Nova população	Gera uma nova população, repetindo os passos seguintes até que a nova população esteja completa.
Fertilização	Seleciona dois pais da população de acordo com suas aptidões (quanto melhor a aptidão, maior a chance de ser selecionado).
Cruzamento	Com uma probabilidade de cruzamento, cruza os pais para formar a nova prole.
Mutação	Com uma probabilidade de mutação, causa mutação na prole.

Seleção	Coloca a prole na nova população, se de acordo com o critério de sobrevivência.
Troca	Substitui a população anterior com a nova.
Teste	Verifica se o critério de saída foi atingido. Se sim, pára e retorna a melhor solução presente na população.
Laço	Volta para Aptidão

O algoritmo básico da Tabela 4 pode ser escrito no paradigma Master-Worker. Em uma possível forma, o mestre executa os passos Início e Aptidão, envia a população gerada para os trabalhadores e espera um conjunto de cromossomos *x* dos trabalhadores para recompor a população *n*, onde *x* é igual a *n* dividido pelo número de trabalhadores.

3.2.3 Web-crawler

Apesar do Grid ser um sistema que coordena diversos tipos de recursos, o uso computacional é o exemplificado na maioria das vezes. Um exemplo de utilização do Grid para se ter acesso a recursos de comunicação é uma aplicação de web-crawler, que segue recursivamente links encontrados em páginas na internet, com finalidade de procurar páginas que atendam uma condição (palavras-chave, por exemplo) ou apenas catalogar a hierarquia das páginas.

Sarmenta [9] construiu uma aplicação utilizando esta idéia e realizou um experimento no MIT. Nele, uma máquina, utilizando um link lento de comunicação, executa a aplicação mestre e outras máquinas, ligadas a um link de acesso rápido, executam os trabalhadores.

Esse mesmo aplicativo poderia ser escrito com o paradigma master-worker com hierarquia utilizado por Lourenço [25] (Figura 3). Nesta abordagem, sempre que existe bifurcação, o trabalhador escolhe um caminho e envia pedidos para outros trabalhadores utilizarem os demais caminhos possíveis.

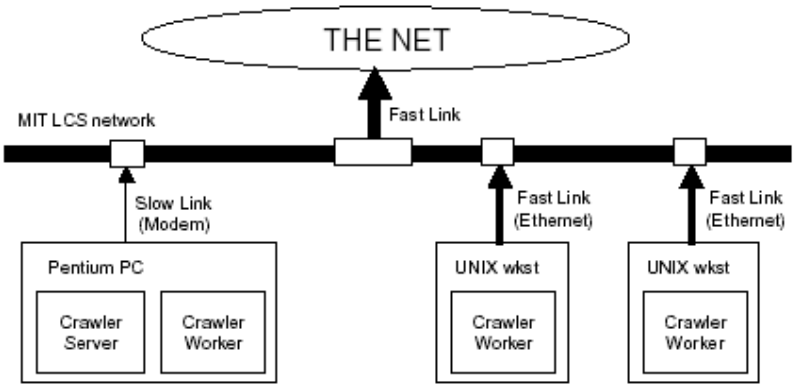

Figura 4 - Experimento de web-crawler distribuído [9].

3.3 Formas de utilização

Até então a taxonomia avaliada, diz respeito aos aspectos técnicos das arquiteturas P2P e Grids. Uma outra abordagem é analisar a forma de como esses são utilizados, tais como abrangência, facilidade de uso e aspectos comerciais. Sarmenta, em sua tese, propõe algumas possíveis formas [9].

3.3.1 Computação voluntária na essência

Sistemas como o SETI@Home [1] podem ser considerados voluntários na essência. Nesta forma de utilização, as máquinas que integram o sistema são fornecidas de forma voluntária, por pessoas distribuídas geograficamente em qualquer parte do planeta, para realizar um determinado trabalho que a comunidade julga importante.

Os voluntários não esperam um retorno pessoal significativo por seus esforços, podendo entrar e sair do Grid quando quiserem. Os administradores do sistema não conhecem, a priori, os participantes do sistema.

Com essa política, sistemas voluntários que realizam trabalhos de valor para a sociedade, tais como busca por curas de doenças, conseguem atrair milhões de internautas dispostos a ceder CPU. O resultado é um poder computacional comparável, ou maior, ao de um supercomputador.

3.3.2 Computação voluntária privada

Uma forma não tão nobre é o uso privado de metacomputação. Essa alternativa é bastante atraente para corporações, laboratórios e universidades, que possuem um imenso parque de estações de trabalho, as quais ficam ociosas na maior parte do tempo. A ociosidade ocorre não somente quando as máquinas não estão sendo usadas, mas também quando os usuários estão utilizando aplicações que não consomem CPU, como processadores de texto, o que ocorre na maior parte do tempo.

Utilizando Grids ou arquiteturas P2P privadas, uma organização pode conseguir um alto poder computacional com pouco custo. Como o ambiente é controlado, a administração e a distribuição dos aplicativos pode ser feita de forma simples, com o próprio apoio dos integrantes da organização.

3.3.3 Consórcio de computação voluntária

Da mesma forma que um ambiente privado pode ser construído, pode-se juntar um consórcio de organizações para construir um único sistema computacionalmente mais poderoso, para ser utilizado por todos os membros do consórcio.

Uma possibilidade interessante de uma relação ganha-ganha é o uso por laboratórios localizados em fusos opostos. Um laboratório americano poderia ter utilização plena durante o dia e não utilizar o sistema à noite, que estaria sendo utilizado por um laboratório japonês.

3.3.4 Computação voluntária comercial

Permitindo que pessoas negociem recursos computacionais, um ambiente P2P ou um Grid podem efetivamente tornarem-se um bem de consumo, onde pessoas podem pagar para ter capacidade computacional, e outras podem fornecer e serem recompensadas por isso.

Um possível modelo de negócio é o fornecimento de CPU em troca de um serviço na internet. Um exemplo seria um site rodar um Applet Java que integraria a máquina em um Grid, enquanto o dono da máquina utiliza o serviço oferecido pelo site.

De fato, nos últimos anos surgiram algumas companhias que criaram Grids comerciais, com finalidade de obter lucros intermediando grupos que querem pagar por computação e outros que querem receber para ceder sua computação ociosa. Entropia

[13], Parabon [14], e United Devices [2], são exemplos de companhias que estão investindo em computação voluntária comercial.

3.4 – Estratégias para detecção de falhas e sabotagens

Uma das grandes vantagens da computação colaborativa em relação a outras formas de computação paralela é a facilidade de acesso de um grande público. Isto é possível graça ao uso da Internet como rede para as aplicações P2P. Junto com esta vantagem, temos um novo problema. Se qualquer um pode se conectar ao ambiente colaborativo, como prevenir que colaboradores maliciosos sabotem a computação através da submissão de resultados falsos? Algumas aplicações como, por exemplo, as que utilizam simulações de Monte Carlo, são naturalmente tolerantes, visto que não requerem cem por cento de acuidade, porém, em alguns casos um tratamento específico é necessário para garantir cem por cento da acuidade. Algumas técnicas possíveis [9] [11] são:

- **Computações verificáveis** – São aquelas em que é possível verificar se um determinado resultado é válido em muito menos tempo do que o necessário para gerá-lo. Esse tipo de computação permite a detecção de erros, permitindo novas submissões dos trabalhos errados. Embora se gaste algum tempo extra de computação, é possível garantir a corretude;
- **Autenticação e criptografia** – Uma outra forma é evitar que o sabotador entre no sistema, utilizando técnicas de autenticação e criptografia. Essa é eficaz contra sabotadores externos, mas não previnem os internos ao sistema;
- **Redundância e randomização** – Uma maneira de tratar os sabotadores, presumindo que existem muito mais bons trabalhadores do que sabotadores, é utilizar redundância. Neste caso um trabalho é sempre submetido a mais de um trabalhador, e o resultado comparado. Se houver qualquer diferença de resultados, o trabalho é submetido a outro trabalhador. Dessa maneira descobrem-se resultados sabotados e os sabotadores, que podem ser incluídos em uma lista negra. Desta forma, a chance de se ter um resultado ruim aceito é reduzida. Porém, essa solução desperdiça bastante computação. O mecanismo de escalonar os trabalhos deve ter um componente aleatório a fim de evitar ataques de sabotagem inteligentes que enganem o mecanismo. Uma técnica que utiliza redundância é a de *spot-checking*.

Nesta, ao invés de se replicar cada trabalho submetido, os trabalhadores são testados, com alguns trabalhos que possuem o resultado conhecido (*spotter work*). Caso alguma divergência seja detectada, todos os trabalhos efetuados por aquele trabalhador são invalidados.

Capítulo 4 – Infra-estrutura Colaborativa (IeC)

Este trabalho introduz a IeC — Infraestrutura Colaborativa — uma arquitetura colaborativa com uma organização descentralizada tipo P2P. Com a adoção de XML [29] para o formato das mensagens trocadas entre os *peers*, tal arquitetura possui o objetivo de facilitar a criação de aplicações colaborativas, possibilitando que os desenvolvedores destas aplicações concentrem-se nas particularidades de seus aplicativos, deixando com a IeC os serviços de troca de mensagens entre os *peers*. A escolha de XML como padrão para a formatação das mensagens deve-se ao fato deste ser um padrão de mercado, que permite interoperabilidade entre diferentes plataformas.

4.1 Arquitetura

A arquitetura proposta provê um ambiente colaborativo transparente para as aplicações que o utilizam, ou seja, uma aplicação construída para funcionar nesta arquitetura não precisa saber dos detalhes de como, ou por quem, uma requisição de trabalho computacional gerada será atendida e nem como a resposta será entregue, cabendo tal à arquitetura **IeC**. Manter a carga balanceada entre os nós presentes, garantir a capacidade da rede se recuperar a falhas de nós, se adaptando a novas topologias, também são atribuições da **IeC**.

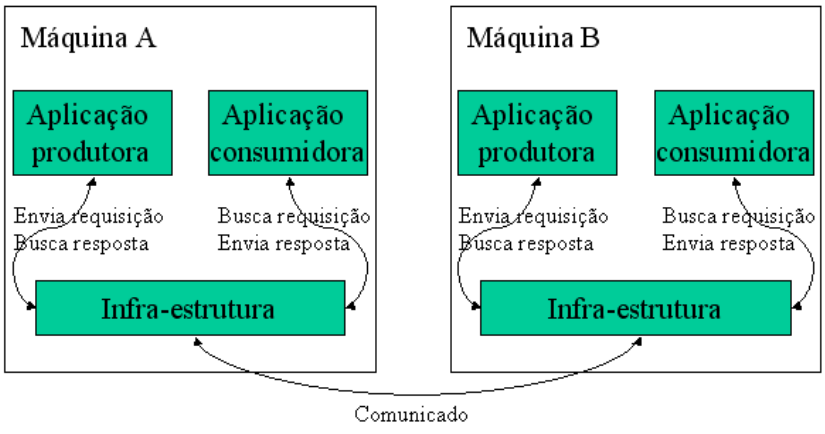

Figura 5. Relacionamento das aplicações com a infraestrutura IeC e entre os nós computacionais.

Segundo a classificação da taxonomia de sistemas distribuídos, definida na Seção 2.2.2, podemos classificar a IeC como sendo uma plataforma P2P pura. Analogamente, apesar da IeC não ser um Grid, porque não implementa todos os serviços necessários para lidar com um universo variado de recursos, podemos encontrar algumas característica presentes nos Grids. A Tabela 5 apresenta estas característica a as classifica segundo a taxonomia dos Grids, descrita na Seção 2.1.3.

Tabela 5 - Características da IeC de acordo com a taxonomia dos Grids.

Característica	***IeC***
Gerência dos recursos computacionais	Computacional de alta vazão
Organização das máquinas	Plana
Escalonamento	Descentralizado sem histórico

A Figura 5 mostra os fluxos de mensagens que circulam no sistema colaborativo. Uma aplicação produtora envia pacotes de requisição para o módulo da infra-estrutura e recebe pacotes de resposta da infra-estrutura, a qual envia pacotes de comunicado para outros nós da rede. Aplicações consumidoras recebem pacotes de requisições da arquitetura **IeC** e enviam os pacotes de resposta de volta.

Na construção da arquitetura **IeC**, os seguintes objetivos foram perseguidos:

- Facilidade de integração com aplicações colaborativas;
- Facilidade de configuração;
- Possibilidade de interoperabilidade entre diversas plataformas;
- Possibilidade de ser estendida.

4.1.1 API

Para a arquitetura **IeC** operar, cada nó computacional que participa do sistema colaborativo deve instalar um módulo do sistema, que fornece uma API padronizada para as aplicações produtoras e consumidoras de trabalhos. Este módulo é responsável por:

- escalonar as requisições recebidas de aplicações produtoras entre os diversos nós que possuem aplicações registradas para consumi-las;
- receber as respostas das requisições feitas por aplicações registradas no nó, entregando-as para estas;

- fornecer informação sobre a distribuição de carga para as aplicações, permitindo que elas controlem o tamanho do grão (Seção 4.1.7);
- reenviar requisições não atendidas no tempo máximo configurado (*time-out*).

A API foi escrita no modelo COM da Microsoft [30], que é um modelo de componentes padrão para o Windows, possibilitando que aplicações desenvolvidas, em praticamente todas as linguagens disponíveis para este ambiente, possam entrar em contato com o sistema. O fato de a arquitetura utilizar mensagens em XML e trocá-las via comunicação TCP/IP via sockets, possibilita que a própria infra-estrutura seja re-escrita para outras plataformas, permitindo inclusive que estratégias diferentes de escalonamento, descoberta e disseminação de recursos e controle de carga, sejam utilizadas na mesma rede. A Figura 6 mostra um diagrama da relação das aplicações com a IeC, e de um Nó com outro. Podemos ver que uma aplicação acessa a IeC por uma interface COM, a IeC interage com o sistema operacional utilizando a interface de *sockets* e os nós computacionais se comunicam por TCP/IP.

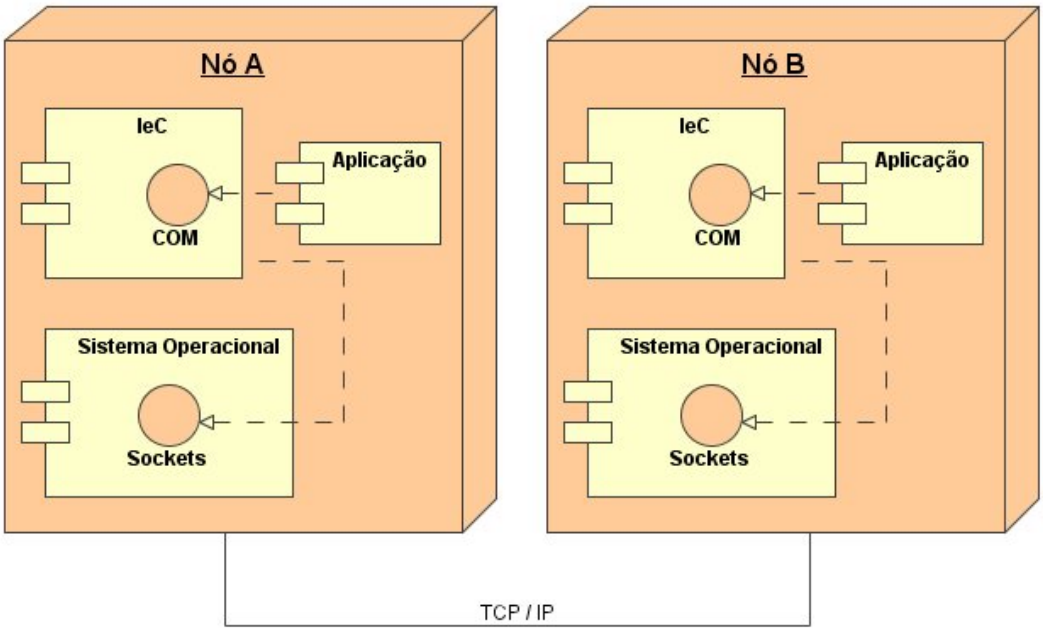

Figura 6 - Interfaces aplicação x IeC x SO e Nó x Nó.

Os seguintes métodos são disponibilizados pela API do sistema (a Figura 7 apresenta a declaração formal da API e a Figura 8 ilustra uma dinâmica desses métodos):

- Registro (DoRegister) – Através desse método, uma aplicação se habilita a utilizar os serviços oferecidos. Caso forneça algum serviço, o tipo de requisição que ela atende é indicado. A aplicação recebe um identificador de aplicação (*AppID*) que deverá ser utilizado para identificá-la no relacionamento com a **IeC**;
- Requisitar serviço (SendRequest) – Este método permite a uma aplicação produtora (ou cliente) requisitar um trabalho. A aplicação recebe um identificador para o trabalho solicitado (*ReqID*);
- Buscar resposta (GetResponse) – Esse método permite que uma aplicação verifique se alguma requisição feita anteriormente foi respondida, sendo entregue caso exista. O *ReqID* identifica a requisição que foi respondida;
- Buscar requisição de serviço (GetRequest) – Através deste método, uma aplicação consumidora (ou servidora) busca uma requisição para resolver. O retorno será vazio se não existir uma requisição pendente;
- Enviar resposta (SendResponse) – Uma aplicação servidora, utiliza este método para enviar a resposta para um trabalho feito, atendendo a uma requisição recebida;
- Buscar distribuição (GetDistribution) – Verifica como está a distribuição das requisições entre o conjunto de máquinas que fornece um serviço.

```
function  GetRequest(AppID: integer; ServiceID: string): string;
function  GetResponse(AppID : integer): string;
function  DoRegister(ServiceID: string): integer;
function  SendRequest(AppID: integer; AppXml: string): integer;
procedure SendResponse(AppXml, Machine : string; ReqID: Integer);
function  GetDistribution(ServiceID: string): integer;
```

Figura 7 - API do IeC.

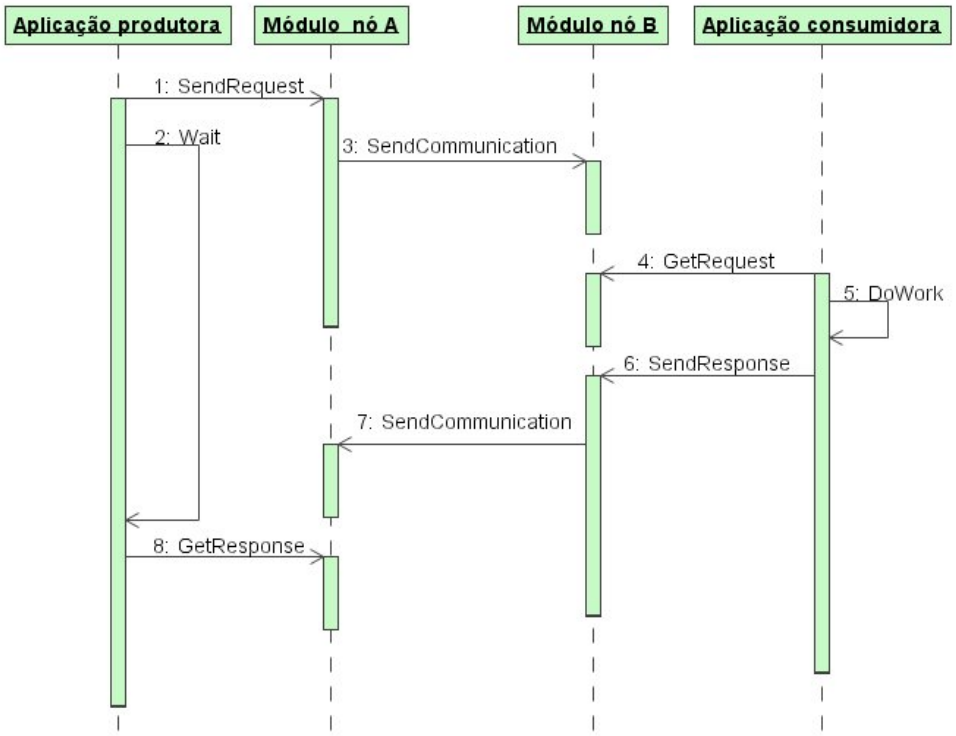

Figura 8 - Ciclo de vida de uma requisição.

4.1.2 Componentes da arquitetura IeC

O módulo da **IeC** foi construído com uma arquitetura de componentes, sendo cada um responsável por atender uma funcionalidade específica. Assim, podemos dividir o módulo nos seguintes componentes principais:

- Gerente de resposta – É o guardião das respostas recebidas para as solicitações feitas. As respostas vão sendo removidas à medida que as aplicações que fizeram as requisições respondidas as buscam. É implementado como um objeto da classe TGerente (Figura 9);
- Gerente de requisição – Mantém uma fila para cada tipo de requisição recebida, sendo a fonte de requisições para as aplicações consumidoras. Implementado como um objeto da classe TGerenteServico (Figura 9);
- Gerente de envio – Mantém o controle das requisições enviadas pelas aplicações produtoras registradas. Para cada requisição pendente, são mantidos os IPs das máquinas que a receberam, e o horário do último envio. Este módulo informa para o escalonador a necessidade de envio redundante da requisição, conforme a política configurada. É implementado como um objeto da classe TGerenteEnvio (Figura 9);

- Escalonador – Componente chave do sistema. Este módulo é responsável por distribuir as requisições recebidas entre os nós computacionais. Cabe a ele decidir qual nó existente é o mais adequando para receber uma requisição, a fim de manter o sistema com carga equilibrada. É implementado como um objeto da classe TDispatcher (Figura 9);
- Gerente de conectividade – Sua função é manter o nó conectado ao sistema da melhor forma para este. Esse componente é responsável em descobrir máquinas no sistema e iniciar a comunicação com estas, fornecendo para o escalonador a lista das máquinas conhecidas. Na implementação, esse papel ficou no mesmo objeto que o Escalonador, ou seja, em um objeto da classe TDispatcher (Figura 9);
- Receptor de comunicados – Recebe os comunicados dos diversos nós, sendo responsável por distribuir as informações aos outros componentes do sistema. Respostas são repassadas ao gerente de resposta, dando baixa no gerente de envio (fim da pendência da requisição); requisições são repassadas para o gerente de requisição, informações de carga são repassadas ao escalonador. Na implementação, este papel está divido entre as classes TIECTalk e TDispatcher (Figura 9).

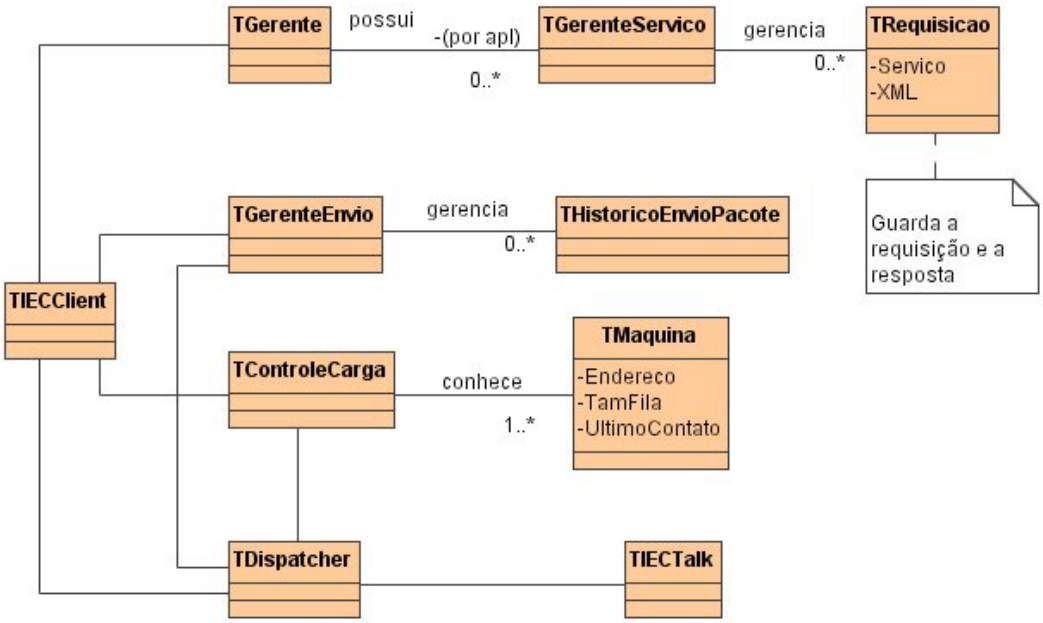

Figura 9 - Principais classes do IeC.

4.1.3 Pacotes de informação

Os tipos de requisição são nomeados de forma única, permitindo que diversas aplicações possam conviver juntas na rede sem que haja conflito. Para cada requisição feita, é gerado um pacote XML contendo o descritor do solicitante e o pacote da aplicação. O descritor contém o IP do nó e o *AppID* da aplicação que gerou a requisição. O pacote da aplicação é um nó do pacote XML que recebe o nome do tipo da requisição. Este nó XML pode conter diversos sub-nós, compostos ou não, de acordo com a necessidade da aplicação representar os dados da requisição. Da mesma forma, uma resposta a uma requisição é representada por um pacote XML contendo o destino da resposta, ou seja, IP do nó e *AppID*, e um nó XML com o nome do tipo da requisição, organizado da maneira que a aplicação julgar necessário.

A troca de mensagens entre os diversos nós presentes na rede é feita por pacotes maiores, chamados de *pacotes de comunicado*, os quais agregam diversos pacotes de respostas e requisições, além de dados sobre a situação no nó, que serão utilizados para a gerência da rede. A Figura 10 mostra a estrutura de um pacote de comunicado. A estrutura dos pacotes são descritas e validadas através de *XML Schemas* [31].

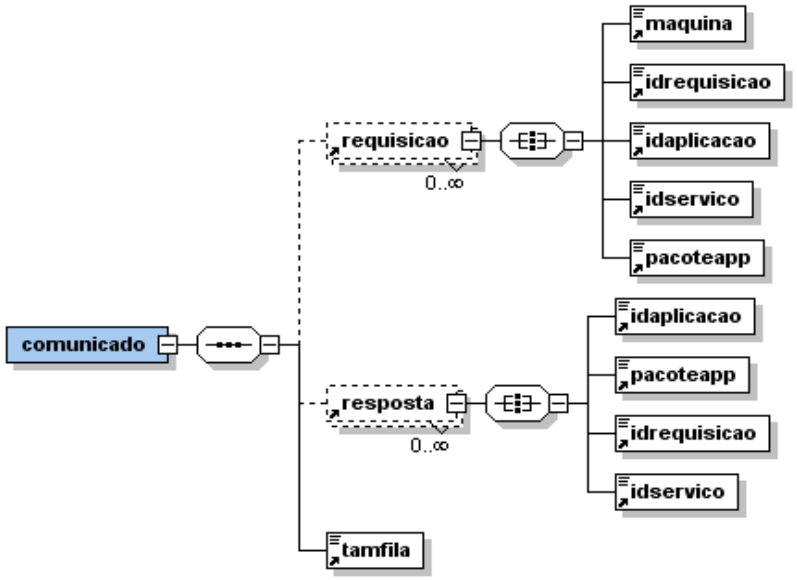

Figura 10. Estrutura de um pacote de comunicado (XML).

O XML foi escolhido porque é um padrão aberto definido por um consórcio, baseado em texto. Por esses motivos, existem muitas ferramentas disponíveis para XML

em diversas linguagens, tal como *parsers*, permitindo a não re-invenção da roda. Além disso, por ser um padrão baseado em texto, é multiplataforma, possibilitando que as interfaces da IeC baseadas em XML possam ser reescritas para outras plataforma, permitindo que a IeC não fique restrita ao ambiente Windows.

Sendo a definição do pacote de comunicado, por ser XML, hierárquico, é possível o acréscimo de novos nós no pacote, além dos atuais (**requisicao**, **resposta** e **tamfila),** permitindo novas funcionalidades em futuras versões da IeC.

4.1.4 Escalonamento

O escalonamento adotado ocorre de forma totalmente descentralizada, ou seja, cada nó toma sua própria decisão sem apoio de um servidor central que possua uma visão completa da rede. Entretanto, ainda é desejado que uma requisição deva ser enviada para um nó que seja a melhor alternativa para o desempenho global do sistema.

Neste ambiente, a estratégia de escalonamento busca uma computação de alta vazão, o que significa a realização da maior quantidade de trabalhos possíveis em um determinado intervalo de tempo, visando maximizar o desempenho do sistema como um todo e não de uma aplicação específica.

A abordagem de escalonamento desejada deve atender aos requisitos acima, porém, sem introduzir *overheads* e retardos significativos no sistema.

Uma primeira estratégia utilizada em nossos testes foi uma abordagem *round-robin*, na qual cada nó envia, para os nós conhecidos que possuem consumidores para o tipo de requisição, suas requisições, escolhendo o nó destino a partir de uma lista circular. Tal abordagem poderia funcionar adequadamente em um ambiente controlado, em que as máquinas, com uma mesma capacidade de processamento, estariam sendo utilizadas exclusivamente para o ambiente colaborativo. Entretanto, em ambientes reais, esta abordagem pode sobrecarregar de requisições máquinas lentas ou não ociosas, deixando máquinas mais capazes desocupadas. Até mesmo em um ambiente controlado, conforme mencionado, estes problemas podem ocorrer, visto que se as requisições possuem tamanho variado, é possível que algumas máquinas fiquem sobrecarregadas com requisições mais demoradas.

A solução adotada para o componente escalonador foi a de cada nó manter a informação do tamanho da fila de requisições, mantida pelo componente “Gerente de Requisição” de cada máquina conectada ao nó em questão. Assim, a máquina escolhida

para enviar uma requisição é sempre a com menor fila que possua uma aplicação consumidora registrada para a requisição. Para manter a informação das filas das máquinas, incluímos o dado tamanho de fila no *pacote de comunicado* trocado em todas as comunicações (nó **tamfila** da estrutura mostrada na Figura 10). O *overhead* de comunicação é aceitável dado o pequeno aumento induzido no tamanho do pacote. Como parte do protocolo de troca de informação de filas, um nó sempre após receber um comunicado, informa ao nó que enviou o comunicado, o tamanho atual de sua fila.

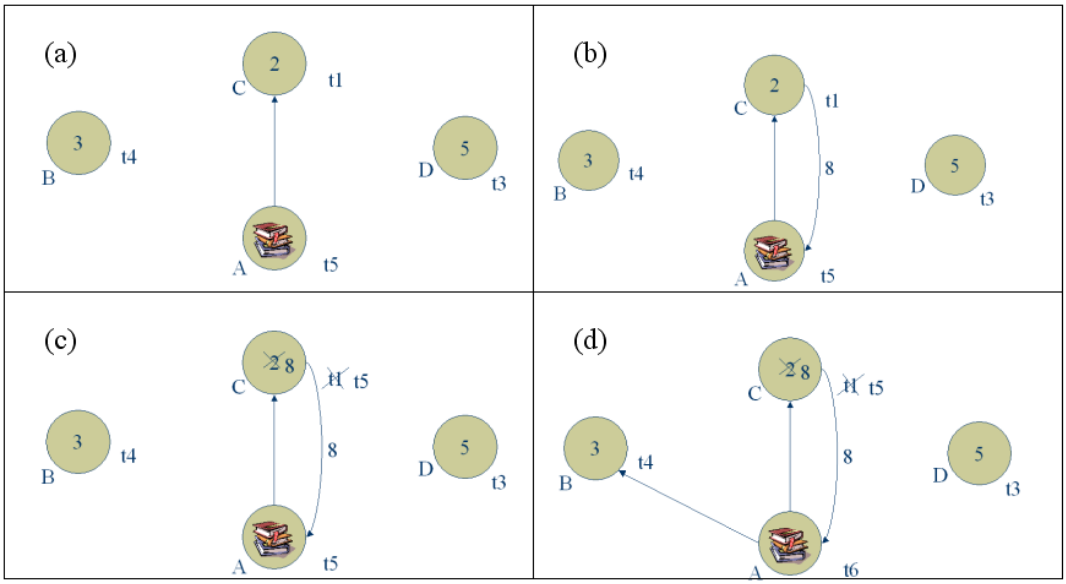

Figura 11 - Dinâmica do escalonamento.

A Figura 11 ilustra a dinâmica deste escalonamento. Em *(a)*, o nó A encontra-se no tempo *t5* e conhece o tamanho das filas dos nós B, C e D, porém tal informação foi obtida em tempos diferentes, respectivamente, *t4*, *t1* e *t3*. Como o escalonador opta pelo nó de menor fila, uma requisição é enviada para o nó C, que é o com menor fila. O nó B, após receber o pacote, envia o tamanho de sua fila, como visto em *(b)*. De posse desta informação, O nó A atualiza sua tabela de filas, indicando que C possui 8 em sua fila *(c)*. É importante salientar que a decisão "errada" tomada em *(a)* não é desfeita, porém o "erro" não é mais repetido, o que é mostrado em *(d)*, ou seja, no tempo seguinte, *t6*, a próxima requisição não é mais enviada para C, e sim, para B.

Apesar do grande dinamismo do sistema, em que diversos nós fazem requisições simultâneas, essa abordagem adapta-se às mudanças de forma rápida a diversas situações, tais como:

- casos em que a visão de um nó sobre outro está defasada em relação ao estado atual. Exemplo: dado uma situação em que o nó A possui a informação de que o nó mais desocupado é o nó D; porém, o nó B fez diversas solicitações, tornando o nó D carregado com requisições. Na primeira requisição do nó A para o nó D, este responde com o tamanho de sua fila, permitindo que o nó A reavalie se o nó D continua, ou não, sendo a melhor opção, permitindo uma rápida adaptação a esta nova situação;
- existência de máquinas mais lentas. Neste caso, todas as máquinas do sistema começam a receber requisições na mesma velocidade que outras mais rápidas. Como as mais rápidas consomem suas requisições de forma mais acelerada, as filas das mais lentas ficam maiores, e o sistema direciona os novos pacotes somente para as máquinas mais rápidas, até que as filas voltem ao equilíbrio.

4.1.5 Redundância

A política de envio de requisições pode ser configurada para gerar redundância de pacotes. Isto é feito por dois principais motivos:

(i) tornar o ambiente tolerante a falhas;

(ii) aumentar a eficiência do sistema quando existem máquinas ociosas.

Como a arquitetura **IeC** é descentralizada, cada nó pode escolher utilizar a configuração mais adequada às necessidades das suas aplicações registradas.

Para atender ao primeiro motivo apresentado, o sistema permite a configuração de um time-out para as requisições enviadas, logo, caso o nó que recebeu uma requisição não responda no tempo programado pelo nó que a enviou, uma nova requisição é gerada para um outro nó. O componente "Gerente de Envio" é o responsável por avisar o escalonador que o pacote deve ser re-enviado, indicando as máquinas que não devem recebê-lo.

Em arquitetura centralizadas como o Bayanihan [9] [10], a política de *eager scheduling* [10] é uma forma simples de aproveitar as máquinas ociosas para trabalharem em requisições redundantes. No Bayanihan os objetos de trabalho são organizados em uma lista circular com um ponteiro indicando o próximo objeto ainda não completo. Desta forma, quando todos os trabalhos forem entregues, o ponteiro é posicionado em um trabalho já entregue, sendo novamente entregue a outra máquina

que solicitar um trabalho. Esta estratégia permite que nenhuma máquina fique ociosa, permitindo que as mais rápidas trabalhem mais, evitando gargalos nas mais lentas.

Na arquitetura proposta, *eager scheduling* não é uma estratégia aplicável diretamente, visto que as aplicações consumidoras não possuem uma fonte centralizada para buscarem requisições. Portanto, foi adotada uma política de "racionamento" para requisições, que visa melhorar o desempenho em aplicações que despejam um lote de requisições iniciais e depois ficam aguardando as diversas respostas. Aplicações no estilo Master-Worker podem se beneficiar da estratégia.

A política de "racionamento" consiste em definir o número máximo de requisições de um determinado tipo que uma máquina pode receber. Desta forma, se uma aplicação despejar um número de requisições maior que o número máximo definido para o tipo de requisição, multiplicado pelo número de máquinas, as requisições serão represadas no nó origem até que alguma máquina entregue alguma resposta. O objetivo é evitar uma má distribuição inicial devido, por exemplo, a uma máquina ser mais lenta que outras. Os pacotes vão sendo liberados conforme as máquinas vão atendendo aos pacotes enviados anteriormente. A redundância é gerada quando o número máximo definido para o tipo de requisição, multiplicado pelo número de máquinas, é maior do que o número de pacotes. Neste caso, são enviadas requisições redundantes em um esquema de lista circular, tal como na abordagem de *eager scheduling* [10], até que todas as máquinas recebam a quota estipulada.

O uso desta política deve ser estudado cuidadosamente. O número máximo estipulado deve ser de um tamanho suficiente para que os nós consumidores não fiquem parados esperando por pacotes, porém, quanto maior o tamanho, mais redundância e *overhead* serão gerados ao sistema. Outro problema é a utilização em aplicações que usem abordagens *Master-Worker* com hierarquia (Figura 3), em que os trabalhadores re-dividem o trabalho, tornando-se mestres. Nestas aplicações, é provável a ocorrência de *deadlocks*, pois, como existe dependência entre as requisições, se requisições que precisam ser resolvidas para liberar seus pais, na árvore de dependência, ficarem represadas no nó de origem, o sistema todo fica parado até que novos nós entrem no sistema, o que pode nunca acontecer.

A Figura 12 é um exemplo da distribuição de requisições quando a IeC está configurada para utilizar política de racionamento. No exemplo, a configuração é que cada nó pode reter apenas duas requisições. Em *(a)*, temos uma situação inicial, em que a aplicação Mestre dividiu um problema em dez partes, enviando dez requisições para o

módulo da IeC de sua máquina. As requisições de 1 até 6 foram enviadas para os três nós trabalhadores existentes, fazendo com que o nó Mestre mantivesse as requisições de 7 à 10, visto que não existiam trabalhadores disponíveis para recebê-las. É importante observar que a aplicação Mestre não tem conhecimento da retenção, ou seja, esta ocorre na IeC. A figura *(b)* representa um momento posterior em que cada um dos trabalhadores resolveu uma requisição, permitindo que estes recebessem mais uma requisição. Neste ponto o nó mestre ficou retendo apenas a requisição 10. Em *(c)*, novamente, cada um dos trabalhadores resolveu mais uma requisição, logo, a requisição 10 foi enviada para o primeiro trabalhador e, como não existiam outras requisições pendentes, também, de forma redundante, para os demais trabalhadores.

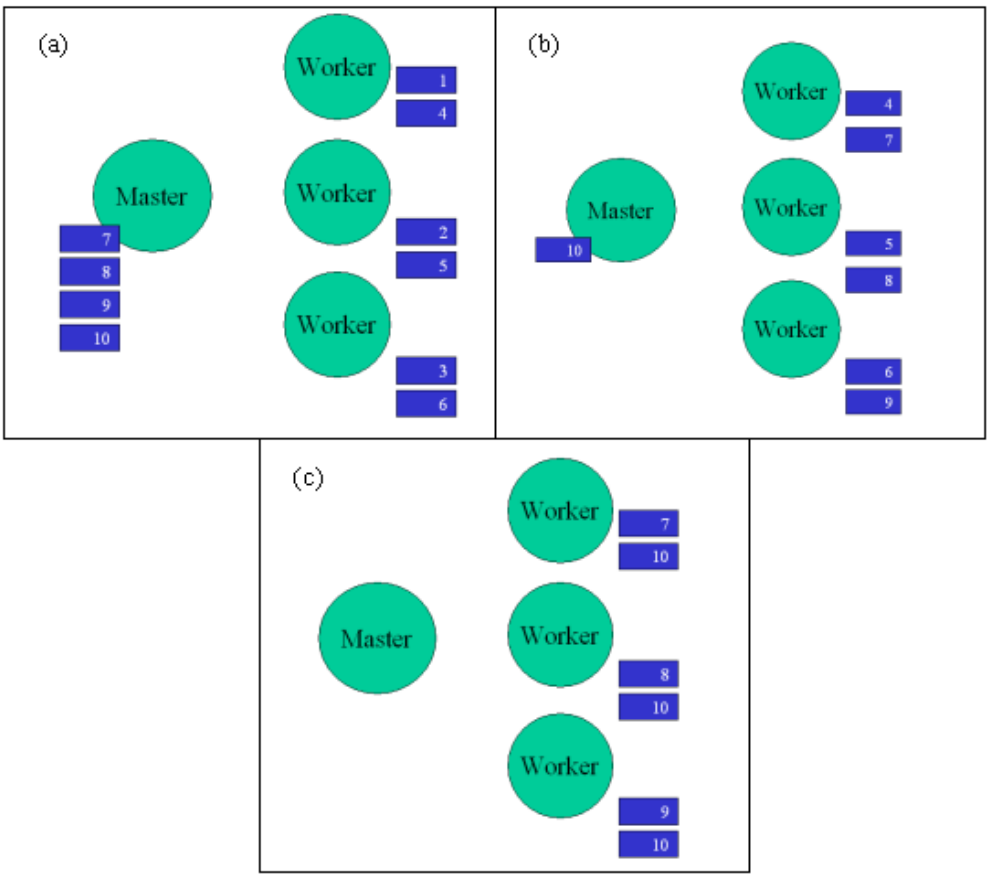

Figura 12 - Exemplo do uso da política de "racionamento".

4.1.6 Conectividade

O objetivo do sistema é que cada nó mantenha um conjunto de conexões suficientes para:

- garantir a resiliência do sistema, mantendo todos os nós do sistema conectados entre si diretamente, ou indiretamente, mesmo após a saída ou quedas de nós;
- garantir acesso aos nós que atendem as requisições solicitadas.

Na versão atual da arquitetura, o gerenciamento de conexões está estático, feito através de um arquivo XML, carregado por cada nó presente na rede. Tal arquivo contém a relação dos nós presentes e os tipos de requisições atendidas por cada nó. A meta é que a arquitetura consiga resolver suas conexões de forma dinâmica, o que pode ser conseguido, por exemplo, através da estratégia utilizada pelo *Gnutella* [19]. Para um nó entrar na rede é necessário um ponto de partida, ou seja, o endereço de pelo menos um nó que faça parte da rede. Isto é feito mantendo-se uma lista dos nós que estão quase sempre conectados. Depois de conectado, o nó utiliza o mecanismo de PING e PONG para descobrir outros nós. Neste mecanismo, o nó que está se juntando a rede envia uma mensagem broadcast PING para anunciar sua presença. Os nós que recebem uma mensagem PING respondem com uma mensagem PONG, a qual contém informações sobre o nó.

4.1.7 Controle da granularidade

Um dos serviços fornecidos pela arquitetura às aplicações colaborativas é o fornecimento da situação da distribuição de requisições entre as máquinas que consomem um dado tipo de requisição. O objetivo desta comunicação, é permitir que as aplicações não fiquem totalmente cegas em relação ao ambiente em que elas estão trabalhando, permitindo-as tomar decisões que possam melhorar o desempenho delas. A informação passada para as aplicações é o nível da dispersão. Para isso, utilizamos o *coeficiente de variação*, cálculo estatístico para medir dispersão relativa, definido como $\frac{\delta}{x}$, onde δ é o desvio padrão do tamanho das filas e x a média. A arquitetura converte o resultado, representado pela variável *y*, em três valores discretos em função do resultado, conforme mostra a Tabela 6.

Tabela 6. Níveis de dispersão informados.

Dispersão calculada (*y*)	Nível informado
0% <= *y* < 30%	0 – Baixa dispersão.
30% <= *y* < 50%	1 – Média dispersão.
y >= 50%	2 – Alta dispersão.

Se o ambiente está com baixa dispersão, isto indica distribuição de carga está boa, logo, uma aplicação pode utilizar essa informação para aumentar o grão de suas requisições, porque não existe nó ocioso, e um grão maior significa menor *overhead* de comunicação. De forma contrária, uma dispersão maior indica que existem máquinas trabalhando mais do que outras. A aplicação pode optar em diminuir o grão nesta situação para aproveitar o processamento ocioso.

A Figura 13 exemplifica uma aplicação fazendo uso deste controle. Esta divide os trabalhos em grãos grandes no início e depois começa a trabalhar no grão junto com os outros nós computacionais. Se a dispersão for média, ou alta, a aplicação opta em não realizar o trabalho inteiro, o dividindo em pedaços menores, realizando o trabalho em um pedaço e enviando os demais para o sistema. Se a dispersão for baixa, esta opta em realizar o trabalho inteiro. Deste modo, a aplicação evita que nós fiquem ociosos.

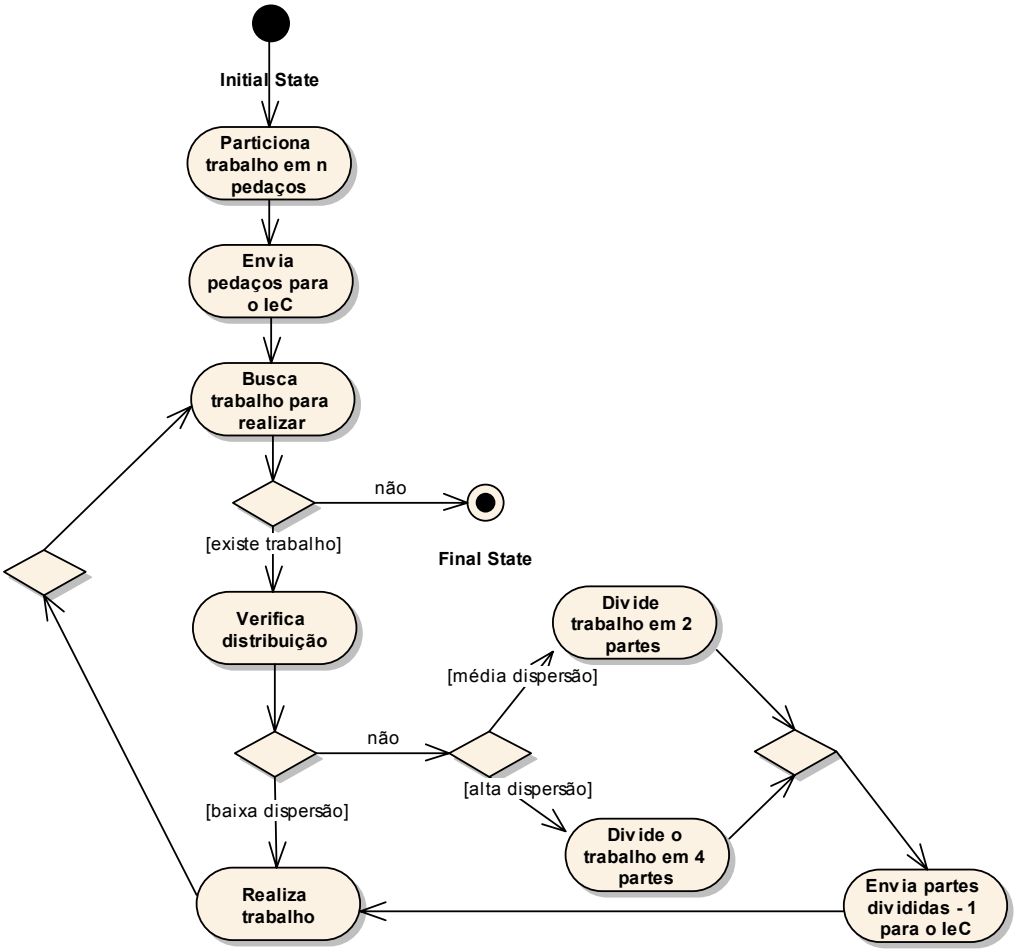

Figura 13 - Diagrama de atividades mostrando uma aplicação fazendo uso da situação da distribuição de carga no IeC.

4.2 Descrição das funcionalidades em caso de uso

Esta seção apresenta as funcionalidades da arquitetura em forma de caso de uso. Os atores identificados são:

- Aplicação Cliente – aplicação que interage com o sistema para requisitar um serviço;
- Aplicação Servidora – aplicação interage com o sistema para resolver requisições solicitadas por aplicações clientes;
- Relógio – representa eventos internos do sistemas que ocorrem em determinados intervalos de tempo;
- Outro IeC – representa um outro nó rodando o IeC que está interagindo com o sistema.

4.2.1 Modelo de caso de uso

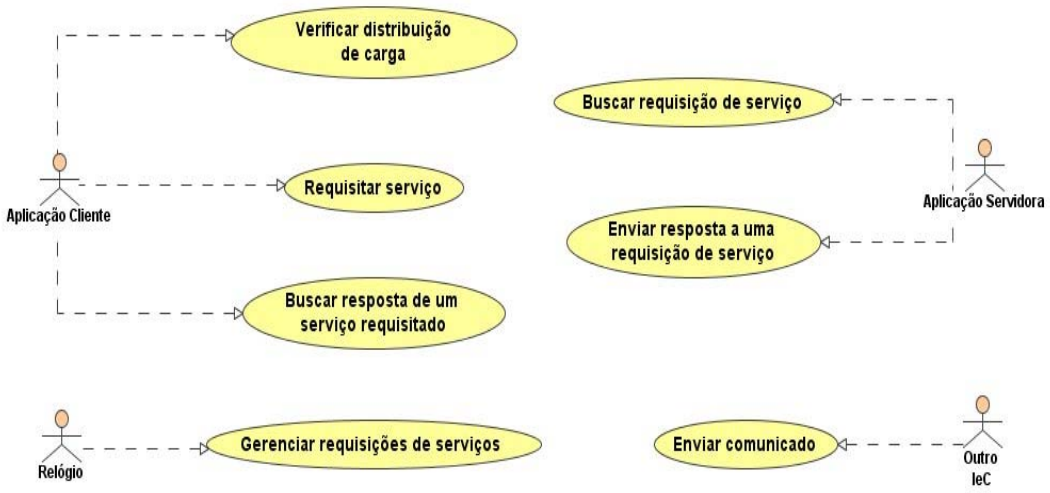

Figura 14 - Modelo de caso de uso.

4.2.2 Diagramas de atividade

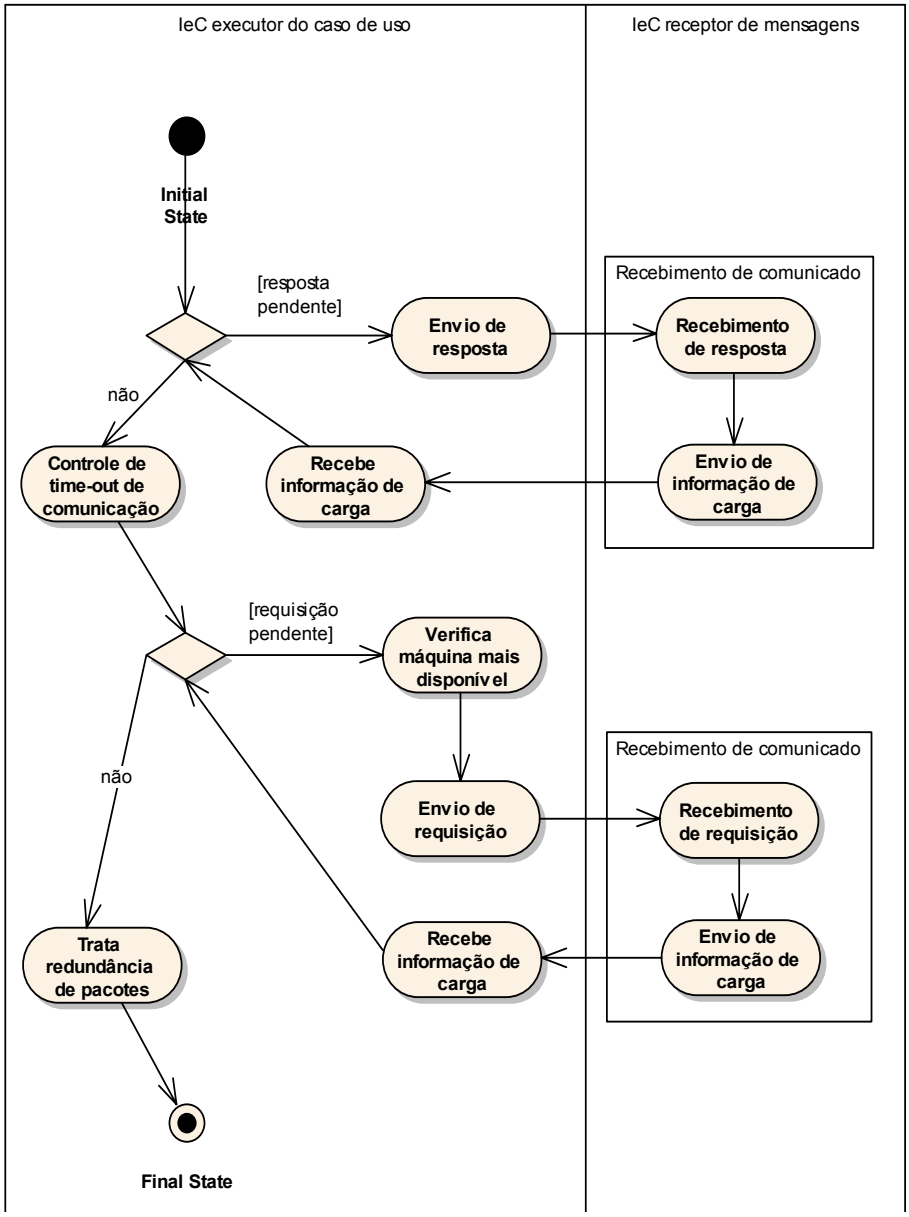

Figura 15 - Diagrama de atividades do processo de envio de respostas, escalonamento e envio de requisições.

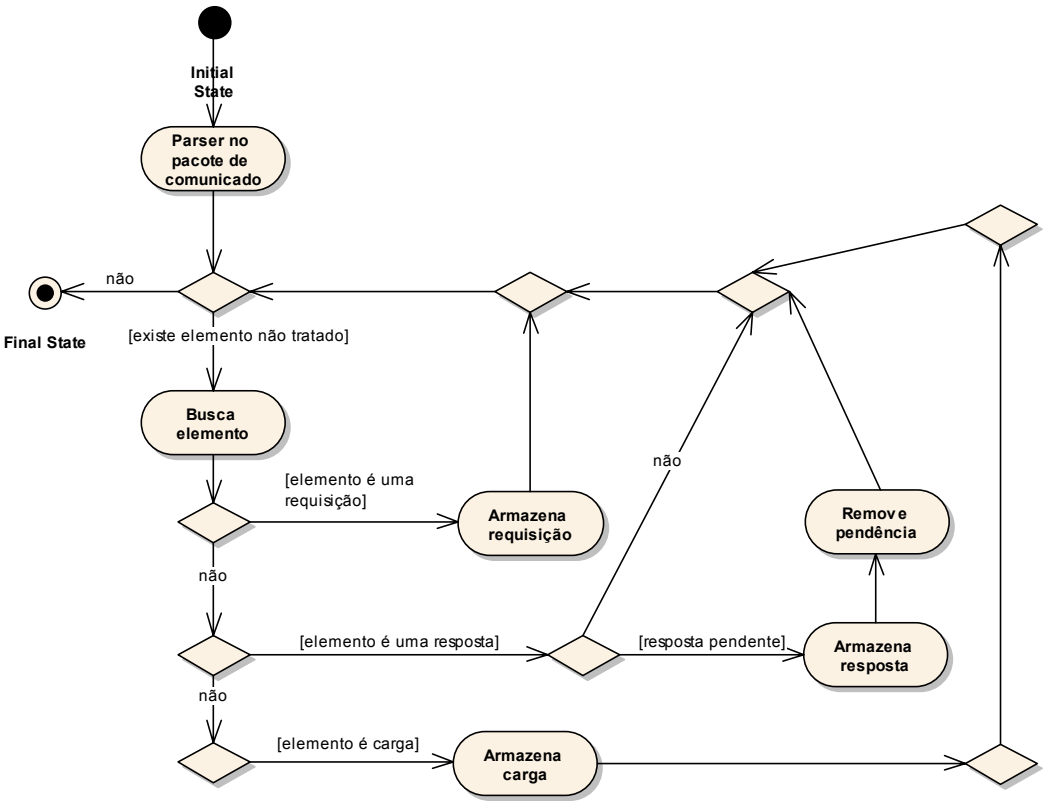

Figura 16 - Diagrama de atividades do processo de recebimento de um comunicado de um outro nó IeC da rede.

4.2.3 Caso de Uso - Requisitar serviço

Representa a interação com o sistema de uma aplicação cliente que está requisitando um serviço para o sistema.

Fluxo básico

O caso de uso inicia quando uma aplicação acessa o IeC para enviar uma requisição.

1. **Recebimento da requisição**

   O sistema recebe da aplicação o pacote de requisição, o serviço desejado e o ID da aplicação (ID fornecido no registro).

2. **Validação da requisição**

   O sistema verifica se o ID da aplicação é válido.

3. **Armazenamento da requisição**

   O sistema gera um ID para a requisição.

O sistema monta um pacote de requisição com a informação do serviço, ID da aplicação, ID da requisição e pacote da aplicação.

O sistema armazena a requisição na lista de requisições pendentes para o serviço solicitado.

O sistema retorna para a aplicação o ID da requisição gerado.

O caso de uso é encerrado.

Fluxos Alternativos

1. **Falha na validação**

   O fluxo alternativo inicia se a validação feita no passo 2 do fluxo básico falhar.

   O sistema envia exceção à aplicação cliente informando o problema encontrado.

   O caso de uso é encerrado.

4.2.4 Caso de Uso - Enviar resposta a uma requisição de serviço

Representa a interação com o sistema de uma aplicação servidora que está enviando uma resposta para uma requisição recebida anteriormente.

Fluxo básico

O caso de uso é iniciado quando uma aplicação servidora acessa o IeC para enviar um pacote de resposta para uma requisição recebida anteriormente.

1. **Recebimento da resposta**

   O sistema recebe da aplicação o pacote de resposta, o ID da requisição respondida e o endereço da máquina que enviou a requisição.

2. **Armazenamento da requisição**

   O sistema monta um pacote de resposta com a informação do serviço, ID da requisição e pacote da aplicação associado ao nó que requisitou.

   O sistema armazena a resposta na lista de respostas pendentes.

   O caso de uso é encerrado.

4.2.5 Caso de Uso - Gerenciar requisições de serviços

A raia “IeC executor do caso de uso” do diagrama de atividades na Figura 15 representa as atividades do fluxo básico deste caso de uso. Este caso de uso foi implementado com uma *thread* que fica constantemente varrendo as listas de requisições e respostas pendentes e executando as ações descritas.

Fluxo básico

O caso de uso é iniciado pelo sistema em intervalos de tempo de um milisegundo, desde que o sistema operacional não esteja ocupado com outros processos.

1. **Envio das respostas pendentes**

   Para cada resposta pendente, recebidas no caso de uso 4.2.4.

   O sistema prepara o pacote de comunicado entre máquinas, contendo a resposta para o serviço, e o tamanho da fila de requisições recebidas ainda não tratadas.

   O sistema envia o pacote de comunicado para o sistema que havia requisitado o serviço.

   O sistema recebe como resposta o tamanho da fila de requisições recebidas ainda não tratadas, da máquina para qual enviou o comunicado.

   O sistema registra a informação sobre a fila da máquina para qual enviou a resposta.

2. **Controle de time-out de comunicação**

   Para toda máquina conhecida pelo sistema com quem não houve comunicação, durante um determinado período de tempo configurável, o sistema atualiza a informação sobre a fila desta como zero.

3. **Envio das requisições solicitadas pendentes**

   Para cada requisição pendente, recebidas no caso de uso 0.

   O sistema prepara o pacote de comunicado entre máquinas, contendo a requisição para o serviço, e o tamanho da fila de requisições recebidas ainda não tratadas.

   O sistema descobre, entre as máquinas conhecidas, a com menor fila de requisições pendentes, que não atingiu o "número máximo de requisições por serviço", conforme explicado no passo 4.

   O sistema envia o comunicado para a máquina descoberta.

   O sistema recebe como resposta o tamanho da fila de requisições recebidas ainda não tratadas, da máquina para qual enviou o comunicado.

   O sistema registra a informação sobre a fila da máquina para qual enviou a resposta.

4. **Envio de pacotes redundantes**

   O sistema verifica, para cada requisição enviada ainda não respondida, se deve enviá-la para uma outra máquina. A redundância pode ocorrer por time-out, por definição do número de requisições de um determinado serviço que uma máquina pode receber, ou de forma híbrida, de acordo com a configuração definida.

- Time-out puro – um novo pacote é enviado sempre que o intervalo de tempo entre o tempo atual e o do último envio, for maior do que o tempo definido para time-out.
- Número máximo de requisições por serviço – um novo pacote é enviado para as máquinas conhecidas que ainda não possuem o número determinado de requisições pendentes para o serviço.
- Híbrido – funciona conforme o "número máximo de requisições por serviço", porém, um novo pacote somente é enviado se o time-out configurado ocorrer.

Em todos os casos, uma requisição nunca é enviada duas vezes para uma mesma máquina, e para cada requisição pendente, apenas um re-envio é feito na execução do caso de uso.

Para cada requisição que atendeu a condição de redundância configurada, o sistema descobre qual a máquina com a menor fila de requisições pendentes, que não recebeu a requisição e não atingiu o "número máximo de requisições por serviço".

O caso de uso é encerrado.

Fluxos Alternativos

1. **Falha no envio de resposta**

   O fluxo alternativo inicia quando, no passo 1 do fluxo básico, o sistema não consegue entregar a resposta.

   O sistema mantém a resposta para tentar enviá-la na próxima rodada do caso de uso.

   O caso de uso continua no passo 1 do fluxo básico.

2. **Falha de envio**

   O fluxo alternativo inicia quando, nos passos 3 e 4 do fluxo básico, não é possível estabelecer contato com a máquina escolhida.

   O sistema utiliza a máquina seguinte segundo a ordenação por fila de requisições pendentes.

   O caso de uso retorna ao passo que gerou o fluxo alternativo.

3. **Não existe outra máquina disponível**

   O fluxo alternativo inicia quando, no fluxo alternativo 2 não é possível determinar outra máquina para atender o envio da requisição.

   O sistema não envia a requisição.

   O caso de uso continua no passo que gerou o fluxo alternativo.

4.2.6 Caso de uso – Verificar distribuição de carga

Uma aplicação servidora consulta o IeC sobre a situação da distribuição de carga no sistema, obtendo o nível de dispersão.

Fluxo básico

O caso de uso é iniciado quando uma aplicação servidora acessa o IeC para saber sobre a distribuição de carga.

1. **Cálculo da distribuição**

   O sistema realiza o cálculo do *coeficiente de variação*, definido como $\frac{\delta}{x}$, onde δ é o desvio padrão do tamanho das filas e x a média. O sistema converte o resultado em três valores discretos, conforme mostra a Tabela 6.

   Tabela 6.

   O sistema informa a aplicação o valor encontrado.

   O caso de uso é encerrado.

4.2.7 Caso de uso – Buscar resposta de um serviço requisitado

A aplicação que requisitou um trabalho ao IeC, utiliza esse caso de uso para buscar a resposta da requisição feita.

Fluxo básico

O caso de uso é iniciado quando uma aplicação cliente deseja saber ser uma resposta para uma requisição já está disponível.

1. **Receber pedido de resposta**

   O sistema recebe o pedido de resposta contendo o ID da aplicação requisitante.

2. **Validação da requisição**

   O sistema verifica se o ID da aplicação é válido.

3. **Busca da resposta**

   O sistema verifica se existe alguma resposta associada ao ID da aplicação.

   Se existir uma resposta, o sistema responde com esta. Se existir mais de uma, o sistema responde com a primeira resposta da fila de respostas para a aplicação. Se não existir, o sistema responde com vazio.

   O caso de uso é encerrado.

Fluxos Alternativos

4. **Falha na validação**

   O fluxo alternativo inicia se a validação feita no passo 2 falhar.

   O sistema informa o problema para a aplicação cliente.

   O caso de uso é encerrado.

4.2.8 Caso de uso – Buscar requisição de serviço

Uma aplicação servidora que está sem uma tarefa para fazer, interage com o IeC para buscar uma tarefa, recebendo uma quando disponível.

Fluxo básico

O caso de uso é iniciado quando uma aplicação servidora solicita uma requisição para resolvê-la.

1. **Recebimento de solicitação de requisição**

   O sistema recebe uma solicitação de requisição de uma aplicação servidora contendo o ID da aplicação e o ID do serviço desejado.

2. **Validação da requisição**

   O sistema verifica se o ID da aplicação é válido e se está associado ao serviço desejado.

3. **Busca de requisição**

   O sistema verifica na lista de requisições pendentes para o serviço solicitado se existe alguma.

   Se existir, o sistema busca a primeira requisição da fila, entregando para a aplicação servidora e retirando a requisição da fila. A requisição é composta pelo pacote da aplicação, o endereço da máquina solicitante e o ID da requisição. Não existindo uma requisição, é entregue um pacote vazio.

   O caso de uso é encerrado.

Fluxos Alternativos

4. **Falha na validação**

   O fluxo alternativo inicia se a validação feita no passo 2 do fluxo básico falhar.

   O sistema informa o problema para a aplicação cliente.

   O caso de uso é encerrado.

4.2.9 Caso de uso – Enviar comunicado

Representa uma comunicação entre nós integrantes do sistema rodando o IeC. Sempre que um nó necessita enviar uma requisição, uma resposta ou informação sobre a sua situação de carga, este envia um comunicado para o outro nó. O diagrama de atividades apresentado na Figura 16 mostra o fluxo deste caso de uso.

Fluxo básico

O caso de uso é iniciado quando um nó do sistema envia um comunicado ao IeC.

1. **Parser no pacote de comunicado**
   Realiza um parser no pacote (XML) recebido, o dividindo nos elementos: requisição, resposta, carga.
2. **Trata requisições**
   Para cada elemento de requisição recebido, o sistema o insere em uma lista de requisições pendentes para o serviço solicitado.
3. **Trata respostas**
   Para cada resposta recebida, o sistema verifica se esta ainda encontra-se pendente. Estando pendente, o sistema armazena na lista de respostas para a aplicação solicitante e remove a pendência.
4. **Trata informação de carga**
   Para a informação de carga recebida, o sistema atribui o tamanho da fila recebido para a máquina que enviou o comunicado.

Capítulo 5 – Avaliação

5.1 Base experimental

Com fins a avaliar preliminarmente a arquitetura construída, dois tipos de problemas foram submetidos à arquitetura: (i) a busca de soluções ótimas de problemas do tipo RCPS (*Resource-Constrained Project Scheduling*) [32] através de um esquema de enumeração implícita com *branch-and-bound* [33] [25]; (ii) a geração de uma distribuição de soluções para diversos cenários de um problema do tipo RCPS estocástico através de simulações de Monte Carlo.

Na busca de soluções ótimas de problemas do tipo RCPS, o objetivo é descobrir como escalonar as tarefas de um projeto tal que este tenha o menor tempo de duração possível. As tarefas podem possuir uma relação de precedência uma com as outras, e necessitam de recursos, os quais, em nosso experimento, são finitos, porém, renováveis. O algoritmo consiste em enumerar todos as possibilidades de escalonamento e achar a melhor delas. Isto significa percorrer uma árvore para achar a folha que represente o melhor resultado. Como é possível avaliar que alguns ramos da árvore jamais serão a solução, estes são podados, diminuindo o espaço de busca. A escolha de problemas RCPS para avaliar a arquitetura **IeC**, como veremos adiante, se justifica pela dinâmica da carga imposta ao conjunto de máquinas participantes. A estratégia *Master-Worker* (Figura 3) com hierarquia é utilizada para se distribuir os trabalhos entre os diversos *trabalhadores*. Cada máquina (nó), recebe um ramo para resolver, descobre os ramos do próximo nível (sub-ramos) e, se estiver em um estado de produtor, solicita a outros trabalhadores para resolvê-los, ou, caso esteja no estado consumidor, adiciona-os a uma pilha interna para buscá-los posteriormente. Cada trabalhador responde para seu respectivo *mestre* o melhor escalonamento conseguido a partir da situação recebida. Isto é feito combinando-se o próprio trabalho com o melhor resultado informado por seus trabalhadores, se houver.

A geração de uma distribuição de soluções para diversos cenários de um problema do tipo RCPS estocástico é um tipo de problema em que a duração de uma atividade não é determinística, sendo baseada em uma função de distribuição. Para descrever a função de distribuição de uma atividade, utilizamos três valores de tempo: o

menor tempo de execução possível da atividade, o maior, e o tempo mais provável. Estes três tempos determinam três pontos que são os parâmetros de uma função triangular de distribuição. A solução do problema consiste em resolver o problema de RCPS estocástico através de simulação de Monte Carlo. Para isto, cada cenário é resolvido utilizando-se uma heurística, e ao fim da execução de *n* cenários, temos como resposta uma distribuição para o resultado. Foi utilizado neste problema a estratégia convencional de Master Worker (Figura 2).

Para resolver os problemas descritos, foram feitos quatro experimentos, sendo três envolvendo o primeiro problema, e um, o segundo problema, conforme descrito nas seções seguintes.

5.1.1 Experimento 1 – Solução Ótima com controle estático de grão

Foi construída uma aplicação para resolver o problema de se encontrar a solução ótima do RCPS utilizando um método estático para estabelecer se o nó encontra-se em um estado produtor (em que são enviadas requisições para o IeC de forma que outros nós possam explorar os sub-ramos encontrados), ou consumidor (em que os sub-ramos encontrados são inseridos em uma pilha da aplicação para serem explorados). Nesta estratégia, todos os nós iniciam no estado produtor, permanecendo neste estado até que receba o número de pacotes configurado para o parâmetro espalhamento. Nesse estado, os nós exploram a árvore utilizando busca em largura, procurando abrir ramos longes da folha para tentar manter uma uniformidade no tamanho do grão. Após a fase de espalhamento, os nós passam ao estado de consumidor, utilizando uma pilha de tamanho conceitualmente infinito para o problema. Devido à utilização de pilha, a busca na árvore passa ser em profundidade, o que economiza recursos de memória dos nós.

A aplicação foi executada cinco vezes, utilizando-se, respectivamente, 2, 4, 6, 8 e 10 máquinas, utilizando um problema RCPS com quinze atividades gerados pelo PROGEN, que é um conhecido gerador de redes de atividade RCPS, e o valor 100 para o parâmetro espalhamento.

5.1.2 Experimento 2 – Solução Ótima com controle dinâmico de grão

Foi construída uma aplicação para resolver o problema de se encontrar a solução ótima do RCPS utilizando um método dinâmico, utilizando o conceito descrito na Seção 4.1.7., para estabelecer se o nó encontra-se em um estado produtor (em que são enviadas

requisições para o IeC de forma que outros nós possam explorar os sub-ramos encontrados), ou consumidor (em que os sub-ramos encontrados são inseridos em uma pilha da aplicação para serem explorados). Nesta estratégia, todos os nós iniciam com suas pilhas internas com valor igual a zero. A aplicação de cada nó pergunta ao IeC a situação da dispersão, aumentando o tamanho da pilha nas situações de baixa dispersão, e diminuindo-a nas situações de média e alta (com velocidade maior quando alta). Para evitar que a pilha alcance um tamanho maior do que a realidade necessária para o problema, a pilha cresce somente quando sua ocupação está igual, ou maior, do que noventa porcento. A mudança entre os estados consumidor e produtor acontece de acordo com a disponibilidade da pilha. Se a pilha interna não possuir capacidade para receber um sub-ramo, o nó será produtor, caso contrário, será consumidor.

A aplicação foi executada cinco vezes, utilizando-se, respectivamente, 2, 4, 6, 8 e 10 máquinas, utilizando o mesmo problema RCPS que foi utilizado no experimento 1.

5.1.3 Experimento 3 – Solução Ótima do Patterson13

O experimento consiste em se executar a aplicação desenvolvida para o experimento 2 com a rede de atividades do problema conhecido como Patterson13 [34]. Este problema é uma rede de atividades complexa utilizada como *benchmark.* A rede de atividades do *benchmark* é composta por 20 atividades, mais duas atividades fictícias no início e fim do projeto. Este *benchmark* é muito utilizado na avaliação da eficiência de heurísticas de escalonamento de projetos.

O problema foi rodado uma única vez, com 10 máquinas, tendo como finalidade principal a validação do algoritmo implementado para os experimentos 1 e 2.

5.1.4 Experimento 4 – Monte Carlo para o RCPS estocástico

Foi construída uma aplicação Master-Worker em que o *Master* divide o problema RCPS em x pedaços com y simulações em cada pedaço, sendo x e y configuráveis, e os envia para o IeC como requisições a serem resolvidas pelos *Workers*. O *Master* recebe as respostas e as consolidada.

Foi utilizado um projeto de 120 atividades para este experimento, sendo x igual a 30 e y igual a 1000.

5.1.5 Medidas

Os seguintes parâmetros foram medidos nos experimentos:

(i) tempos de execução real e serial em função do número de máquinas utilizadas;

(ii) distribuição de carga nas máquinas durante o tempo de execução;

(iii) Número de requisições em que o problema foi partido.

O tempo de execução real é o tempo do início da execução real do problema até o final, medido pela máquina que iniciou o problema. Dado que cada máquina *i* gasta um tempo x_i processando dados para aplicação, o tempo serial é o somatório dos tempos x_i de todas as máquinas, ou seja, o tempo que seria gasto se nenhuma máquina trabalhasse em paralelo. O tempo ideal consiste na divisão do tempo serial pelo número de máquinas utilizadas, que seria o tempo decorrido se todas as máquinas permanecessem trabalhando ao mesmo tempo durante todo o experimento. A partir dessas medidas temos as seguintes medidas derivadas:

- tempo ideal – É o tempo ideal de resolução do problema, que seria o tempo gasto se todas os nós computacionais gastassem o mesmo tempo de processamento. É definido como sendo o tempo serial divididos pelo número de máquinas que participaram da rodada do experimento;
- SpeedUp (Aceleração) – Representa a equivalência de processamento com o processamento de uma máquina. Por exemplo, se foram utilizados 10 máquinas para realizar um trabalho, e o SpeedUp medido foi 8. Isto significa que as 10 máquinas tiveram potência computacional equivalente a 8 máquinas. É definido como sendo a divisão do tempo serial sobre o tempo real.

5.1.6 Hardware

Todos os experimentos foram realizados utilizando-se até 11 máquinas AMD Athlon 1,1 Ghz, com 256 Mbytes de memória e conectadas em uma rede Ethernet a 10 Mbps.

5.2 Resultados

5.2.1 Experimento 1 – Solução Ótima com controle estático de grão

A seguir, da Figura 17 à Figura 21, temos os gráficos das distribuições de pacotes para os cenários de 2, 4, 6, 8 e 10 processadores para a descoberta da solução ótima do problema RCPS utilizando o controle estático de grão.

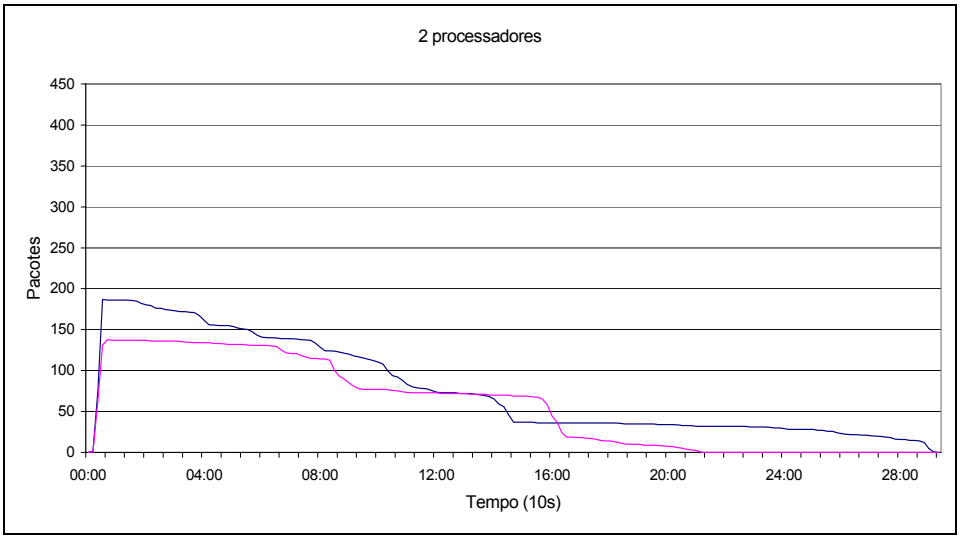

Figura 17 - Distribuição de carga com 2 processadores, controle estático do *buffer* e 100 de espalhamento.

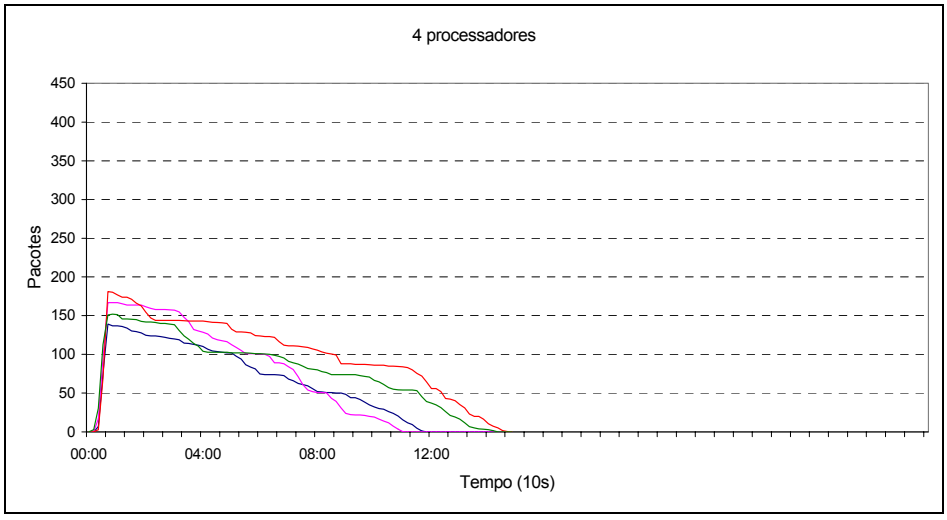

Figura 18 - Distribuição de carga com 4 processadores, controle estático do *buffer* e 100 de espalhamento.

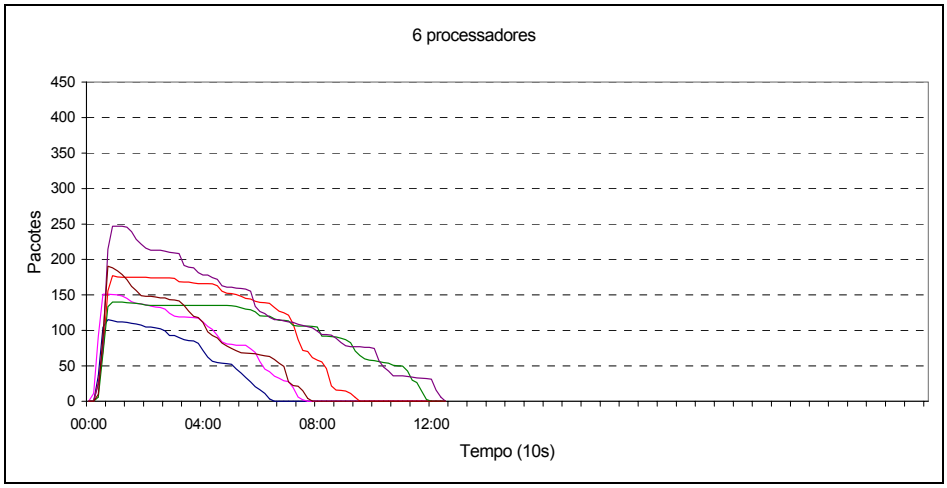

Figura 19 - Distribuição de carga com 6 processadores, controle estático do *buffer* e 100 de espalhamento.

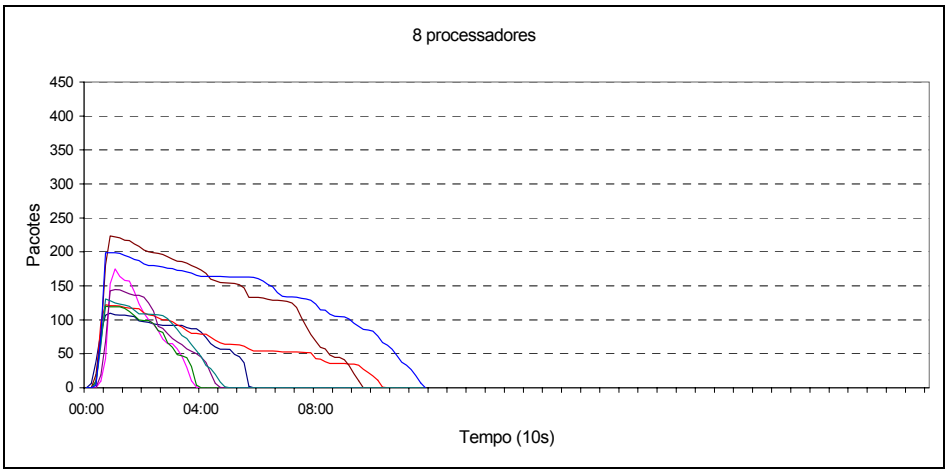

Figura 20 - Distribuição de carga com 8 processadores, controle estático do *buffer* e 100 de espalhamento.

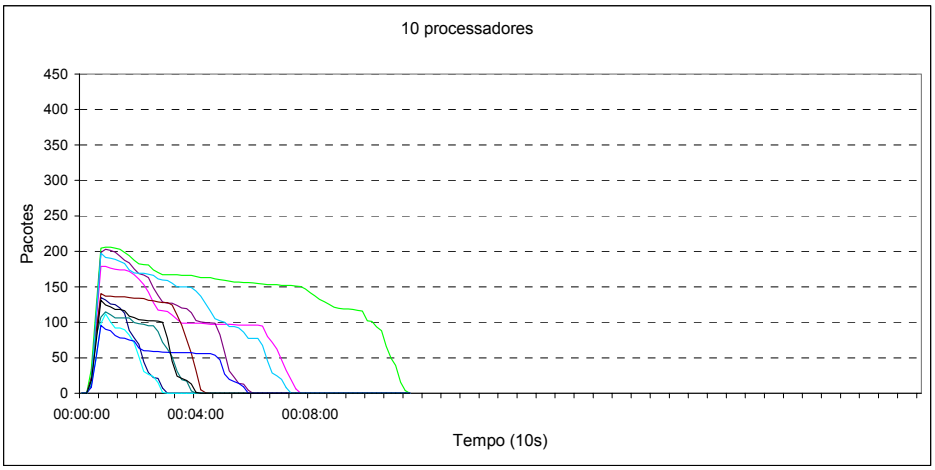

Figura 21 - Distribuição de carga com 10 processadores, controle estático do *buffer* e 100 de espalhamento.

5.2.2 Experimento 2 – Solução Ótima com controle dinâmico de grão

A seguir, da Figura 22 à Figura 26, temos os gráficos das distribuições de pacotes para os cenários de 2, 4, 6, 8 e 10 processadores para a descoberta da solução ótima do problema RCPS utilizando o controle dinâmico de grão.

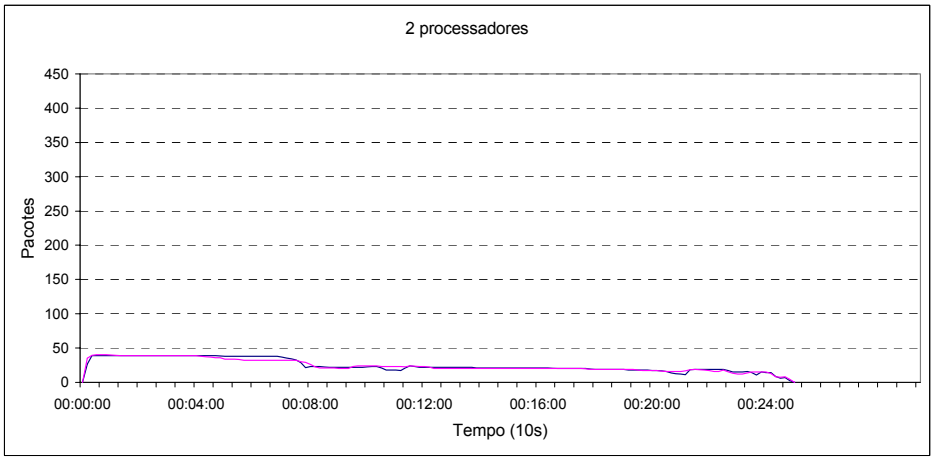

Figura 22 - Distribuição de carga com 2 processadores e controle dinâmico do *buffer*.

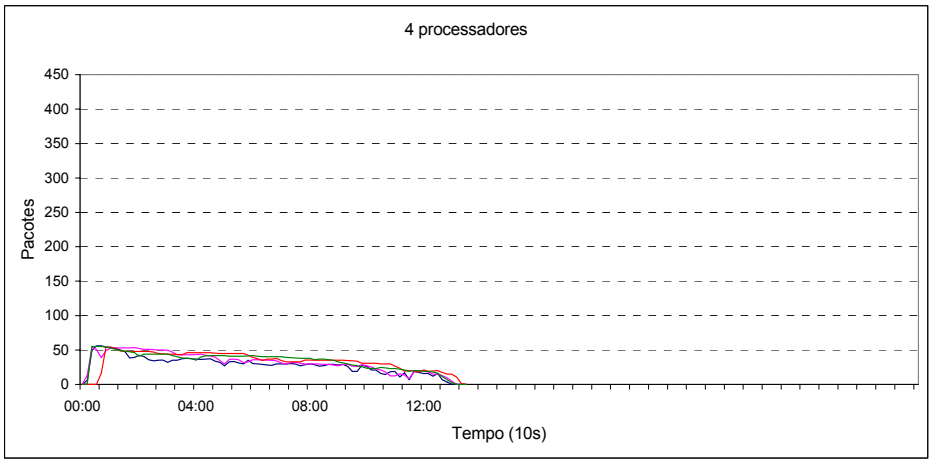

Figura 23 - Distribuição de carga com 4 processadores e controle dinâmico do *buffer*.

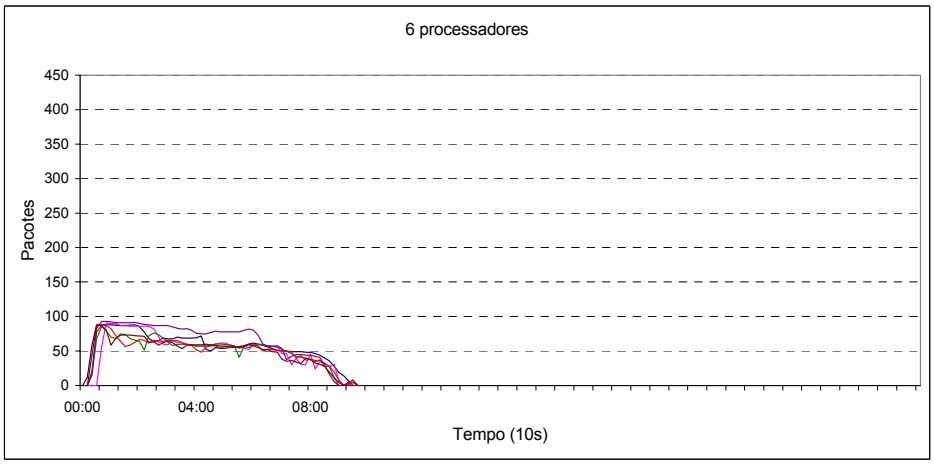

Figura 24 - Distribuição de carga com 6 processadores e controle dinâmico do *buffer*.

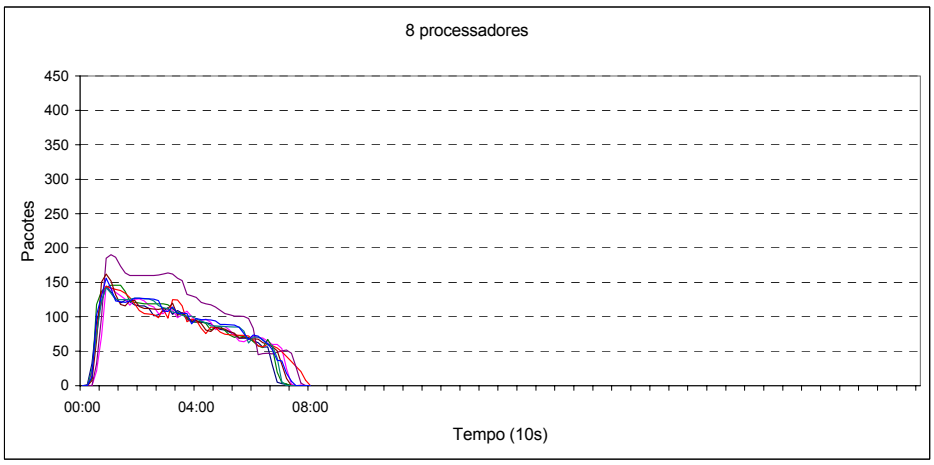

Figura 25 - Distribuição de carga com 8 processadores e controle dinâmico do *buffer*.

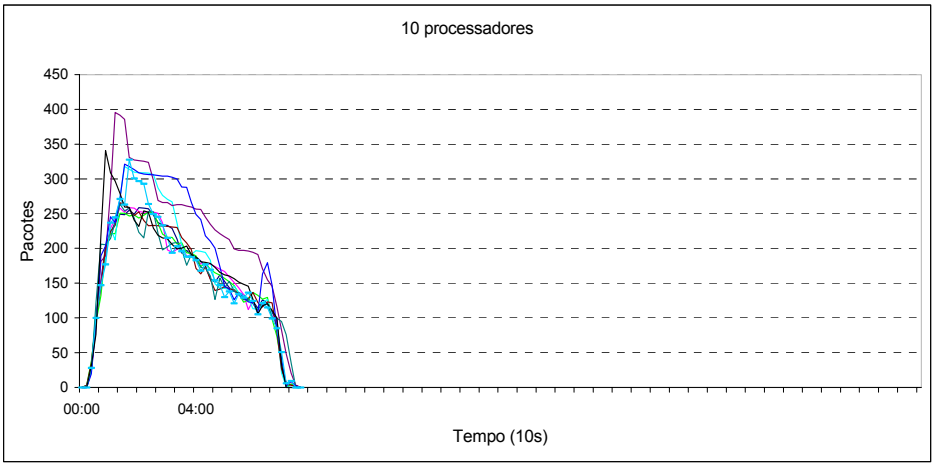

Figura 26 - Distribuição de carga com 10 processadores e controle dinâmico do *buffer*.

5.2.3 Experimento 1 x Experimento 2

Na Figura 27, temos o gráfico comparativo entre o tempo de resolução do problema com o controle dinâmico e estático.

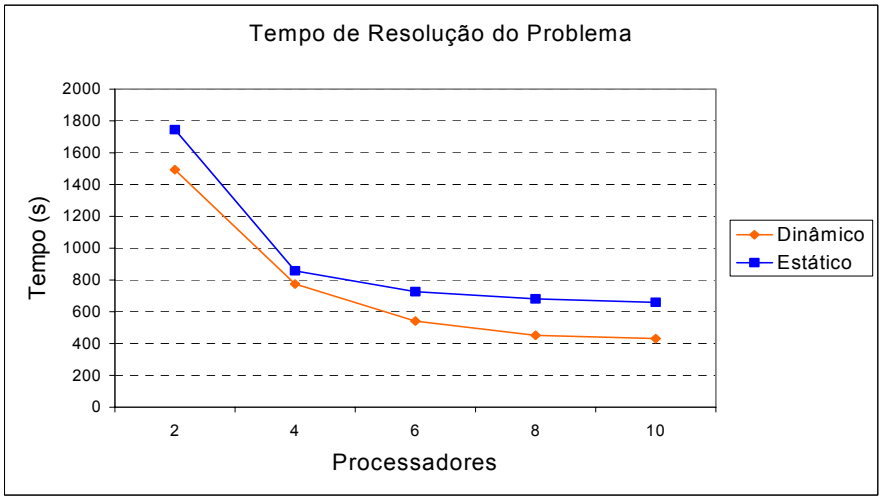

Figura 27 - Comparação do tempo de resolução do problema.

Na Figura 28 e Figura 29 temos o comparativo entre os tempos de processamento real, serial e ideal,. com controle estático e dinâmico.

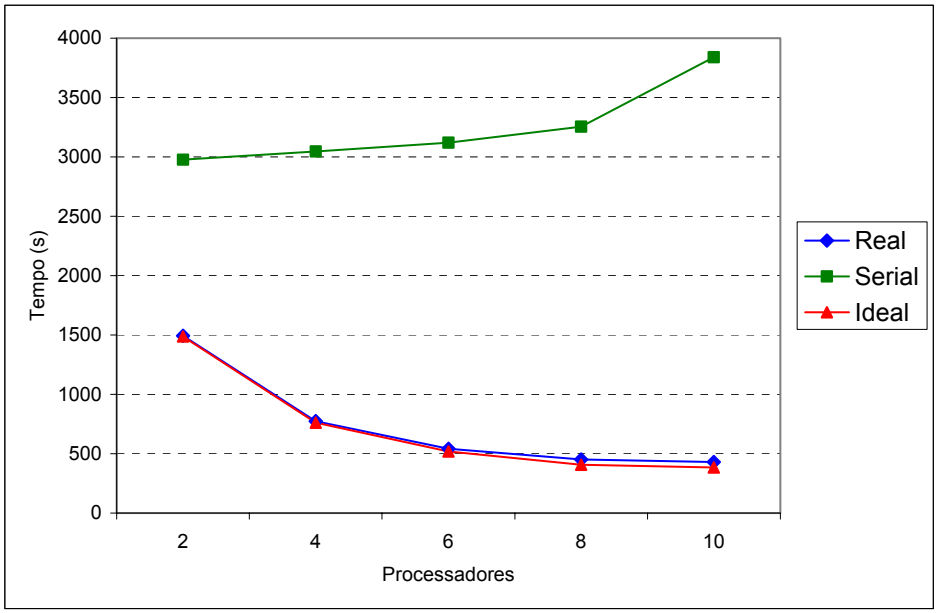

Figura 28 - Variação dos tempos de processamento real, serial e ideal com controle dinâmico do *buffer*.

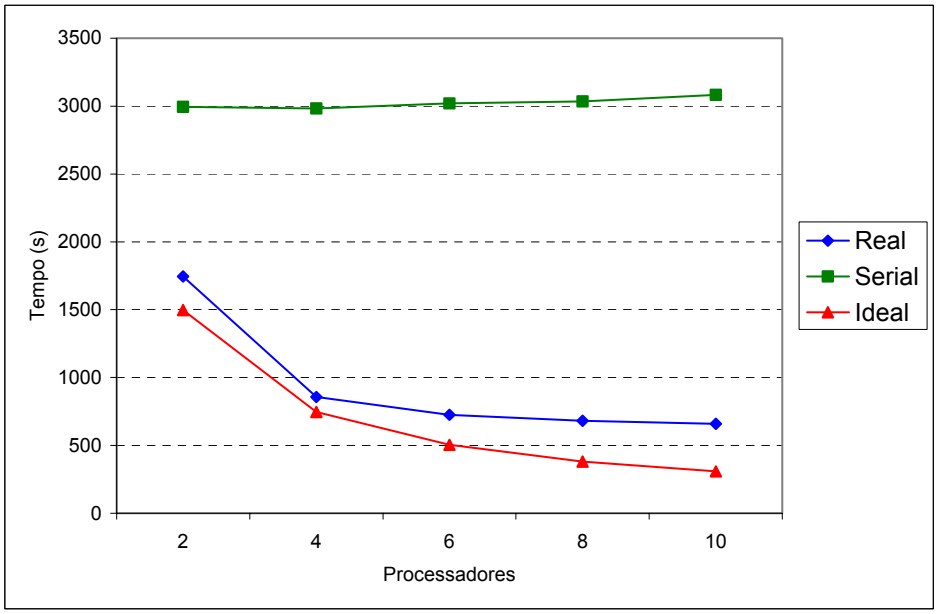

Figura 29 - Variação dos tempos de processamento real, serial e ideal com controle estático do *buffer*.

No gráfico da Figura 30 temos o comparativo entre o *speed-up* obtido com o controle estático e dinâmico do *buffer*. O gráfico mostra ainda a linha do valor ideal para o *speed-up*.

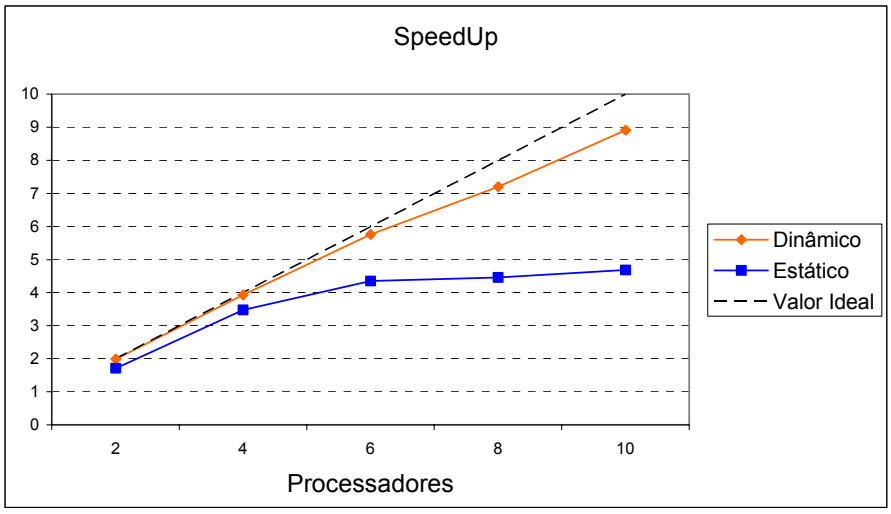

Figura 30 - *Speed-up* do algoritmo.

Tabela 8 mostram as medidas feitas para o problema de se achar a solução ótima do RCPS.

Tabela 7 - Tempos para se achar a solução ótima para o problema RCPS.

Dinâmico			
Máquinas	Real (s)	Serial (s)	SpeedUp
2	1493,355	2976,582	1,993218
4	774,572	3046,374	3,932977
6	541,558	3120,018	5,761189
8	451,884	3255,178	7,20357
10	430,739	3840,092	8,915125
Estático			
2	1745,306	2993,371	1,715098
4	857,186	2983,627	3,480723
6	726,043	3020,814	4,350726
8	681,524	3035,582	4,454109
10	658,957	3083,655	4,6796

Tabela 8 - Número de requisições geradas para se achar a solução ótima para o problema RCPS .

Máquinas	Requisições (Estático)	Requisições (Dinâmico)	Média máquina (Estático)	Média máquina (Dinâmico)
2	378	179	189	89,5
4	740	951	185	237,75
6	1165	2114	194,17	352,33
8	1426	3205	178,25	400,63
10	1801	9960	180,1	996

A Figura 31 mostra de forma gráfica a evolução do número de requisições utilizadas para resolver o problema da solução ótima do RCPS utilizando a estratégia estática de espalhamento e a dinâmica.

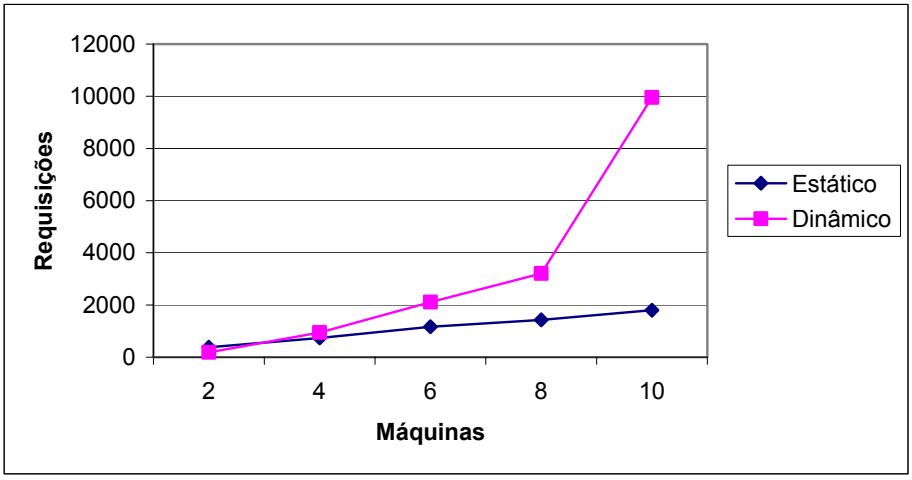

Figura 31 - Requisições geradas em função do tipo de espalhamento e número de máquinas.

5.2.4 Experimento 3 – Solução Ótima do Patterson13

Na Tabela 9 mostramos os resultados medidos para a resolução do problema Patterson13. Os valores foram obtidos utilizando 10 processadores e o controle automático do *buffer*.

Tabela 9 –Resultados obtidos rodando o Patterson13 com 10 máquinas.

Número de atividades	22
Resultado ótimo	20 unidades de tempo
Tempo para solução ótima	59min
Tempo serial	9h 42min
Speed-up	9,8

5.2.5 Experimento 4 – Monte Carlo para o RCPS estocástico

A Figura 32 apresenta o tempo serial, real, e serial para a resolução do problema RCPS estocástico.

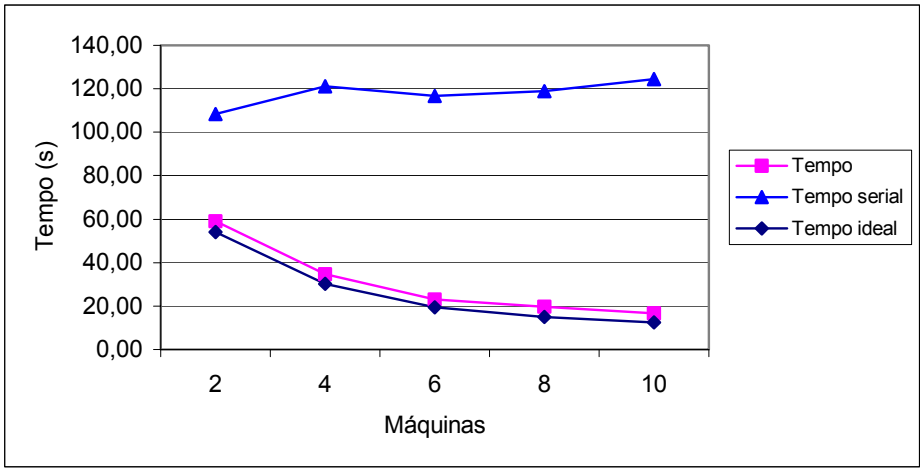

Figura 32 – Tempo de resolução do problema estocástico em função do número de processadores.

A Figura 33 apresenta a comparação do Speed-up medido com o ideal.

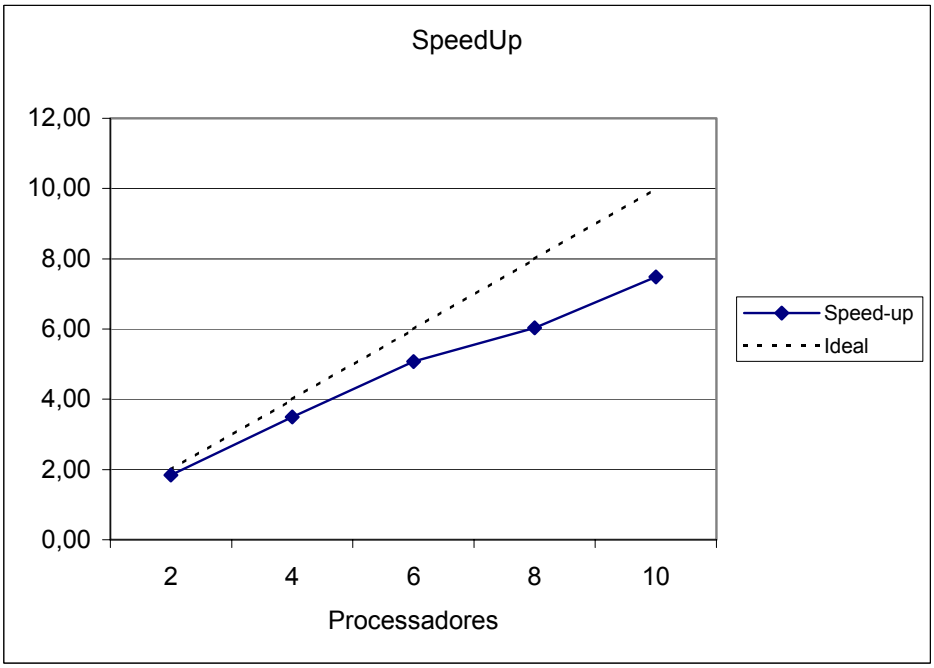

Figura 33 - SpeedUp em função do número de processadores.

A Figura 34 mostra a distribuição de trabalho para a rodada do experimento com dez máquinas (processadores). Duas máquinas receberam duas requisições, sete receberam três, e uma recebeu cinco.

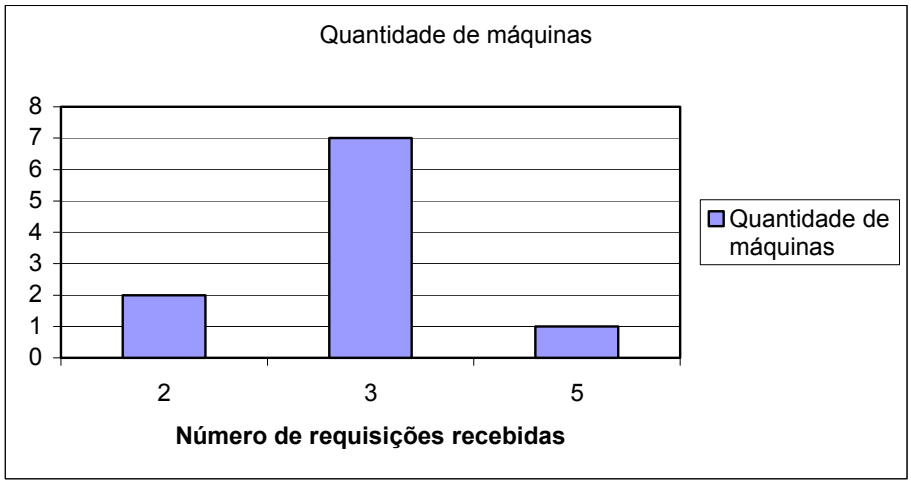

Figura 34 - Distribuição das requisições entre os nós.

A Tabela 10 apresenta as medições feitas ao se rodar a aplicação para cálculo do RCPS estocástico.

Tabela 10 – Medidas para o problema RCPS estocástico.

Máquinas	Tempo	Tempo serial	Speed-up
2	58,91	108,36	1,84
4	34,60	121,05	3,50
6	22,99	116,66	5,07
8	19,70	118,93	6,04
10	16,61	124,35	7,48

Capítulo 6 – Discussão

6.1 Escalabilidade

Os resultados apresentados na Figura 28, Figura 29 e Figura 32 sugerem um resultado interessante quanto a escalabilidade, pois, tanto na procura da solução ótima do problema do RCPS (solução ótima) utilizando ambas estratégias de espalhamento (estática e dinâmica), como na simulação de Monte Carlo para o RCPS estocástico (solução estocástica), o tempo real segue a mesma tendência do tempo ideal. Entretanto, todas as soluções caminham para um limite assintótico do melhor tempo. Para a solução estocástica, isto é óbvio, dado que o número de grãos é fixo, igual a trinta, logo, adicionar mais do que trinta máquinas no sistema significa deixar máquinas ociosas e, conseqüentemente, abaixar o *speed-up* medido. Já para a solução ótima, quanto mais máquinas, maior o número de requisições geradas, ou seja, menor o grão utilizado. Isto acontece em ambas as estratégias de espalhamento (ideal e estática), podendo ser verificado na Tabela 8 e na Figura 31. Como resultado da diminuição do grão, temos aumentado o tempo serial, que inclui o *overhead* de se empacotar e desempacotar pacotes do IeC e comunicação com o IeC, como visto na Figura 28 e na Figura 29. Podemos concluir que em um certo momento, adicionar mais máquinas não reduzirá mais o tempo ideal porque o poder computacional incluído não será maior do que o *overhead* adicionado devido à diminuição do grão.

Em tempo, na contabilidade do tempo serial, o tempo gasto em comunicação via TCP/IP não está incluído, somente os tempos de execução locais às máquinas participantes o são. No entanto, quando medido o tempo real, esses tempos de comunicação estão sendo contabilizados. Ou seja, a Figura 28, a Figura 29, e a Figura 32 são ligeiramente pessimistas com relação ao desempenho real da arquitetura IeC.

6.2 Distribuição de carga

Da Figura 17 à Figura 26 temos a distribuição de carga para os experimentos de solução ótima. Já a distribuição para a solução estocástica para o caso de dez máquinas é observado na Figura 34.

Para a solução ótima com espalhamento estático, podemos observar que em todas as distribuições, as máquinas recebem uma carga inicial nos primeiros instantes da rodada do experimento e esta carga vai sendo consumida durante o experimento. O comportamento da distribuição está de acordo com esperado, visto que sabemos que nesta estratégia, cada máquina assume um papel produtor no início, passando para consumidor somente ao receber as primeiras 100 requisições.

Dois fatos principais são possíveis de serem observados nos gráficos de distribuição. O primeiro é que o pico de carga de cada máquina varia, e esta variação aumenta conforme aumento do número de máquinas. Por exemplo, na rodada de quatro máquinas, Figura 18, a máquina com menor pico, representada pela linha azul, recebeu cerca de 140 requisições, enquanto a de maior, representada pela linha vermelha, recebeu cerca de 180 requisições. Na rodada com 10 máquinas, Figura 21, a com menor pico, linha azul, recebeu cerca de 100 requisições e a de maior, linha verde, cerca de 200 requisições. O segundo fato é que taxa de queda de carga não é constante durante o tempo, sendo que a taxa média entre uma máquina e outra também não é a mesma. Este fato pode ser observado em todas as distribuições. Por exemplo, na rodada com seis máquinas, Figura 19, a máquina representada pela linha verde é a segunda com menor distribuição. Entretanto, esta é a penúltima a ficar sem carga, ou seja, a taxa de queda de carga foi menor do que as quatro máquinas que receberam mais requisições.

O primeiro fato pode ser explicado devido ao mecanismo de escalonamento utilizado combinado com a forma utilizada para o espalhamento que é feito numa janela pequena de tempo. Como o escalonador depende da atualização da informação de carga para uma escolha apropriada, algumas máquinas com informação de carga desatualizada enviam carga para outras que possuem mais carga que outras.

A explicação para o segundo fato, visto que em nosso ambiente controlado sabemos que todas as máquinas estão trabalhando especificamente para o experimento e possuem a mesma configuração, é que as requisições não possuem tamanhos iguais. Isto é verdade, visto que estamos falando de uma busca em uma árvore em que cada requisição representa a busca a partir de um nó desta árvore, sendo que os nós possuem

profundidades diferentes, além de complexidade variável e possibilidade de serem podados devido à utilização da estratégia *branch and bound*. Logo, mesmo que o espalhamento inicial de tarefas fosse perfeito, ainda assim a estratégia estática não conseguiria manter todas as máquinas trabalhando até a solução do problema. Portanto, o melhor resultado obtido com a estratégia dinâmica (Tabela 7) não é nenhuma surpresa, pois, esta estratégia utiliza o conhecimento do ambiente para otimizar a distribuição de carga.

Observamos nos gráficos de distribuição dinâmica que não ocorre um pico de espalhamento inicial, sendo feito novos espalhamentos durante toda a execução do problema, causando com que praticamente todas as máquinas fiquem sem carga no mesmo instante. Isto mostra que o sistema se adapta aos dois fatos citados, redistribuindo carga quando outras máquinas começam a ficar ociosas devido ao primeiro ou segundo fato.

Uma conseqüência da estratégia dinâmica é o aumento não linear do número de requisições conforme o aumento do número de máquinas, mostrando que quanto maior o número de máquinas, menor deve ser o grão para que se consiga que todas trabalhem o tempo todo. A Figura 31 mostra a evolução do número de requisições geradas tanto para a estratégia dinâmica como para a estratégia estática. Na estratégia estática vimos um crescimento linear do número de requisições geradas, o que reflete o mesmo comportamento linear do crescimento do tempo serial (Figura 29). Já na dinâmica, o crescimento não é linear, tendo também como reflexo o crescimento não linear do tempo serial (Figura 28).

Para a solução estocástica, temos todas as requisições com exatamente o mesmo tamanho, portanto, a chave para escabilidade está no escalonamento. Na Figura 34 podemos ver a distribuição das requisições para o experimento com 10 máquinas. O esperado, ecalonamento perfeito, era que todas as máquinas recebessem três requisições, porém, não aconteceu para três máquinas. 7 receberam as 3 requisições esperadas, duas receberam 2 requisições e uma recebeu 5 requisições. O resultado não foi perfeito, mas foi muito bom, visto que em trinta requisições, apenas duas foram direcionadas para máquinas erradas.

Capítulo 7 – Conclusões e trabalhos futuros

7.1 – Conclusões

Nesta monografia propomos uma arquitetura XML colaborativa P2P, a IeC, que tem a intenção de tornar fácil a construção e a implantação de aplicações colaborativas P2P. Foi dada ênfase a um modelo dinâmico de escalonamento totalmente distribuído, e no controle da granularidade dos pacotes gerados com base na informação da carga no sistema. É importante notar que a resolução de problemas RCPS via *branch-and-bound* se encontraria como alvo de uma exploração de paralelismo com granularidade comparável às abordagens consolidadas por soluções do tipo *cluster computing*. Os resultados do experimento realizado indicam que a plataforma é potencialmente escalável, e mostra o benefício que uma aplicação pode obter ao controlar o tamanho do grão de forma dinâmica.

7.2 – Trabalhos Futuros

As seguintes melhorias poderão ser implementadas em futuras versões:

- Segurança – Para o funcionamento da plataforma em ambientes não controlados deve ser construído um protocolo de autenticação e criptografia de forma que os dados transferidos não sejam violados ou corrompidos por indivíduos mal intencionadas na rede;
- Re-escalonamento de carga – O mecanismo de escalonamento adaptativo utilizado apesar de ter se mostrado eficiente nos experimentos apresentados pode ser otimizado para melhor se adaptar em ambientes em que outras aplicações concorram com as aplicações colaborativas, permitindo que requisições ainda não tratadas sejam re-escalonadas para outros nós no sistema. Por exemplo, digamos que um nó recebeu diversas requisições num período de ociosidade deste, porém, após isto, a máquina começou a realizar um intenso processamento por uso de uma aplicação local. Se durante este processamento, as requisições não foram tratadas e as demais máquinas do sistema tornaram-se ociosas, o nó ocupado poderia re-escalonar suas requisições, equilibrando o sistema novamente;

- Independência de IP – Na atual implementação da IeC, a chave utilizada para reconhecer um determinado nó, é seu IP. Este mecanismo não é adequado para situações onde um determinado nó sai da rede, retornando depois. Tal nó mesmo fora da rede pode estar realizando algum processamento ou esperando respostas, logo, é desejável que ele seja reconhecido quando voltar à rede. Isto não é possível com a chave IP, pois, muitos nós não possuem IPs fixos, tendo seu IP modificado em cada conexão à rede;
- Implantar um módulo escalonador na arquitetura JXTA – O projeto JXTA também fornece uma infra-estrutura para aplicações P2P, fornecendo mecanismos para formação de grupos de interesse, busca de nós e grupos, e um conjunto de serviços abertos de comunicação. Entretanto, tal projeto não define um módulo para escalonar trabalhos. Portanto, o módulo de escalonamento da IeC seria uma valiosa contribuição para o projeto JXTA;
- Avaliação – Este trabalho forneceu uma avaliação preliminar da arquitetura IeC, mostrando que esta é bastante promissora. Uma avaliação completa com diversas aplicações diferentes em ambientes diversos é necessária para estabelecer a eficiência, ou não, da arquitetura em várias situações.

Referências Bibliográficas

[1] SETI@home, 2003, "Search for Extraterrestrial Intelligence at home". em: http://setiathome.ssl.berkeley.edu/, acessado em 06/12/2003.

[2] Grid.org, 2001, "The United Devices Cancer Research Project", em: http://www.grid.org/projects/cancer/, acessado em 09/03/2004.

[3] Pereira, F., Lourenço, F., Schmitz, E., França, F. – "Uma Arquitetura XML para Computação Colaborativa P2P" – Quinto Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD2004), 2004.

[4] Lourenço, F., Pereira, F., Schmitz, E., França, F. – "Uma Abordagem *Branch and Bound* Para o RCPSP em um Ambiente de Computação Colaborativa" – *XII Congreso Latino Iberoamericano de Investigación de Operaciones* (CLAIO 2004), 2004.

[5] Foster, I., Kesselman, C. – "The GRID: Blueprint for a Future Computating Infrastructure". Computational Grids, Capítulo 2, 1998.

[6] N'emeth, Z., Sunderam, V. – "A Formal Framework for Defining Grid Systems" – 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 202-211, 2002.

[7] Reinefeld, A., Schintke, F. – "Concepts and Technologies for a Worldwide Grid Infrastructure" – Technical paper, Euro-Par 2002 Parallel Processing, 2400: 62-71, Springer 2002.

[8] Krauter, Klaus – "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing" – Software - Practice and Experience, 32(2): 135-164, 2002.

[9] Sarmenta, L. – "Volunteer Computing" – Tese de doutorado Massachusetts Institute of Technology, 2001.

[10] Sarmenta, L., Hirano, S. – "Bayanihan: Building and studying web-based volunteer computing systems using Java" – Future Generation Computer Systems, 15(5-6): 675-686, 1999. Special Issue on Metacomputing.

[11] Sarmenta, L. – "Sabotage-tolerance mechanisms for volunteer computing systems" –Future Generation Computer Systems, 18(4): 561-572, 2002.

[12] Foster, I., Kesselman, C., Nick, J., Tuecke, S. – "The Physiology of the Grid - An Open Grid Services Architecture for Distributed Systems Integration" – Globus Project, 2002, www.globus.org/research/papers/ogsa.pdf..

[13] Entropia, "Entropia PC Grid Computing", em: http://www.entropia.com, Acessado em 11/03/2004.

[14] Parabon, "Parabon Computation", em: http://www.parabon.com, acessado em 11/03/2004.

[15] United Devices, em: http://www.ud.com, acessado em 11/03/2004.

[16] Chunlin, L., Layuan, L. – "Agent framework to support the computational grid" – The Journal of Systems and Software, 70 (1-2): 177-187, 2004.

[17] Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z. – "Peer-to-peer computing." – Technical Report HPL-2002-57, HP Lab, 2002. http://citeseer.ist.psu.edu/milojicic02peertopeer.html

[18] Foster, I., Iamnitchi, A. – "On Death, Taxes, and Convergence of Peer-to-Peer and Grid Computing" – *2nd International Workshop on Peer-to-Peer Systems* (IPTPS'03), February 2003, Berkeley, CA.

[19] Ripeanu, M. – "Peer-to-Peer Architecture Case Study: Gnutella Network" – International Conference on Peer-to-peer Computing (P2P2001), Linkoping, Sweden, August 2001.

[20] Napster, em: http://www.napster.com, acessado em 29/08/2004.

[21] The Free Network Project, Em: http://www.freenetproject.org/, acessado em 29/08/2004.

[22] KaZaa, em: http://www.kazaa.com, acessado em 29/08/2004.

[23] Heymann, E., Senar, M., Luque, E., Liviny, M. – "Adaptive Scheduling for Master-Worker Applications on the Computational Grid" – Proceedings of the First International Workshop on Grid Computing (GRID 2000), 214-227, 2000.

[24] Goux, J., Kulkarni, S., Linderoth, J., Yoder, M. – "An Enabling Framework for Master-Worker Application on the Computational Grid" – IEEE 43-50, 2000.

[25] Lourenço, F. – "Uma Abordagem Branch And Bound para o RCPSP em um Ambiente de Computação Colaborativa" – Dissertação de mestrado. IM/NCE Universidade Federal do Rio de Janeiro, em andamento.

[26] Obitko, M. – "Introduction to Genetic Algorithms" – Em http://cs.felk.cvut.cz/~xobitko/ga/, Acessado em 30/03/2004.

[27] Iamnitchi, A., Foster, I. – "On Fully Decentralized Resource Discovery in Grid Environments" – International Workshop on Grid Computing, Denver, CO, novembro, 2001.

[28] Godfrey, B., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I. – "Load Balancing in Dynamic Structured P2P Systems" – In The 23rd Conference of the IEEE Communications Society, Infocom, 2004.

[29] The World Wide Web Consortium (W3C), "Extensible Markup Language (XML)", Em: http://www.w3c.org/XML/, Acessado em 12/05/2004.

[30] Chung, P., Huang, Y., Yajnik, S., Liang, D., Shih, J., Wang, Y. – "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer". Em: http://www.research.microsoft.com/~ymwang/papers/HTML/DCOMnCORBA/S.html, acessado em 31/08/2004.

[31] The World Wide Web Consortium (W3C), "XML Schema", Em: http://www.w3c.org/XML/Schema, Acessado em 12/05/2004.

[32] Herroelen, W., Demeulemeester, E., Reyck, B. – "A classification scheme for project scheduling" – in J. Węglarz (Ed.), Project Scheduling: Recent models, algorithms and applications, Kluwer, Dordrecht, 1999.

[33] Demeulemeester, E., Herroelen, W., Elmaghraby, S. – "Optimal procedures for the discrete time/cost trade-off problem in project networks" – European Journal of Operational Recearch, 88: 50-68, 1996.

[34] Patterson, J. – "A comparison of exact approaches for solving the multiple constrained resource project scheduling problem" – Manegement Science, 30: 854–867, 1984.

[35] Project JXTA(TM), em http://www.jxta.org/, acessado em 12/02/2005.