CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

EMERSON DA SILVA BORGES

GRIDS COMPUTACIONAIS: UMA PROPOSTA DE MÉTODO DE PLANEJAMENTO DE EXECUÇÃO DE APLICAÇÃO BASEADA NO TIPO DE TAREFA COM O FOCO NA ANÁLISE DO DESEMPENHO

SÃO PAULO

2012

EMERSON DA SILVA BORGES

GRIDS COMPUTACIONAIS: UMA PROPOSTA DE MÉTODO DE PLANEJAMENTO DE EXECUÇÃO DE APLICAÇÃO BASEADA NO TIPO DE TAREFA COM O FOCO NA ANÁLISE DO DESEMPENHO

Dissertação apresentada como exigência parcial para obtenção do título de Mestre em Tecnologia pelo Centro Estadual de Educação Tecnológica Paula Souza, no Programa de Mestrado em Tecnologia: Tecnologia da Informação Aplicada, sob a orientação do Prof. Dr. Maurício Amaral de Almeida.

SÃO PAULO

Julho - 2012

EMERSON DA SILVA BORGES

Borges, Emerson da Silva

B732g Grids computacionais: uma proposta de método de planejamento de execução de aplicação baseada no tipo de tarefa com o foco na análise do desempenho / Emerson da Silva Borges. – São Paulo : CEETEPS, 2012.

122 f. : il.

Orientador: Prof. Dr. Maurício Amaral de Almeida.

Dissertação (Mestrado) – Centro Estadual de Educação Tecnológica Paula Souza, 2012.

1. Grids computacionais. 2. Computação distribuida de alto desempenho. 3. Planejamento de tarefas. 4. Estudo de desempenho computacional. 5. Pré-processamento de textos.. I. Almeida, Mauricio Amaral de. II. Centro Estadual de Educação Tecnológica Paula Souza. III. Título.

EMERSON DA SILVA BORGES

GRIDS COMPUTACIONAIS: UMA PROPOSTA DE MÉTODO DE PLANEJAMENTO DE EXECUÇÃO DE APLICAÇÃO BASEADA NO TIPO DE TAREFA COM O FOCO NA ANÁLISE DO DESEMPENHO

PROF. DR. MAURICIO AMARAL DE ALMEIDA

PROF. DR. MARCOS CRIVELARO

Aristides Novelli Filho

PROF. DR. ARISTIDES NOVELLI FILHO

São Paulo, 20 de julho de 2012

Dedico este trabalho aos meus pais pelo apoio incondicional, modelo de sabedoria e padrão de valores que carinhosamente eles investiram na minha caminhada.

À Ezilda, pela fidelidade e incentivo em seguir o inestimável caminho do saber durante a nossa convivência.

À minha avó Emília pela coragem em encarar tantos desafios apresentados a ela, para priorizar o investimento na benção que é a instituição eterna familiar e, também por me ensinar um modelo de fidelidade e boa vontade ao servir as pessoas.

Ao grande Osvaldo Borges pelo modelo de homem que és, Pai. Tenho te observado como base para a minha caminhada.

Agradecimentos

Agradeço em primeiro lugar a Deus, que é dono de todas as coisas tudo e permite as estações importantes em nossa vida e, dentre elas, a sabedoria e oportunidade de desenvolver este trabalho.

Aos meus pais, pelo modelo, exemplo de vida, investimento e quão grande dedicação que dispuseram, com paciência, para que fosse possível eu chegar vitorioso até aqui. Vocês são os melhores pais com quem alguém poderia sonhar.

À querida Professora Marlene Yurgel, pelo incentivo na carreira acadêmica e encorajamento de aceitar este Projeto de Pesquisa em nível de Mestrado no renomado Centro Paula Souza. A sua semente, minha professora, gerou este motivador e gratificamente Projeto de Pesquisa na desafiadora seara de Ciências da Computação. O referido trabalho, que publico hoje com a comunidade científica, já estava nascendo quando nos conhecemos, desde o primeiro bate-papo, lá no LABARQ da Faculdade de Arquitetura e Urbanismo, Universidade de São Paulo – FAU-USP.

Ao meu orientador Maurício Amaral de Almeida, pelo inestimável apoio neste projeto, pela dedicação, compreensão dos meus momentos de dificuldades no projeto de pesquisa e, principalmente, pela sua habilidade e perícia em tornar conceitos e princípios muito complexos em singelos exemplos de situações que vivemos no dia a dia.

Ao Prof. Dr. Marcos Crivelaro e ao Prof. Dr. Aristides Novelli, componentes da banca examinadora, pelo empenho na contribuição científica, essencial desde o exame de qualificação, sem o qual esta pesquisa não atingiria o seu objetivo.

Aos professores do Mestrado em Tecnologia do Centro Estadual de Educação Tecnológica Paula Souza, em especial àqueles com quem cursei créditos: Profa. Dra Marília Macorin de Azevedo, Prof. Dr. Alfredo Colenci Jr., Prof. Dr. Napoleão Verardi Galegali, Profa. Dra Senira Anie Ferraz Fernandes e Prof. Dra. Márcia Ito.

A todos os meus bons professores, pesquisadores e profissionais que têm investido na minha carreira profissional e científica.

Ao pessoal da Secretaria do Programa de Mestrado em Tecnologia do Centro Paula Souza: Cleonice Viana Lima da Silva, Alex, Wallace, Carlos dos Santos, Marcos Bezerra, Natália Ferreira, Geraldo de Souza, Sérgio Eugênio Menino.

Aos meus amigos de caminhada e também aos colegas de Mestrado pelo compartilhamento de suas experiências em sala de aula e às madrugadas, em especial ao Edison Fontes, Carlos Palhares, Cláudio Candido, Danúbio Borba, Djalma dos Santos, José Abranches, Thiago Ferauche e Wilson Staub. Tenho aprendido muito com cada um de vocês.

Ao amigo e mestre João Staub pela amizade, investimento e jornada que você me encorajou a seguir nestes últimos dias. Vale a pena viver o melhor projeto de vida de um homem. Eu tenho observado que você é um parceiro que têm buscado a sabedoria de trazer as palavras adequadas de acordo com as estações temos vivido em família. Obrigado João.

Aos colegas da Coordenadoria de Informática do Ministério Público Federal, em especial ao Igor Queiróga, José Antônio Di Domenico e Wesley Vidal. À amiga e gentil Alice Kanaan pelo incentivo e investimento estratégico numa fase muito importante desta pesquisa: as organizações ficam mais

interessantes com a sua humildade e valorização do profissional como ser humano no relacionamento interpessoal.

Aos amigos do Coren-SP pelo incentivo e, inclusive pelo compartilhamento da experiência acadêmica e apoio prático em ambientes de infraestrutura computacional e também pela amizade, especialmente do Reginaldo José de Souza (M. Régis).

Aos colegas da Secretaria de Tecnologia da Informação do Tribunal Regional do Trabalho da 2ª Região, pelo apoio ao colega Thiago Ferauche e fornecimento das Bases de Dados Textuais, que permitiram a aplicação dela ao Estudo de Caso desta dissertação.

À minha amada e bem-vinda Laura Borges Santos, pelo amor, a dedicação e alegria que você ofertou gratuitamente na vida da nossa pequena e unida família, especialmente à partir de setembro de 2012.

E, também especialmente a Luma, pela sua contribuição em todo o período do projeto. Você me deu a impressão que sempre esteve tudo bem desde o início, especialmente a cada noite e, também, claro pela sua alegria e amor incondicional a mim dedicado.

"A mente que se abre a uma nova ideia jamais voltará ao seu tamanho original".

Albert Einsten.

"Você não pode ensinar nada a um homem; você pode
apenas ajudá-lo a encontrar a resposta dentro dele mesmo".

Galileu Galilei.

Resumo

BORGES, E. S. **GRIDS COMPUTACIONAIS: UMA PROPOSTA DE MÉTODO DE PLANEJAMENTO DE EXECUÇÃO DE APLICAÇÃO BASEADA NO TIPO DE TAREFA COM O FOCO NA ANÁLISE DO DESEMPENHO.** 2012. 122 f. Dissertação (Mestrado em Tecnologia) – Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2012.

O objetivo desta dissertação é propor uma metodologia de avaliação da viabilidade de utilização de uma aplicação em Computação Distribuída de Alto Desempenho, especialmente em *Grids* Computacionais, com base no tipo de tarefas da aplicação. Uma das técnicas para descobrir se uma aplicação é candidata a ser executada em ambientes de computação distribuída é a classificação da aplicação quanto à sua divisibilidade. O foco do estudo será o planejamento das tarefas de uma aplicação de um domínio específico para a execução nesses ambientes. Realizar-se-á uma pesquisa conceitual sobre problemas de escalonamento e alocação de tarefas e recursos nesses ambientes e, tomando por base a técnica de divisão de aplicações, propor-se-á uma metodologia de planejamento de execução das tarefas com sete etapas, a fim de se obter uma Análise de Viabilidade, sendo esta relativa à aplicação de um determinado domínio em ambientes de Grids Computacionais. O estudo aplica-se ao problema de Pré-processamento de Ementas da Justiça Trabalhista do Estado de São Paulo, utilizando-se todas as etapas, com o propósito de verificar se é possível fazer uma Análise do Desempenho Computacional por meio da aplicação do processo.

Palavras-chave: Grids Computacionais, Computação Distribuída de Alto Desempenho, Planejamento de Tarefas, Estudo de Desempenho Computacional, Pré-Processamento de Textos.

Abstract

BORGES, E. S. **GRIDS COMPUTACIONAIS: UMA PROPOSTA DE MÉTODO DE PLANEJAMENTO DE EXECUÇÃO DE APLICAÇÃO BASEADA NO TIPO DE TAREFA COM O FOCO NO ESTUDO DE DESEMPENHO.** 2012. 122 f. Dissertação (Mestrado em Tecnologia) – Centro Estadual de Educação Tecnológica Paula Souza, São Paulo, 2012.

The aim of this dissertation is to propose a methodology to assess the feasibility of using an application on Distributed Computing, especially in Computational *Grids*, based on the type of application tasks. One of the techniques to find out if an application is a candidate to run in distributed computing environments is the classification of the application as its divisibility. The study's focus is the planning of the tasks of an application for a specific domain for implementing these environments. It conducted a survey in the conceptual problems of scheduling and allocation of tasks and resources in these environments and, based on the technique of dividing applications, we propose a design methodology for performing the tasks with seven steps in order to obtain an analysis feasibility of implementing an application in a particular field in Computational Grid environments. The study is applied to the problem of Pre-processing Menus Justice Labor of the State of São Paulo, using all steps in order to verify that you can make a Performance Analysis Computational through the application process.

KEY WORDS: COMPUTACIONAL GRIDS, HIGH PERFORMANCE DISTRIBUTED COMPUTING**,** TASK PLANNING, STUDY OF COMPUTATIONAL PERFORMANCE, PRE-PROCESSING OF TEXTS.

Lista de figuras

Lista de tabelas

Lista de siglas e abreviaturas

API	Application Programming Interface
DNS	Domain Name System
GUI	Guide User Interface
HPC	HIGH PROCESSING COMPUTING
ICMP	Internet Control Message Protocol
IP	Internet Protocol
LAN	Local Area Network
NFS	Network File System
OGF	Open Grid Forum
OGSA	OPEN GRID STRUCTURE ARCHICTETURE
OSPF	Open Shortest Path First
QoS	Quality of Service
RFT	Reliable File Transfer
RLS	Replica Location Service
RMS	Resource Management and Systems
RSVP	Resource Reservation Protocol
SSI	SINGLE SYSTEM IMAGE
TCP	Transmission Control Protocol
UCP	Unidade Central de Processamento
UDP	User Datagram Protocol
VO	Virtual Organization

Sumário

Introdução

Os Grids Computacionais emergem como uma infraestrutura cibernética e Global de aplicações para a próxima geração de *e-science*, integrada em larga escala, distribuída e de recursos heterogêneos.

Entende-se por *Grid* (*Grid Computing*) um modelo e/ou arquitetura de processamento computacional, o qual constitui a infraestrutura que possibilita o modelo de computação distribuída compostas por Organizações Virtuais (OV). O surgimento das *Grids Computacionais* nasceu da comunidade de Processamento de Alto Desempenho (PAD). Tanto os *Grids* quanto os *Clusters* Computacionais são provedores de infraestrutura de Computação Distribuída de Alto Desempenho, utilizados em geoprocessamento e aplicações meteorológicas, por exemplo.

O *Grid* é um caso particular da computação distribuída, uma vez que esta infraestrutura computacional é orientada essencialmente para aplicações que precisam de uma grande capacidade de cálculos, ou enormes quantidades de dados, transmitidos de um lado para o outro, ou ambos (CHEDE, 2004).

Comunidades científicas, como é o caso de física de alta energia e de ondas gravitacionais, de geofísica, de astronomia e de bioinformática, estão utilizando *Grids* para compartilhar, gerenciar e processar grandes conjuntos de dados. Para alcançar esse potencial em Grids Computacionais, escalonamento de tarefas é uma importante questão a ser considerada (BARUAH, 2012).

O problema de pesquisa é norteado pela investigação em princípios de escalonamento de tarefas, considerado um dos principais desafios dos ambientes heterogêneos e, muitas vezes geograficamente distribuídos como o caso dos Grids Computacionais. Busca-se uma proposta de método de estudo de viabilidade de execução de aplicações na referida infraestrutura.

Diversos estudos têm sido propostos no sentido de direcionar quais são os tipos de aplicações candidatas à execução em ambientes de Grids Computacionais com

abordagem de RMS, tais como: BUYYA; ABRAMSON (2000); ASSIS (2006); CASANOVA (2002); BERMAN (2003), CASANOVA (2000); CASANOVA (2002), todos com base em políticas e algoritmos de escalonamento.

Todavia, questiona-se qual seria o método que poderia ser aplicado para se realizar o estudo sobre viabilidade de executar uma aplicação nos ambientes de Grids Computacionais.

Uma das técnicas para se descobrir se uma aplicação é candidata a ser executada em ambientes de computação distribuída é a classificação da aplicação quanto a sua divisibilidade (WILKINSON, 2010).

Nesse sentido, este trabalho objetiva elaborar uma proposta de metodologia que possibilite o estudo acerca da viabilidade de execução de uma aplicação de um determinado domínio, tomando por base o estudo da divisão da aplicação em tarefas menores.

Justifica-se o trabalho porque a utilização de Grids Computacionais enfrenta vários desafios, dentre os quais está à padronização dos *middlewares* para interoperabilidade das organizações virtuais heterogêneas, as quais se constituem de diferentes políticas de acesso aos recursos, assim como a criação de protocolos leves de comunicação, a fim de que seja possível a distribuição das aplicações para serem executadas nos pontos de cooperação que compõem os diferentes *Grids* multi-institucionais (FOSTER, 2001).

Esses ambientes são também plataformas atrativas para execuções de aplicações em larga escala e aplicações de uso intensivo (CASANOVA, 2002). Como os diversos estudos pesquisados focalizam no escalonamento de aplicações para execução em tais ambientes, o presente trabalho toma por base pesquisa desse domínio para propor uma metodologia com base no estudo de caso da etapa de pré-processamento de ementas, passo importante para a técnica de classificação das ementas do direito trabalhista, com o propósito de contribuir com o estudo de viabilidade de execução de aplicações em computação distribuída.

Desse modo, a metodologia utilizada nesta pesquisa é composta de revisão bibliográfica, a qual oferece base às sete etapas propostas para o planejamento de execução de uma aplicação em Grids Computacionais por meio da interface de sistemas RMS (Resource Management and Systems).

Nesta investigação, tal metodologia, utilizada para se alcançar o objetivo, constituir-se-á das seguintes etapas:

a) Pesquisa exploratória do problema de escalonamento em Grids Computacionais;

b) Proposta de método com sete etapas para o planejamento da execução de uma aplicação no ambiente;

c) Aplicação do método proposto à etapa de Pré-processamento de Ementas da Justiça Trabalhista de São Paulo.

Quanto à estrutura, este trabalho está organizado em sete capítulos, iniciando-se com uma breve introdução, acerca dos principais problemas de pesquisa encontrados na busca do planejamento de um estudo a respeito da viabilidade de aplicações em Computação Distribuída, especialmente no caso dos Grids Computacionais.

No Capítulo 1, será realizada uma revisão da literatura sobre o histórico desses ambientes, mostrando-se as principais características, desafios e aplicabilidade da emergente plataforma virtual de Computação Distribuída.

Na sequência, o Capítulo 2 constituir-se-á de uma pesquisa sobre as principais características da alocação de recursos e escalonamento de tarefas desse ambiente, com foco na divisão das aplicações em tarefas, concordando com a metodologia proposta na presente dissertação.

O Capítulo 3 descreverá a metodologia proposta na pesquisa. Como o estudo é aplicado ao domínio da metodologia aplicado ao universo de classificação, auxiliada pelo computador, de ementas referentes à jurisprudência da justiça trabalhista, no Capítulo 4 explicar-se-á a proposta do caso de estudo.

Em continuidade ao trabalho, o Capítulo 5 contém a descrição da aplicação do método de planejamento proposto ao caso de estudo e, por fim, no Capítulo 6, elaborar-se-á a análise e a interpretação dos resultados, apontando-se para trabalhos futuros no capítulo 7, realizando-se, ainda, considerações finais desta dissertação.

Capítulo 1: Computação Distribuída

Este capítulo apresenta a definição de computação paralela e de computação paralela distribuída. A seção inicial traz o conceito da primeira e a recente utilização da infraestrutura de computação distribuída para prover a segunda, técnica chamada de computação paralela distribuída.

Ainda na mesma seção é apresentada a classificação de arquitetura de computadores de Flynn e a inserção dos ambientes de Grids Computacionais nos conceitos de Flynn e de Computação Paralela Distribuída.

A segunda seção apresenta um breve resumo de Grids Computacionais, do seu surgimento até a concepção das atuais organizações virtuais.

Já a terceira seção apresenta uma breve pesquisa sobre os *Clusters* Computacionais, classificados também como soluções de multicomputadores.

Por fim, a quarta seção traz a pesquisa de Grids Computacionais no contexto de organização virtual, arquitetura em camadas, taxonomia, funcionalidade, *middleware,* abrangência e identificação das infraestruturas de *Grids*.

1.1. Histórico

1.1.1 Computação Paralela Distribuída

Segundo Almasi, computação paralela é

> Uma grande coleção de elementos de processamento que podem se comunicar e cooperar entre si para resolver problemas de forma mais rápida que o método sequencial. (ALMASI; GOTTLIEB, 1994)

Essa abordagem data de 1989 e é utilizada até os dias atuais pela sua abrangência na definição de Sistemas de Computação Paralela.

Em uma máquina tipicamente paralela, os processadores são interligados por barramentos de alta velocidade, compartilham a mesma memória e dispositivos de entrada e saída, e são compostos todos por um mesmo tipo de processador.

A principal desvantagem desse tipo de máquina é o seu alto custo de aquisição, que pode se tornar indesejável devido à rapidez de processamento que máquinas sequenciais têm alcançado. Entre as principais arquiteturas essas máquinas estão os SMPs (*Symmetric Multiprocessor*), as NUMA (*Nonuniform Memory Access)* e processadores vetoriais (STALLINGS, 2000).

Quando um grupo de elementos de processamento, que comunicam entre si para trabalhar em conjunto, encontra-se distribuído em máquinas distintas, está caracterizado um tipo de Sistema de Computação chamado de Sistemas Distribuídos.

De acordo com Coulouris (COULOURIS *et al.*, 2001), Sistemas Distribuídos consistem em sistemas nos quais os componentes localizados em uma rede de computadores comunicam-se e coordenam suas ações somente por meio de troca de mensagens.

A Computação Paralela, de uma maneira geral, é concebida para se atingir melhorias no desempenho, motivo pelo qual é também conhecida por Computação de Alto Desempenho. Os Sistemas Distribuídos, por sua vez, visam o compartilhamento de recursos, tanto lógicos quanto físicos, sendo que o trabalho realizado num sistema desse tipo recebe o nome de Computação Distribuída.

A grande vantagem dos sistemas distribuídos está na sua atraente relação de custo e benefício, pois eles podem ser formados até mesmo pela união de computadores pessoais. Essa característica, juntamente com a busca por maior desempenho, tem levado à realização da Computação Paralela sobre Sistemas Distribuídos, tornando-se, por esse motivo, a ser chamada de Computação Paralela Distribuída (AMORIM *et al.*, 1988; SOUZA, 2000).

Nesse tipo de processamento, um conjunto de máquinas independentes é interligado para simular um computador paralelo, compondo-se o que é chamado de máquina paralela virtual. O fato dos sistemas serem formados por máquinas independentes, que não compartilham a mesma memória, faz com que eles sejam chamados de sistemas fracamente acoplados, ao contrário das máquinas ditas tipicamente paralelas, que são chamadas de fortemente acopladas (DANTAS, 2005).

1.1.2 Classificação de Flynn

Devido à existência de uma grande diversidade de arquiteturas de computadores, inúmeras taxonomias já foram propostas, no intuito de se uniformizarem, de maneira mais coerente, as características dos diferentes sistemas Computacionais. A classificação dos ambientes de *hardware* mais aceita na área de arquitetura de computadores é a conhecida como taxonomia de Flynn (FLYNN, 1972).

A prestigiada classificação foi proposta há mais de trinta anos e, aceita ainda hoje devido a sua grande abrangência, leva em consideração o número de instruções executadas em paralelo versus o conjunto de dados para as quais as instruções são submetidas.

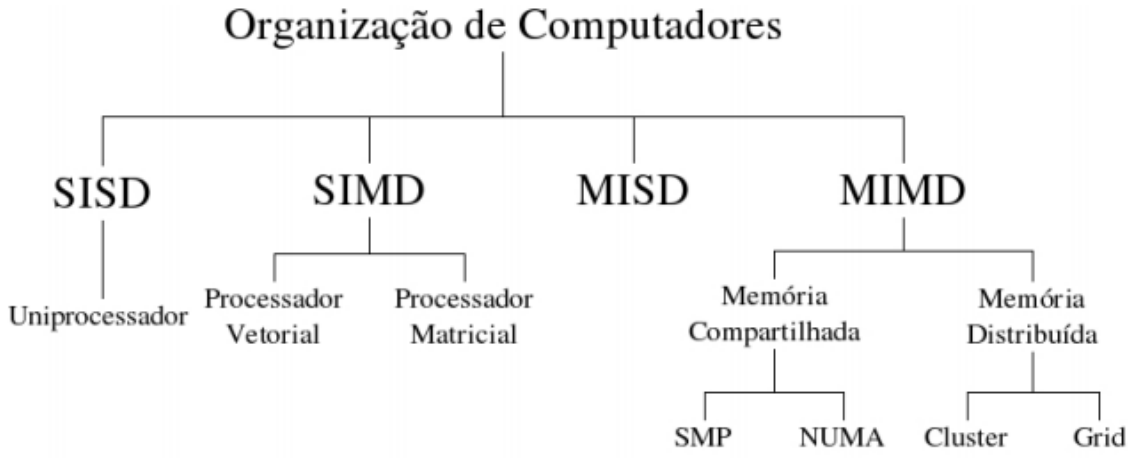


Figura 1 - Extensão da Classificação das arquiteturas de computadores segundo Flynn (REIS; SANTANA, 2005)

A Figura 1 ilustra uma extensão das quatro categorias de arquitetura existentes, segundo a Taxonomia de Flynn. As seguintes classes são formadoras dessa taxonomia:

- SISD (*Single Instruction Single Data*) – computadores com essa característica são aqueles que executam uma instrução de um programa por vez, ou seja, o modelo tradicional do processador único. Um exemplo seria o computador pessoal com um processador convencional;

- SIMD (*Single Instruction Multiple Data*) – nesse tipo de arquitetura existe também a execução de uma única instrução. Todavia, devido à existência de facilidades em *hardware* para armazenamento (um vetor ou *array*), a mesma instrução é executada sob diferentes itens de dados;

- MISD (*Multiple Instruction Single Data*) – não se tem conhecimento de arquitetura de máquinas com múltiplas instruções trabalhando com um único conjunto de dados, uma vez que, apesar de autores como Almasi (ALMASI; GOTTLIEB, 2000) considerarem as máquinas com técnica de *pipeline* representantes da classe MISD, não há nenhuma implementação que se enquadre em tal classificação;

- MIMD (*Multiple Instruction Multiple Data*) – a classe MIMD é a representante das arquiteturas que realizam múltiplas instruções simultaneamente e, por isso, a que mais tem se destacado ao longo do tempo. Nela, um conjunto de processadores executa diferentes sequências de instruções sob diferentes conjuntos de dados.

  Como é possível visualizar na **Figura 1**, a organização dos processadores MIMD divide-se nos processadores de memória compartilhada, também chamados de fortemente acoplados, e nos de memória distribuída, chamados de fracamente acoplados. Existe ainda uma nomenclatura para esse tipo de sistemas, sendo que o conjunto de processadores que compartilham a mesma memória é chamado de multiprocessador, tendo os SMPs e NUMAs como representantes e o conjunto de processadores independentes, que não

compartilham memória, chamados de multicomputadores. Nesse último grupo estão os *Clusters* e os Grids Computacionais.

1.1.3 Grids Computacionais como Computação Paralela Distribuída

Os Grids Computacionais constituem uma plataforma que se enquadra na configuração de Computação Paralela Distribuída, devido a sua utilização frequente tanto na computação de alto desempenho – High Processing Computing (HPC) quanto para o compartilhamento de recursos.

O conceito de G*rid* também se enquadra na definição de Sistemas Distribuídos de Tanenbaum (2002), a qual postula que um “Sistema Distribuído é uma coleção de computadores independentes que parecem um único computador para o usuário do sistema” (TANENBAUM; MAARTEN, 2002).

Nos sistemas de *Grids,* os recursos heterogêneos são unidos para a construção de uma grande máquina virtual, também conhecida como metacomputador (FOSTER *et al.*, 2001), utilizada pelo usuário final.

Um *Grid* geralmente é formado por diversos recursos espalhados geograficamente, e que, por esse motivo, não costumam ser homogêneos. Todavia, esse fato não impede, ainda, a união de tais componentes a um único sistema.

1.2 Grids Computacionais – Antecedentes

De acordo com o GridCafe (2012), a computação em *Grid* não emergiu do nada. Ela cresceu a partir de esforços e ideias anteriores, como é ilustrado na **Figura 2**.

A Computação em *Grid* teve origem em um *workshop* chamado "Building a Computational *Grid*", realizado no Argonne National Laboratory, situado nos Estados Unidos da América, em setembro de 1997. Depois disso, em 1998, Ian Foster do Argonne National Laboratory e Carl Kesselman da Universidade do Sul da Califórnia publicaram "The Grid: Blueprint for a New Computing Infrastructure", frequentemente chamado de "a bíblia do Grid".

Ian Foster já tinha sido envolvido no projeto I-WAY, e a dupla Foster-Kesselman havia publicado um artigo em 1997, chamado "Globus: a Metacomputing Infrastructure Toolkit", ligando claramente o Globus Toolkit com o seu antecessor, metacomputação.

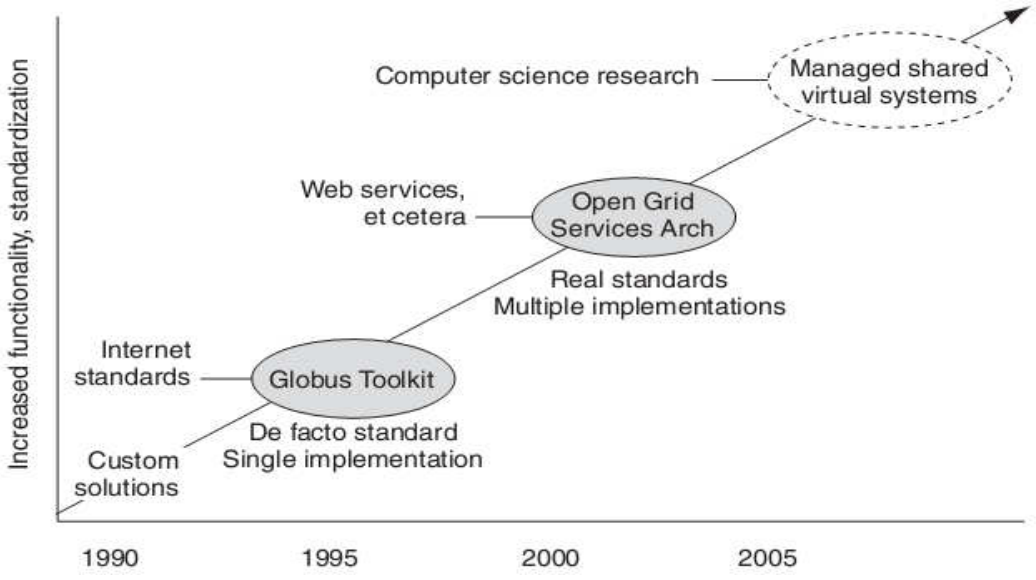


Figura 2 - The Evolution of Grid Technologies (FOSTER; KESSELMAN, 2004).

De acordo com Foster e Kesselman (2004), as tecnologias de *Grid* resultaram de dez anos de pesquisa e de desenvolvimento tanto na área acadêmica quanto na da indústria.

Essa seção contém uma breve descrição dos esforços e ideias que originaram as infraestruturas Computacionais hoje conhecidas como Grids Computacionais. Com base na **Figura 2**, é possível destacar quatro principais fases na evolução tecnológica, descritas a seguir:

1. *Custom solutions* (soluções personalizadas): iniciado nos anos 1990, esse esforço conhecido como “metacomputação” é relacionado com campos que envolvem soluções personalizadas de problemas de computação em *Grid.* O foco desses frequentes esforços heroicos era fazer trabalhos que explorassem todas as coisas que eram possíveis. Aplicações eram feitas diretamente sobre os protocolos de Internet com típicas limitações de segurança, escalabilidade e robustez. A interoperabilidade não era uma preocupação significativa.

   Segundo o GridCafe (2012), o termo “metacomputação” foi também utilizado para descrever os esforços de interconectar os centros de supercomputação dos Estados Unidos da América (EUA). A popularização do termo é creditada a Larry Smarr, diretor formador do Centro de Aplicações de Supercomputação nos Estados Unidos da América.

   São exemplos dessa fase os projetos FAFNER (Factoring via Network-Enabled Recursion), destinados à fatoração de números muito grandes (um desafio muito relevante para a segurança digital), e I-WAY (Information Wide Area Year), que visa interligar supercomputadores ligados pelas redes existentes.

   Muitas técnicas empregadas no **FAFNER** para dividir e distribuir problemas Computacionais foram utilizados nos projetos SETI@home (ANDERSON et. Al, 2002) e outros projetos atuais de computação oportunista . A utilização de um computacional resource *broker*[1], uma das inovações que o **I-WAY** trazia, influenciou a utilização desse conceito, utilizado nos Grids Computacionais atuais.

[1] Resource broker: O *Resource Broker* atua como um mediador entre o usuário e os recursos do grid usando serviços de *middleware.* Ele é responsável pelo descobrimento de recursos, seleção dos recursos, ligação de software, dados e recursos de hardware. A tarefa dele é identificar dinamicamente os recursos avaliados no sistema, então selecionar e alocar os recursos mais apropriados para um determinado *job.*

2. Globus Toolkit: em 1997 o *software* livre Globus Toolkit versão 2 (GT2) surgiu como um padrão, de fato, de Computação em *Grid*. Focalizado na usabilidade e na interoperabilidade, o GT2 definiu e implantou protocolos, APIs e serviços usados em milhares de instalações de Grid ao redor do mundo. Com a provisão de soluções para problemas comuns de autenticação, descoberta de recursos e acesso aos recursos, o GT2 acelerou a construção de aplicações real de Grids Computacionais. Também pela definição e implantação de "padrões" de protocolos e serviços, o *software* foi o pioneiro na criação da interoperabilidade de sistemas de *Grid* e possibilitou progressos significativos nas ferramentas de programação para *Grid.*

   As tecnologias do Globus Toolkit são incorporadas de quatro principais formas: segurança (GSI), serviços de informações (MDS), gerenciamento de recursos (GRAM) e gerenciamento de dados (GridFTP). Comentários semelhantes se aplicam a outras tecnologias de *Grid* importantes, que surgiram durante esse período, como é o caso do sistema de computação de alto rendimento Condor (CONDOR, 2012).

3. OGSA: em 2002 evidenciou-se a tecnologia Open Grid Services Architecture (OGSA), uma verdadeira comunidade de padrões com múltiplas implementações, incluindo, em particular, a OGSA baseada no GT3, versão de 2003. Criada sob a extensão significativa dos conceitos e tecnologias do GT2, a OGSA firmemente alinha a computação em *Grid* com as iniciativas gerais da indústria, como a Arquitetura Orientada a Serviços (SOA) e os Web Services (Serviços Web). A OGSA, além de definir um conjunto de interfaces padrão e comportamentos que resolveram muitos dos desafios técnicos introduzidos anteriormente, fornece um *framework* sobre o qual pode ser definida uma ampla gama de serviços interoperáveis e portáteis.

4. Managed, Shared, Virtual System (Organização Virtual Multi-institucional): as definições técnicas iniciais de OGSA é um passo importante, mas ainda há muito a ser feito para a resolução dos desafios de computação em *Grid*. O estabelecimento de organizações virtuais multi-institucionais interoperáveis

delimita a fase que se inicia em 2005, como ilustra a Figura 2 **- The Evolution of Grid Technologies (FOSTER; KESSELMAN, 2004).**. Trata-se do desafio que tem requerido um trabalho muito pesado para os grupos de pesquisa em Ciências da Computação, abordado desde 2003, São os que, na Figura 2, foram endereçados na arquitetura proposta por Foster e Kesselman (2004) à seção "1.1.1 Computação Paralela Distribuída".

1.3 *Clusters* Computacionais

Um *cluster* computacional é um ambiente de computação paralela, formado por um conjunto de computadores, chamados nós, interligados, muitas vezes, por dispositivos do tipo *switches* em uma Rede LAN de alto desempenho, como no exemplo da Myrinet[2] e ATM. Os nós cooperam entre si para atingir um determinado objetivo comum (Baker, 2000).

A arquitetura de um *cluster* é classificada, segundo Tanenbaum (1999), como MIMD do tipo fracamente acoplado , conforme o estudo de Dantas (DANTAS, 2003), ou seja, tem memória distribuída (multicomputadores), por isso os nós devem se comunicar a fim de coordenar e organizar todas as ações a serem tomadas. Desse modo, externamente, o *cluster* é visto como sendo um único sistema.

As principais diferenças entre os ambientes de *cluster* e os de *grid* estão indicados na tabela 1, em que se destacam a autoridade reguladora (o *cluster* tem autoridade única, já o *grid* contém múltiplas), a segurança (no *cluster* pode ser desnecessária e no *grid,* indispensável (FOSTER, *et al*., 2001) e o sistema operacional (no *cluster* deve ser homogêneo e no *grid* pode ser heterogêneo).

[2] A Myrinet é um a padrão público e aberto, publicado e registrado na ANSI (ANSI/VITA 26-1998). Esta tecnologia foi desenvolvida para prover alto desempenho, comunicação eficiente de rede e comutação com uma razoável relação custo-eficiência (Myrinet, 2005). A tecnologia tem como objetivo a formação de *Clusters* de estações de trabalho. PCs e servidores. Para este objetivo uma configuração Myrinet dispõe de switches e placas de redes especiais para interligação do ambiente de rede.

Configuração	*Cluster*	*Grid*
Domínio	Único	Múltiplos
Nós	Milhares	Milhões
Segurança do processamento e do recurso	Desnecessária	Necessária
Custo	Alto, pertencente a um único domínio	Alto, todavia dividido entre domínios
Granularidade do problema	Grande	Muito grande
Sistema operacional	Homogêneo	Heterogêneo

Tabela 1 - Diferenças entre as configurações de cluster e grid. (PITANGA, 2004).

Em um ambiente *cluster,* a alocação de recursos é efetuada por um domínio administrativo centralizado, sendo desnecessária a segurança do processo e do recurso, caso a rede de interconexão (intrac*luster*) seja desacoplada da rede de acesso externo. Além disso, esse tipo de ambiente pode se beneficiar de protocolos de comunicação mais eficientes entre suas unidades de processamento, pois, como a rede de interconexão pertence ao mesmo domínio administrativo, o recurso é controlado (BAKER, 2000; PITANGA, 2004).

A abordagem de *cluster* tem alta escalabilidade, visto que tarefas de inclusão ou exclusão de nós escravos não exigem que sejam feitas modificações no ambiente, sendo realizadas de forma isolada, partindo do nó mestre, por meio da execução de algum comando específico do *software* escolhido. No entanto, um fator limitante é o número de nós, na ordem de dezenas de recursos.

1.4 Grids Computacionais

A computação em *Grid* surgiu como um importante novo campo que se diferencia da computação distribuída convencional pelo foco no compartilhamento de recursos em larga escala, aplicações inovadoras e, em alguns casos, orientada ao alto desempenho.

Pode-se entender como primeira definição um *Grid* Computacional como um grupo de recursos heterogêneos, distribuídos e integrados, que compartilham diversos recursos como se fossem um único e que utilizam redes de altíssima velocidade (FOSTER, et al., 2001).

De uma forma geral, um *grid* é conceituado como:

> Um ambiente computacional distribuído paralelo que permite o compartilhamento, a seleção, a agregação de recursos autônomos e geograficamente distribuídos. Estas operações e recursos podem ser utilizados durante a execução de uma aplicação, dependendo de sua disponibilidade, capacidade, desempenho e custo. O objetivo é prover aos usuários serviços com os requisitos de qualidade corretos para o perfeito funcionamento de suas aplicações. (Foster(1999), Tony(2003), Globus (2005) e Grid (2005)

Tomando essa definição como base, podemos considerar que um *grid* computacional tem como seu principal objetivo alcançar interoperabilidade entre as organizações virtuais, por meio da habilidade de cooperação de compartilhamento e de agregação de recursos Computacionais distribuídos, e disponibilizá-los como recursos e serviços.

A **Figura 3** ilustra um ambiente onde o usuário faz acesso a um ambiente de *grid*. Ele tem uma determinada aplicação e deseja que ela seja executada na configuração. É muito importante observar que um usuário da Internet não dispõe da facilidade de submissão de sua aplicação.

Diferentes dos ambientes de *Clusters* e Grids Computacionais, na *Web*, os serviços são oferecidos como *pushing*, mecanismo no qual o usuário seleciona um serviço oferecido que é executado no sentido *site*-usuário. Nesse ambiente, não é possível solicitar a execução de uma determinada tarefa diferente daquelas predefinidas.

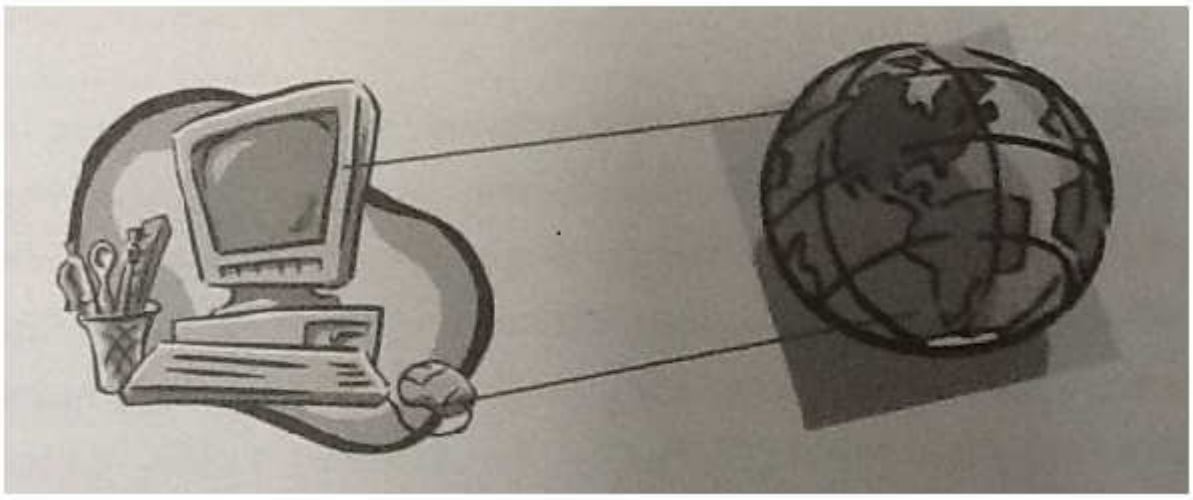


Figura 3 - Ambiente usuário e ambiente Grid. (Dantas, 2005)

Um *Grid* Computacional (FOSTER, et al., 1999) é um ambiente em que se permite a segurança de acesso, tolerância a falhas e balanceamento de carga, que é a técnica para distribuir a carga de trabalho uniformemente entre dois ou mais computadores, enlaces de rede, UCPs, discos rígidos, ou outros recursos, a fim de racionalizar a utilização dos recursos, maximizar o desempenho, minimizar o tempo de resposta e evitar a sobrecarga.

A palavra *Grid* tem sua origem no termo *electrical power grid,* o qual denota uma rede de energia elétrica, proporcionando a sua geração, transmissão e distribuição . Essa rede é a infraestrutura que possibilita o uso da energia – o recurso, nesse caso – de forma transparente, generalizada e confiável (FOSTER; KESSELMAN, 2004).

O *Grid Computacional* é uma tecnologia emergente, que mudou a forma de abordagem de problemas computacionais complexos. Assim como a Internet revolucionou a forma do compartilhamento de informações, o *Grid* Computacional, similarmente, revolucionou o compartilhamento de poder computacional e de armazenamento (NASSIF, 2006).

1.4.1 Organização Virtual

As organizações participantes de uma configuração de *grid* são conhecidas como organizações virtuais (VO). Traçando uma comparação com a Internet, a entidade

organização virtual seria semelhante a um *site*, no entanto com a possibilidade de prover serviços solicitados pelo usuário. Uma organização virtual é uma entidade que compartilha recursos sob uma determinada política em uma configuração de *grid* (Dantas, 2005).

Exemplos de organizações virtuais são empresas, centros de pesquisas que proveem facilidades de armazenamento de dados, poder de processamento e o uso de equipamentos como telescópios e aplicações (pacotes de *software* de simulação que podem executar com dados fornecidos pelo próprio usuário).

O conceito de Organização Virtual (VO), ou grupo de usuários com interesses comuns, que se organizam por meio de uma rede de longo alcance (WAN) como a Internet, é fundamental para a arquitetura dos sistemas de Grids Computacionais.

VOs nos sistemas orientados à ciência, tal como no OSG[3], tipicamente, tem uma missão orientada pela ciência em áreas tão diversas, tais como mapeamento do genoma, mapeamento ambiental e simulação da nanoeletrônica. No entanto, não há nenhum requisito para que as *VOs* sejam constituídas somente por cientistas, grupos industriais trabalhando em um novo produto, artistas que exigem capacidades de renderização distribuídas de imagens e engenheiros que simulam novas estruturas.

Esses são exemplos de grupos que poderiam formar Organizações Virtuais e utilizar serviços de *Grid* (FOSTER; KESSELMAN; TUECKE, 2001). Além disso, as organizações que fornecem recursos para o *Grid* também são organizadas em VOs, os quais tentam oferecer qualidade específica de serviço (QoS) para as metas do consumidor dos recursos, como no exemplo da **Figura 4**.

[3] OSG: **O**pen **S**cience **G**rid (**OSG**), prevê fornecer o serviço comum e suporte para provedores de recursos e instituições científicas utilizando uma malha de computação distribuída de alto rendimento em serviços Computacionais. A OSG não possui recursos próprios, mas fornece software e serviços para usuários e provedores de recursos tanto para permitir a utilização oportunista como para o compartilhamento de recursos. O OSG é financiado conjuntamente pelo Departamento de Energia (USA) e da Fundação Nacional de Ciências (USA).

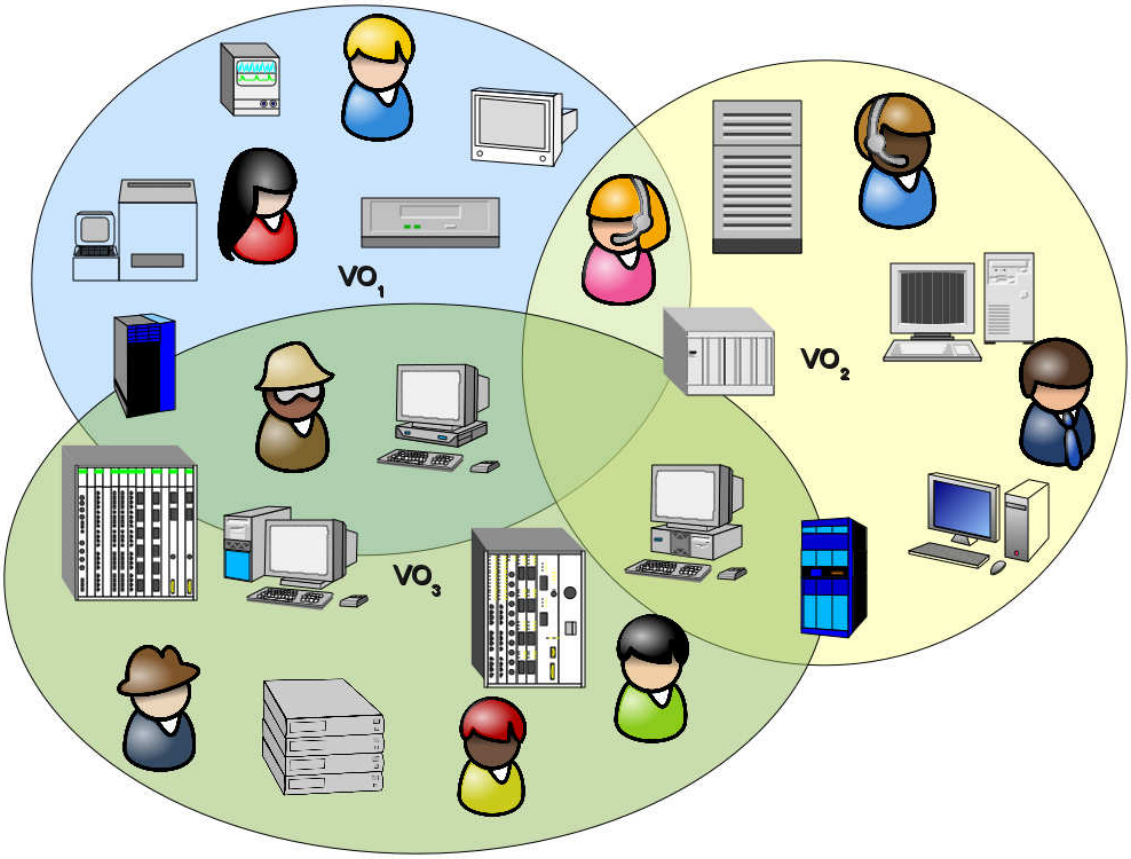


Figura 4 - Organizações Virtuais no Grid: VOs podem representar grupos de consumidores, grupos de provedores de recursos ou grupos que são ambos: provedores e consomem (MURPHY, 2010)

A distinção entre “consumidor” e “provedor” nas Organizações Virtuais não é muito clara nesse cenário ilustrado pela **Figura 4**.

1.4.2 Arquitetura

A arquitetura de Grids Computacionais identifica os componentes fundamentais do sistema, especifica o propósito e função de tais componentes e identifica como estes interagem entre si.

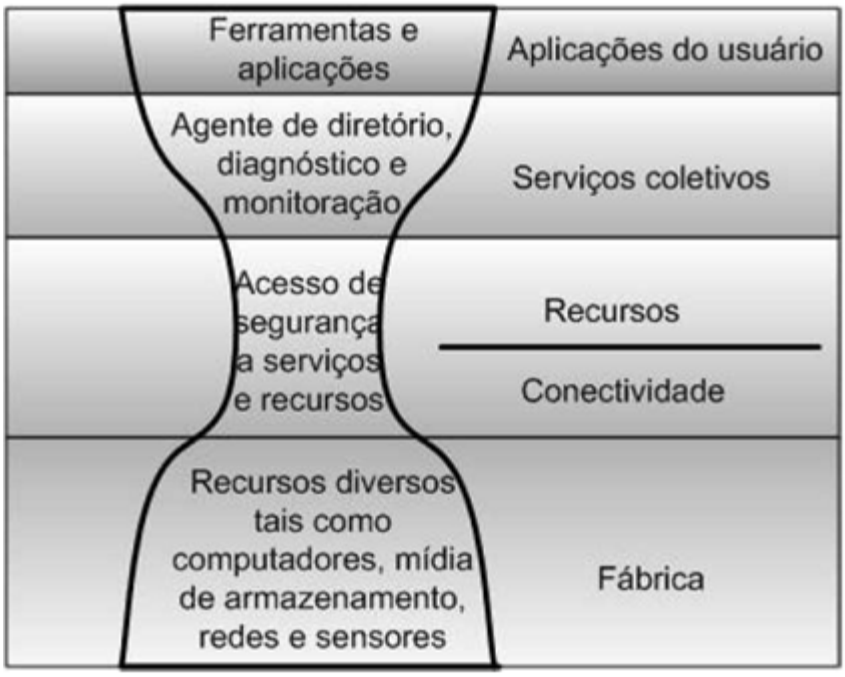


Figura 5 – A Arquitetura de Grid em Camadas (FOSTER; KESSELMAN, 2004).

1.4.2.1 Camada Fábrica

A camada fábrica engloba os recursos pelos quais os acessos compartilhados são mediados pelos protocolos do *Grid*. Exemplos dessa camada são recursos Computacionais, sistemas de armazenamento, catálogos, recursos de rede e sensores. Um recurso pode ser uma entidade lógica, como um sistema de arquivos distribuído, *pool* distribuído de computadores.

Há uma interdependência fina entre as funções implementadas em nível da camada fábrica, de um lado e, operações de compartilhamento suportados, do outro lado. Exemplos dos elementos dessa camada são os diversos recursos que podem ser lógicos ou físicos, tais como sistemas de armazenamento, discos virtuais, recursos de rede, sensores ou *Clusters* de computadores.

Os desafios nessa camada estão relacionados à implementação de mecanismos internos que permitam, por um lado, a descoberta de sua estrutura, estado e capacidade e, por outro lado, o controle da qualidade de serviços (NASSIF, 2006).

Os exemplos que seguem explicam os desafios da camada Fábrica: (FOSTER; KESSELMAN, 2004).

- Recursos Computacionais: são necessários mecanismos para iniciar programas e para monitorar e controlar a execução dos processos resultantes. Funções internas são necessárias para determinar as características de *hardware* e *software,* assim como as informações relevantes de estado , tais como carga de trabalho corrente.

- Recursos de armazenamento: nos recursos de armazenamento, é necessário o desenvolvimento de mecanismos para enviar e recuperar arquivos. Esses mecanismos geralmente são para a leitura, escrita e execução de arquivos remotos.

- Recursos de Rede: nos recursos de rede, deve ser provido o desenvolvimento de mecanismos de gerenciamento que forneçam controle sobre os recursos alocados para a transferência na rede. Funções internas devem ser providas para determinar as características de carga na rede.

1.4.2.2 Camada de Conectividade

A camada de Conectividade define os protocolos de autenticação de comunicação para transações de rede específicas de *Grid* Computacional. Os protocolos de comunicação permitem a troca de dados entre os níveis de ambiente e recursos.

Entre os requisitos de comunicação estão o transporte, o roteamento e o serviço de nomes. Os protocolos de autenticação constroem os serviços de comunicação de modo a prover mecanismos seguros e criptográficos para a verificação da identidade de usuários e recursos.

Tais protocolos são desenhados com base na pilha de protocolos *Transmission Control Protocol / Internet Protocol* (TCP/IP), tais como IP, Internet Control Message Protocolo (ICMP), TCP, *User Datagram Protocol* (UDP), *Domain Name System* (DNS), *Open Shortest Path First* (OSPF), *Resource ReSerVation Protocol* (RSVP), dentre outros. Desafios futuros de comunicação em Grids Computacionais deverão

trazer a necessidade de novos protocolos, para tipos particulares de redes dinâmicas, como, por exemplo, redes óticas de alto desempenho e redes sem fio.

Os temas de segurança para Grids Computacionais que são mais importantes atualmente são os que se referem a:

- Autenticação única: um usuário deve se autenticar somente uma vez, dispensando autenticações sucessivas para acessos a recursos ou domínios administrativos diferentes;

- Delegação: um usuário deve ter o poder de delegar a execução de um programa para os recursos para os quais ele tem autorização de uso. O programa deve ser capaz de delegar seus direitos para outro programa;

- Integração com soluções de segurança local: as soluções de *Grid* Computacionais devem ser interoperáveis, com soluções de segurança local;

- Relacionamento de confiança baseado no usuário: caso o usuário tenha permissões para executar programas no recurso A e B, ele pode habilitar o uso dos *sites* A e B simultaneamente, sem a necessidade de que os administradores de A e B interajam.

1.4.2.3 Camada de Recursos

A Camada de Recursos é construída sob os protocolos de autenticação e comunicação da camada de conectividade, e seu papel é definir os protocolos para a negociação, inicialização, controle, contabilização e pagamento de operações compartilhadas de forma segura em recursos individuais.

As implementações desses protocolos da camada de recursos são baseadas nas funções da camada de fábrica para acessar e controlar os recursos locais. Tais protocolos concentram-se nos recursos individuais e ignoram o estado global. Duas classes principais desse tipo de protocolo podem ser distinguidas:

- Protocolos de informação: são usados para obter informação sobre a estrutura e o estado do recurso, como a configuração, a carga de trabalho corrente e a política de uso (por exemplo o custo);

- Protocolos de gerenciamento: são usados para negociar o acesso aos recursos compartilhados, especificando, por exemplo os requisitos do recurso e as operações a serem executadas, tais como a criação de processo e o acesso aos dados.

No projeto desse tipo de protocolo, é preciso ter um ponto de aplicação de política, assegurando-se que as operações solicitadas sejam consistentes com a política do recurso a ser compartilhado. Entre as questões que podem ser consideradas nesse contexto, estão a contabilização de uso do recurso e o pagamento pelo uso.

Embora muitos desses protocolos possam ser imaginados, a camada de recursos e a camada de conectividade formam o gargalo do modelo de *Grid* Computacional (**Figura 5**), por isso devem ser limitados a um conjunto pequeno de protocolos.

1.4.2.4 Camada de Serviços Coletivos

A camada de serviços coletivos endereça problemas de descoberta, seleção e alocação de recursos, segurança, política e contabilização. Essa camada contém protocolos e serviços não associados a um recurso específico. Ela pode implementar soluções para uma coleção de recursos.

Pelo fato dos componentes serem construídos um nível acima da camada de recursos e, portanto, participarem de uma camada mais “larga” do modelo de arquitetura da figura 3, a camada de serviços coletivos pode programar uma grande variedade de serviços sem adicionar novos requisitos aos recursos que estão sendo compartilhados, tais como:

- Serviços de diretório: permitem descobrir a existência de recursos compartilhados aos participantes de uma Organização Virtual (VO);

- Serviços de coalocação, escalonamento e *brokering:* permitem aos participantes de uma Organização Virtual (VO) requisitarem a alocação de um ou mais recursos para uma proposta específica e escalonar as tarefas nos recursos apropriados;

- Serviços de monitoramento e diagnóstico: monitoram recursos para detectar falhas, intrusão e sobrecarga;

- Serviços de réplica de dados: gerenciam o armazenamento de recursos para maximizar o desempenho no acesso aos dados. Exemplo de serviços de réplica de dados são os serviços *Reliable File Transfer* (RFT) e *Replica Location Service* (RLS). Tais serviços fazem parte do pacote *Data Grid Tools* (FOSTER, 2005; ALLCOCK; CHERVENAK; FOSTER, 2005).

Essa camada faz utilização de recursos *Application Programming Interface* (API) e *Software Development Kits* (SDK) específicas para facilitar a programação no m*iddleware (*FOSTER; KESSELMAN, TUECKE, 2001).

1.4.2.5 Camada de Aplicações

A camada de aplicações da arquitetura de *Grid* compreende as de usuários que operam dentro de uma Organização Virtual. Estas podem requisitar serviços de qualquer outra camada por meio de protocolos, APIs ou SDKs, conforme a especificação de *framework* utilizado em FOSTER *et al(* 2001).

1.4.3 Funcionalidade

A taxonomia de um *Grid Computacional* quanto a sua funcionalidade é dividida em três classes: (KRAUTER; BUYYA; MAHESWARAN, 2002), como na **Figura 6**.

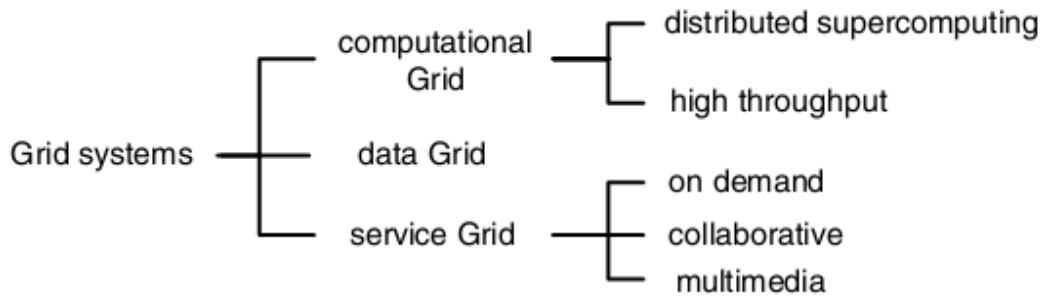


Figura 6 - A Grid System Taxonomy. (KRAUTER; BUYYA; MAHESWARAN, 2002)

- *Grids* de Processamento: essa classe de *Grid* agrega alta capacidade de valor computacional. Dependendo de como sua capacidade é utilizada, esses sistemas podem ser empregados para supercomputação distribuída ou para computação de uso intensivo – *High Troughtput Computing* (HTC).
    - Aplicações de supercomputação distribuída: utilizam um agregado de recursos computacionais para solucionar problemas que não seriam viáveis de serem resolvidos por um recurso único, devido ao tempo necessário para obtenção dos resultados. Exemplos de usos desse modelo de computação são as aplicações científicas de modelagem do tempo e simulações nucleares;
    - HTC: esse modelo de computação é utilizado geralmente em aplicações da classe *parameter sweep* (item 3.2.1.1), como é o exemplo das simulações de Monte Carlo. (ABRAMSON; GIDDY; KOTLER, 2000 e BUYYA; ABRAMSON; GIDDY, 2000).
- *Grids* de Dados: essa classe de *Grids* é utilizada para sistemas provedores de uma infraestrutura de sintetização de novas informações com base em repositórios de dados, como bibliotecas digitais ou repositórios de dados distribuídos pela Wide Area Network (WAN). Um exemplo de aplicação de tais sistemas é a mineração de dados, que correlaciona dados de muitas fontes de dados distintas e geograficamente distantes.

- *Grids* de Serviço: essas *Grids,* devido à grande generalidade das suas aplicações, são subdivididas em três classes distintas:

    - Colaborativa – nessa classe, um conjunto de usuários compartilha um recurso em comum por meio de ambientes virtuais compartilhados, sendo que esse recurso pode ser um instrumento científico ou mesmo resultado obtido de pesquisas. As aplicações dessa classe possibilitam a comunicação dos indivíduos em tempo real, permitindo aos seus integrantes trabalharem espalhados geograficamente de modo normal;

    - Multimídia – na classe de serviço multimídia é oferecida uma infraestrutura para aplicações que fazem uso de diversas mídias num mesmo intervalo de tempo, garantindo qualidade de serviço ao longo das muitas máquinas pertencentes ao *Grid*;

    - Sob demanda – a classe sob demanda presta serviços capazes de, dinamicamente, alocar mais recursos de acordo com a necessidade da aplicação. Na computação sob demanda, pode-se compartilhar processamento, *softwares*, dados e instrumentos científicos do tipo sensores e telescópios. Nessa classe, os recursos do *Grid* são compartilhados, por um tempo limitado, apenas quando os recursos locais são insuficientes, pois além do desempenho buscam um bom custo.

1.4.4 Middleware

O *middleware* é uma camada de *software* entre a camada de aplicação e a de *hardware* e *software* de mais baixo nível que é composta pelos recursos computacionais, pelos sistemas operacionais e pela rede. Utilizando esse mecanismo, o desenvolvedor de aplicações não necessita se preocupar com o gerenciamento, comunicação da camada de mais baixo nível, ficando essa função sob a responsabilidade do *middleware*.

O *middleware* em Computação em *Grid* tem como objetivo ocultar a complexidade da alocação, do gerenciamento e da comunicação de recursos computacionais.

O *middleware* para o *Grid* Computacional é entendido como sendo um *software* que conecta duas ou mais aplicações. É um conceito diferente de importar e exportar dados, pois o *middleware* conecta-se às aplicações, lendo e enviando as informações necessárias para o processamento.

Segundo Foster e Kesselman (2001), um *middleware* para *Grid* seria a união de protocolos, serviços APIs e SDKs.

Um *Grid* Computacional é, portanto, um meio de comunicação, troca de informações e compartilhamento de recursos entre aplicações.

É essencial a existência de um *middleware* para garantir a interoperabilidade entre as organizações virtuais. As organizações virtuais, para existir, precisam de mecanismos de descoberta, identidade, autorização e compartilhamento (FOSTER, 2001).

Entre os middlewares de *Grid* disponíveis destacam-se os seguintes:

- Globus Toolkit: é um *middleware* muito utilizado nos projetos de desenvolvimento de *Grid*.O recurso, segundoFOSTER; KESSELMAN(2005) é um *software* de código aberto, baseado em serviços de construção do *Grid*. Ele fornece uma API e protocolos para a criação das aplicações e dos sistemas de *Grid*. O Globus é mantido pelo Globus Alliance, composto por diversas instituições acadêmicas;

- gLite (gLite, 2011): é um *middleware* para a construção de sistemas e desenvolvimento de aplicações em *Grid*. É mantido principalmente pelo Enabling *Grids* for E-Science (EGEE) da União Europeia;

- Condor: o *middleware* Condor (CONDOR, 2011) é um software de *Grid open-source* que permite aos usuários enviar trabalhos de uma forma confiável para redes remotas e sistemas de lote, incluindo Globus, Condor, NorduGrid, UNICORE, PBS e LSF. Tem origem na Universidade de Wisconsin-Madison, nos EUA;

- Legion: é um projeto de *software* baseado em objetos, da Universidade de Virginia-EUA. Esse *middleware* aborda questões como escalabilidade, facilidade de programação, tolerância a falhas, segurança, autonomia local, entre outros recursos. É um projeto em andamento, planejado para suportar grandes graus de paralelismo no código do aplicativo e gerenciar as complexidades do sistema físico para o usuário (LEGION, 2012);

- InteGrade (GOLDCHLEGER *et al*., 2004) e Ourgrid (ANDRADE, 2003): são *middlewares* brasileiros relacionados à computação oportunista em *Grid*. O objetivo destes é aproveitar o tempo ocioso dos recursos para executar aplicações;

- UNICORE: ele oferece um sistema de *Grid* que faz a distribuição de computação e recursos avaliados num caminho seguro. A última versão implementa vários padrões abertos que alcançam interoperabilidade e integram forte capacidade de segurança e workflow. O UNICORE pode trabalhar com diferentes sistemas operacionais e provê uma interface para a maioria dos sistemas em *batch*. O projeto envolve França, Alemanha, Suiça e Reino Unido (UNICORE, 2010).

1.4.5 Abrangência

Segundo CHEDE (2004), o valor de um *Grid* difere de acordo com a sua abrangência. De maneira geral, podemos classificar os *Grids* de acordo com a sua abrangência, e estimamos que sua evolução possa se dar de forma similar ao que ocorreu com a computação em rede, a qual evoluiu de redes locais (LAN), e *intranets* internas às empresas, para extranets, com redes de empresas parceiras se interconectando, chegando finalmente às conexões amplas, efetuadas pela Internet.

Os Grids Computacionais são classificados quanto a sua abrangência em três grupos:

- Local: são *Grids* internas a uma única organização, seja em nível departamental ou alcançando toda a organização. Também podem ser chamadas de *IntraGrids*, *Enterprise Grids* ou Campus *Grids;*

- Regional: são *Grids* formados entre organizações parceiras ou que tenham interesse comum. Também são chamadas de *ExtraGrid* ou *partner;*

- Global: são *Grids* amplas, que abrangem várias localidades. Elas têm base na infraestrutura da Internet e alguns especialistas já começam a questionar se no futuro projetos como o TeraGrid seriam embrião para a criação de *Grids* chamados de GGG (Great Global Grid), uma rede independente da WWW (Word Wide Web).

1.4.6 Identificação

Para fazer a identificação se uma tecnologia implementada trata-se de um *Grid* Computacional ou não, há necessidade de se verificarem as seguintes características propostas por Foster(FOSTER, 2002), fazendo o seguinte *checklist*:

- Os recursos não podem estar subordinados a um controle centralizado: como um *Grid* integra e coordena recursos e usuários de diferentes domínios de controle, diferentes unidades administrativas da mesma organização, ou mesmo de diferentes organizações, com abordagens de segurança, contabilização, ele não deve ter um mecanismo de controle único, senão teremos um sistema de gerenciamento local e não um *Grid;*

- O *Grid* é criado com base em interfaces e protocolos com abordagem em questões fundamentais como autenticação, descoberta de recursos e acesso a eles, sendo importante que sua interface e protocolos sejam abertos e padrões. Do contrário, teremos um sistema de aplicação específica, e não de muitos propósitos;

- Entrega de serviços de alta qualidade: o objetivo do *Grid* é fornecer vários serviços de qualidade, relacionados, por exemplo, com tempo de resposta, *thoughput,* alta disponibilidade, segurança e coalocação de muitos tipos de recursos heterogêneos para atender a demandas complexas de usuários, tendo o sistema combinado de ser maior que a soma das partes, a fim de que sua utilização seja justificada.

Capítulo 2: Escalonamento e Alocação de Tarefas em Computação Distribuída

O objetivo deste capítulo é introduzir os problemas de escalonamento e alocação de tarefas em Computação Distribuída, conceitos estes que atuam em conjunto, apesar de serem abordados muitas vezes de maneira distinta em boa parte da literatura sobre o assunto.

A compreensão básica dos referidos problemas é essencial ao desenvolvimento desse trabalho de pesquisa devido à necessidade de um prévio conhecimento dos limites impostos pela infraestrutura de execução de uma aplicação no ambiente de planejamento.

Nesse sentido, inicialmente, far-se-á a introdução da necessidade de escalonamento de recursos e tarefas nos principais ambientes computacionais, inclusive no escopo dos *Grids*. Após a introdução, realizar-se-á a classificação dos principais escalonadores de computação utilizados, tanto em computação local quanto distribuída.

Em sequência, efetuar-se-á a classificação das aplicações quanto à divisibilidade das tarefas, passando-se a serem explicados os sistemas gerenciadores de tarefas (RMS) e suas características de interface com as aplicações especialmente por meio da submissão, monitoramento e gerenciamento de *Jobs.*

Por fim, na seção 2.4, explicitar-se-á a metodologia geral de escalonamento de tarefas e recursos em Grids Computacionais proposta por SCHOPF (2012), que é uma das principais referências de modelo para a proposta de metodologia desse trabalho de pesquisa.

Nos Sistemas de Computação Distribuída, é importante que os recursos disponíveis sejam utilizados, possibilitando-se a otimização de certos critérios, como é o exemplo da minimização do tempo de resposta dos aplicativos ou a maximização do *throughput* do sistema. A maneira de garantir a boa utilização dos recursos está em grande parte associada ao escalonador de recursos (TANENBAUM, 1995). O

escalonador tem o objetivo de garantir que os recursos do sistema sejam atribuídos às tarefas (consumidores dos recursos **Figura 7**) de modo a aperfeiçoar certo critério. No caso geral, os recursos são dos mais variados, como é o exemplo dos dispositivos de armazenamento, unidades de processamento ou largura de banda.

Ressalta-se que em ambientes de Computação Distribuída, especialmente em Grids Computacionais, o escalonamento tem sido estudado devido a sua grande importância para o desempenho geral do ambiente. Um caso particular, e muito importante de escalonamento, sobretudo para o escopo deste trabalho, é o escalonamento de tarefas.

Nesse cenário, os recursos a serem considerados são as unidades de processamento e os consumidores de tais recursos são as tarefas computacionais que devem ser executadas no sistema de computação distribuído.

Desse modo, o objetivo do escalonamento é associar uma unidade de processamento para cada tarefa, buscando-se a otimização de um critério especificado. Esse problema vem sendo estudado amplamente pela comunidade de pesquisa, e ficou provado que, em geral, ele é um problema NP-Completo (GRAHAM *et al.*, 1979).

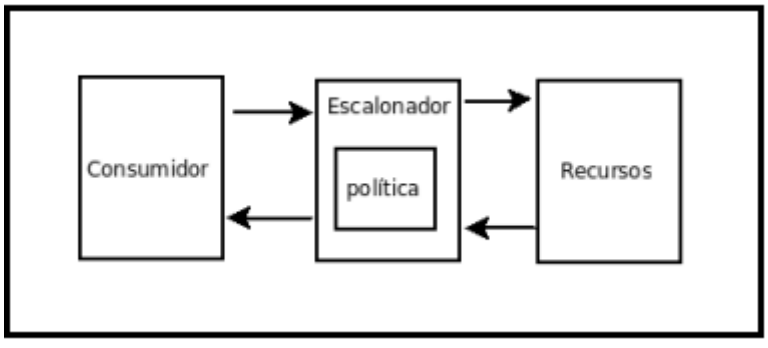


Figura 7 - Sistemas de Alocação (CASAVANT; KUHL, 1988) tradução do autor.

Torna-se importante lembrar que o problema de escalonamento de tarefas não é exclusividade dos sistemas de computação distribuídos, uma vez que um

computador monoprocessado também usa um escalonador a fim de determinar a maneira com que a CPU será utilizada pelas diversas tarefas que podem ser por ela executadas. No entanto, em sistemas monoprocessados, o escalonamento deve determinar apenas quando uma tarefa terá a CPU a sua disposição (TANENBAUM, 1995). Já no caso dos sistemas distribuídos, o problema consiste em se determinar quando e onde as tarefas irão ser executadas.

2.1 Classificação dos Sistemas de Escalonamento

Para que os sistemas de escalonamento possam ser melhor estudados, há necessidade de classificá-los de acordo com as suas características. Dessa forma, é possível estabelecer comparações entre escalonadores e fazer afirmações sobre um escalonador com base nas suas características ou limitações.

Com o intuito de contemplar o maior número possível de características existentes nos algoritmos de escalonamento de processos, diversos autores têm sugerido, em seus estudos, taxonomias para a área de escalonamento de processos (CASAVANT; KUHL, 1988, Shirazi *et al*., 1995, XU; LAU, 1997, LÜLING et al, 1993, LÜLING; MONIEN, 1993).

Dentre as diversas taxonomias propostas, destaca-se a de Casavant (CASAVANT; KUHL, 1988) por ser a mais abrangente e também de grande aceitação entre as comunidades científicas. A estratégia de escalonamento proposta por Casavant é apresentada na **Figura 8**.

Figura 8 - Modelo de escalonamento hierárquico (CASAVANT; KUHL, 1988) tradução do autor.

Segundo a taxonomia hierárquica de Casavant (CASAVANT; KUHL, 1988), o nível mais alto é dividido em local e global. O escalonamento local atribui intervalo de tempo de um único processador, enquanto o global refere-se ao local (qual processador) em que o processo será executado.

O nível abaixo do global é subdividido em escalonamentos estáticos e dinâmicos. No escalonamento estático, as decisões do escalonamento são determinadas inicialmente. Já no caso do escalonamento dinâmico, as informações dos recursos e das tarefas são variadas, por isso as decisões de escalonamento não são determinadas de início mas no momento em que acontece a execução.

O escalonamento dinâmico é aplicado quando as tarefas chegam dinamicamente e há dificuldade de fazer a estimativa do custo das aplicações. Assim, é necessário coletar as informações dos estados dos recursos para se fazer uma estimativa. Em seguida, é realizado o processo de decisão com base nas informações, determinando-se em qual recurso a tarefa deve ser executada (DONG; AKL, 2006).

O escalonamento estático pode ser ótimo ou sub-ótimo. O primeiro pode ser utilizado quando se conhece bem o estado e recursos do sistema e, quando é possível computacionalmente, utilizar o critério de seleção adotado.

Em alguns casos, o objetivo é minimizar o tempo de execução de um processo e maximizar a utilização de um recurso ou número de tarefas processadas por unidade de tempo. Entretanto, é inviável computacionalmente descobrir a melhor solução, podendo-se aplicar uma solução sub-ótima para o problema.

Já o escalonamento Aproximado é satisfeito quando encontrada uma boa solução, podendo-se utilizar algumas métricas para a sua otimização.

No caso do escalonamento Heurístico, as suposições são baseadas no conhecimento prévio do processo e das características do sistema. Um exemplo de escalonadores baseados em Heurística para Grids Computacionais é encontrado em ASSIS et al (2006) .

No escalonamento dinâmico, as suposições são feitas com base em poucas informações sobre o ambiente onde o processo será executado.

Enquanto no escalonamento estático a decisão é realizada antes de o processo ser executado, no dinâmico as decisões são tomadas posteriormente, com o ambiente já em execução.

No subtipo de escalonamento dinâmico, têm-se o escalonamento não distribuído se a tarefa puder ser realizada em um único processador, e o distribuído se a tarefa puder ser distribuída fisicamente entre mais de um processador.

Atualmente, o escalonamento distribuído pode ser aplicado a um computador moderno, que contém um ou mais processadores ou mais de um núcleo de processamento.

Esse tipo de escalonamento pode ser aplicado para vários computadores, podendo ser subdividido em Cooperativo (quando os componentes distribuídos colaboram entre si) e o Não cooperativo (quando os processos tomam decisões independentemente dos resultados dos outros). O cooperativo é dividido em sub-ótimo e ótimo, com aproximação e heurística, mencionadas anteriormente.

Para Casavant e Kuhl (1988), há também mais duas características que os sistemas de escalonamento podem ter:

• Adaptativo e não-adaptativo: sistemas adaptativos são baseados em algoritmos e parâmetros que alteram sua política de acordo com o comportamento dinâmico do sistema. Nos sistemas não-adaptativos, o mecanismo de controle não mudam necessariamente com base nas atividades do sistema. Assim, os pesos dos parâmetros permanecem os mesmos. No escalonamento do primeiro grupo, os algoritmos e os parâmetros usados para implementar a política de escalonamento podem mudar dinamicamente de acordo com o comportamento atual e anterior do sistema;

• Balanceamento de Carga: o objetivo dessa característica é tentar manter um equilíbrio de cargas entre os processadores, fazendo com que estes tenham aproximadamente a mesma taxa de trabalho para ser processada.

2.2 Alocação de Tarefas em Computação Distribuída

Em um ambiente de computação distribuída, para que uma aplicação seja executada, ela é geralmente “quebrada”, ou dividida em partes menores denominadas tarefas.

Com a aplicação dessa técnica, cada tarefa em potencial pode ser alocada para execução em um nó distinto, respeitados os requisitos mínimos de CPU, memória, sistema operacional, dispositivos de armazenamento, entre outros. Dessa maneira,

tira-se proveito do paralelismo existente entre tais ambientes, para que a aplicação como um todo seja executada mais rapidamente. A alocação de tarefas determina, portanto, em qual nó cada tarefa será executada.

Em um problema de alocação de tarefas geralmente não existem restrições de precedência temporal entre elas (CASAVANT; KUHL, 1988). Por outro lado, as políticas de escalonamento determinam o mapeamento versus tarefa, considerando as restrições temporais em um Grafo Acíclico Dirigidas (DAG).

Para Casavant e Kuhl (1988), escalonar é um problema de gerenciamento de recursos. Basicamente um mecanismo ou uma política é usada para, eficientemente e efetivamente, gerenciar o acesso e o uso de um determinado recurso.

Todavia, de acordo com o *OGF* (ROEHRIG; ZIEGLER; WEDER, 2012), escalonamento é o processo de ordenar tarefas sobre os recursos computacionais e ordenar a comunicação entre as tarefas; assim sendo, ambos, aplicações e sistemas, devem ser escalonados.

2.2.1 Classe de Aplicações

Uma das principais abordagens para o escalonamento de tarefas consiste no desenvolvimento de escalonadores especializados em determinados tipos de aplicações (BERMAN, 1999).

Segundo Budenske e Ramanujan (1997), existem escalonadores voltados para aplicações de um determinado domínio. Por outro lado, existem escalonadores que são especializados em classes de aplicações que tenham uma estrutura em comum. Muitas dessas classes de aplicações já foram identificadas, tais como *parameter sweep*, *bag-of-tasks e workflow.*

A vantagem que há em se desenvolver tipos de escalonadores específicos para um determinado tipo de aplicação é que torna-se possível incorporar ao processo do algoritmo de escalonamento características do tipo da aplicação, o que possibilita uma maior previsibilidade no comportamento da aplicação e o uso de políticas de escalonamento especializadas (BERMAN, 1999).

A utilização da classificação de aplicações por tarefas colabora com o estudo de taxonomia de um *Grid Computacional* quanto a sua funcionalidade, dividida em três classes: *Grids* de Recursos, *Grids* de Dados e *Grids* de Serviços (KRAUTER; BUYYA; MAHESWARAN, 2002,) descrita na seção 2.3.3.

Essa técnica também se aplica a outros modelos de computação distribuída, como é o caso dos *Clusters* Computacionais. Isso ocorre porque muitas vezes as Organizações Virtuais (VO) que compõem os Grids Computacionais, geralmente multi-institucionais, formadas por recursos heterogêneos, são compostas por conjuntos de *Clusters* Computacionais.

Para o escopo desse trabalho, a abordagem de classificação de tarefas é a base para a metodologia proposta na pesquisa.

2.2.1.1 Aplicações Parameter Sweep

Para que uma aplicação seja classificada como *parameter sweep,* ela pode ser definida por um conjunto em que T { t1, t2, ..., t_n } de n tarefas sequencialmente independentes.

Nesse contexto, independência significa dizer que não há nenhum tipo de comunicação, ou relação de precedência entre as tarefas que compõem a aplicação. Além disso, todas as n tarefas realizam o mesmo tipo de processamento.

A única diferença entre uma tarefa e o processamento entre duas tarefas quaisquer, como o exemplo t1 e t_n, são os parâmetros de entrada usados pelas tarefas (CASANOVA; BERMAN, 2000).

As aplicações *parameter sweep* são, em geral, aplicações desenvolvidas para explorar um grande espaço de possibilidades de resolução de um problema. São criadas várias tarefas, cada uma resolvendo o mesmo problema, mas com o conjunto de parâmetros de entrada distintos, que juntas abrangem o espaço de parâmetros possíveis.

Por serem tarefas independentes entre si, muitos consideram que esse tipo de aplicação é ideal para serem executados em sistemas de computação distribuída como os Grids Computacionais, em que a distribuição geográfica dos recursos pode implicar em altos custos de comunicação.

Diversos autores têm proposto estudos de escalonamentos heurísticos para essa classe de aplicações como em CASANOVA et al(2000), ABRAMSON; GIDDY; KOTLER(2000), CASANOVA; HAYES; YANG(2002) e CASANOVA; BERMAN, (2003).

2.2.1.2 Aplicações Bag-of-Tasks

As Aplicações *Bag-of-Tasks* (BoT) são as aplicações paralelas compostas por tarefas que podem ser executadas independentes umas das outras, no sentido de que não há relação de precedência entre as tarefas e também não há nenhuma comunicação entre elas.

Essa classe de aplicações pode ser entendida como sendo um tipo de generalização das aplicações do tipo *parameter sweep*Entretanto,ao contrário das aplicações *parameter sweep,* nas aplicações BoT não há garantias de que as tarefas que compõem a aplicação realizam necessariamente o mesmo tipo de processamento. Assim, duas tarefas - nomeadas t1 e t2 - podem realizar processamento totalmente distintos uma das outras (ASSIS et al., 2006).

As aplicações BoT são usadas numa grande variedade de cenários, incluindo a mineração de dados, aplicações *parameter sweep* (ABRAMSON, D.; GIDDY, J.; KOTLER, 2000), (SMALLEN et al., 2000), computação biológica (STILES et al, 1998) e renderização de imagens (SMALLEN; CASANOVA; BERMAN, 2001). Há uma proposta de escalonamento para esse tipo de aplicações em CIRNE et al (2003).

2.2.1.3 Aplicações de Workflow

As Aplicações da classe *Workflow* são compostas por um conjunto de tarefas que devem ser executadas seguindo uma ordem parcial determinada por dependência de controle de dados.

Essa classe de aplicações representa todas as aplicações que podem ser representadas por um Grafo Acíclico Dirigido (DAG). Há uma proposta de escalonamento para esse tipo de aplicações em Cooper et al.(2004). Alguns pacotes RMS, como é o caso do Condor (CONDOR, 2012), suportam tal classe de aplicações.

2.3 Sistemas Gerenciadores de Recursos (RMS)

As ferramentas *Resource Management and Systems* (RMS) têm seu foco principal voltado para a gerência das tarefas e de recursos geograficamente distribuídos nas configurações de *Clusters* e Grids Computacionais (DANTAS, 2005).

Os ambientes *RMS* são muitos empregados como ferramentas de gerenciamento de recursos nos pacotes de *middleware* de *Grids* e *Clusters.* É interessante observar que, considerando uma configuração de multicomputadores, como numa rede de computadores espalhada geograficamente ou não, todos os nós têm seus próprios sistemas operacionais e, portanto um gerenciador de tarefas e recursos local.

Não existe uma preocupação com o gerenciamento da configuração como um todo (*Clusters* ou *Grid*). Dessa forma, há necessidade da utilização de ferramentas de gerenciamento de tarefas e de recursos distribuídos para cobrir a falta nativa de cooperação entre os escalonadores locais.

Seguem alguns exemplos de pacotes RMS comerciais e de uso aberto:

- Pacotes RMS comerciais:
    - *Loading Sharing Facility* (LSF): *Plataform Computing* (LSF,2012);

- *Univa Grid Engine* (SGE)[4]: (UNIVA CORPORATION, 2012);
- Tivoli Workload Scheduler *LoadLeveler*: (IBM, 2012).

- Pacotes RMS com abordagem de uso aberto:
  - Condor, da Universidade de *Wisconsin* (CONDOR, 2012);
  - *Distributed Queueing System* (DQS) da Universidade da Flórida (DQS, 2012);
  - NQS (*Network Queueing Environment*) da Universidade de Maryland, fonte (NQS, 2005);
  - PBS (*Portable Batch System*) da Nasa – Lab. *Aimes* (PBS, 2012).

Segundo Dantas (2005), os pacotes RMS, disponíveis comercialmente ou com a abordagem de uso aberto, normalmente oferecem as seguintes facilidades numa configuração de computação distribuída:

- Suporte a sistemas operacionais heterogêneos;
- Ambiente de *batch*, paralelo e interativo;
- Verificação e migração de processo;
- Balanceamento de carga;
- Limite ao número de tarefas executadas em um determinado período;
- Interface gráfica amigável (GUI).

[4] O Univa Grid Engine previamente chamado de SGE (Sun Grid Engine) foi criado a partir do CODINE, atualmente é comercializado pela Oracle Corporation como Oracle Grid Engine e pela Univa Corporation com o nome de Univa Grid Engine. O SGE é normalmente usado em um aglomerado computacional ou computação de alto desempenho cluster (HPC) e é responsável por aceitar, agendar, despachar, e gerenciar a execução remota e distribuída de um grande número de postos de trabalho do usuário autônomo, paralelas ou interativo. Ele também gerencia os horários de alocação de recursos distribuídos, tais como processadores, memória, espaço em disco e licenças de software.

A **Tabela 2** mostra um sumário dos componentes da arquitetura de um sistema *RMS.*

Componente	Descrição Funcional
GUI	Interface através da qual o usuário solicita que suas tarefas sejam submetidas, monitoradas, excluídas ou colocadas em estado suspenso temporário.
Administração	Módulo que contém as características dos computadores pertencentes à configuração, acesso permitido para os usuários e suas tarefas, limitações de recursos para tarefas e usuários, controle de tempo de uso de recursos e funcionamento de filas.
Filas	O conceito de filas é empregado nos sistemas RMS, visando organizar a forma de execução de tarefas submetidas à configuração distribuída e prover uma abordagem eficiente de controle durante a execução das tarefas.
Computadores	São os elementos aonde serão executadas as tarefas, submetidas ao RMS. Como esses podem ter configurações heterogêneas em Termos de *hardware* e *software*, o módulo de Administração auxilia na filtragem dos serviços e recursos solicitados a cada computador.
Tarefas	Representam a unidade de solicitação de serviços e recursos distribuídos que são submetidos pelos pacotes RMS às configurações distribuídas.
Recursos	Processadores, quantidade de memória, sistemas de armazenamento e outros tipos especiais de dispositivos que podem ser oferecidos para que tarefas de usuários remotos possam utilizar.
Políticas	Geralmente as políticas que são consideradas pelo RMS são as de utilização e escalonamento. Quanto à política de utilização considera-se o módulo de Administração como uma base para sua implementação. Por outro lado, a política de escalonamento pode ser baseada na forma estática ou dinâmica.
Escalonamento-Balanceamento de Carga	O escalonamento pode ser baseado em algoritmos, tais como: o primeiro a chegar será o primeiro a ser servido, selecionar o ambiente com menor carga; utilizar uma seleção fixa baseada em algum conhecimento de administração do ambiente ou empregar um algoritmo híbrido considerando uma mistura dos algoritmos anteriores. Quanto ao balanceamento de carga, podemos utilizar formas que considerem: informações de pesos previamente estabelecidos ou séries históricas de carga do ambiente.

Tabela 2 - Sumário de elementos e funções dos sistemas RMS. (DANTAS, 2005) adaptado.

Os sistemas RMS funcionam normalmente integrados à camada de Recursos da arquitetura em camadas de um *Grid* Computacional (**Figura 5**) e são utilizados como interface entre o usuário que submete suas tarefas e a infraestrutura computacional disponível para a execução, recebendo, portanto, as solicitações da camada de aplicações de usuário e enviando-as para a camada Fábrica.

Segundo Krauter, Buyya e Maheswaran (2002), para um ambiente de *Grid* suportar eficientemente uma variedade de aplicações, o sistema de gestão de recursos (RMS), elemento central para o funcionamento de toda a infraestrutura do ambiente, deve abordar os desafios apontados nos trabalhos de FOSTER(1999) e ABRAMSON; GIDDY; KOTLER(2000), que levantam, entre outros pontos, as seguintes questões:

(a) adaptabilidade suficiente;

(b) extensibilidade e escalabilidade;

(c) permitir que os sistemas com diferentes políticas administrativas interoperem preservando a autonomia do site;

(d) coalocação de recursos;

(e) a qualidade do serviço de apoio e

(f) atender as limitações de custo computacional.

O RMS gerencia o conjunto de recursos que estão disponíveis para o *Grid*, ou seja, a programação de processadores, a largura de banda de rede e o armazenamento em disco.

Em uma infraestrutura de *Grid*, um *pool* pode incluir recursos de diferentes provedores, os quais requerem um RMS para manter a confiança de todos os provedores de recursos.

Manter o nível exigido de confiança não deve prejudicar a eficiência do RMS, aumentando-se a sobrecarga para as operações básicas. Devido às diferentes

políticas administrativas, à heterogeneidade de recursos e às proporções esperadas da *Grid*, pode ser necessário o emprego de uma federação de RMSs em vez de um único RMS.

Os RMSs da federação devem interoperar utilizando um conjunto acordado de protocolos para gerenciar os recursos, como ilustra o modelo da **Figura 9**.

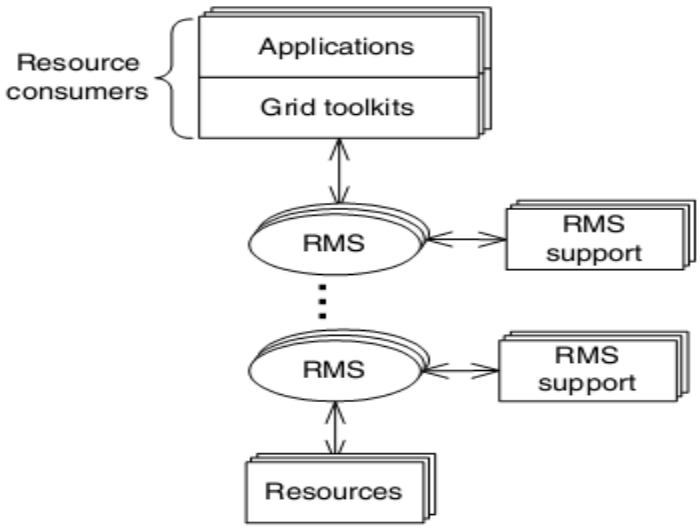


Figura 9 - RMS System Context. Fonte: (KRAUTER; BUYYA; MAHESWARAN, 2002).

As candidaturas podem, direta ou indiretamente, solicitar recursos ao *Grid*. Tais recursos solicitados são considerados como *Jobs* pela *Grid*. Dependendo da aplicação, o *job* pode especificar a qualidade de serviço (QoS) ou aceitar níveis de serviço de melhor esforço.

O RMS é necessário para executar decisões de gestão de recursos, maximizando as métricas de QoS entregues aos clientes quando os trabalhos tem restrições de QoS (MAHESEARAN, 1999).

Na prática, um RMS de *Grid* pode ser necessário para lidar com postos de trabalho diferentes, utilizando políticas diferentes. Por exemplo, alguns trabalhos podem

necessitar de suporte a QoS, enquanto outros podem exigir mais esforço de processamento, caso de *High Throughtput Computing* (HTC).

Em geral, exigir a RMS para apoiar várias políticas pode obrigar à programação de mecanismos para resolver problemas de otimização multicritérios.

2.3.1 Submissão de *Jobs*

De acordo com Wilkinson (2010), o propósito preliminar de um ambiente de computação distribuída, especialmente no caso dos *Clusters* e Grids Computacionais*,* é prover recursos de infraestrutura computacionais localmente ou geograficamente distribuídos para que os usuários possam executar seus *Jobs*.

No contexto de Grids Computacionais, o termo *job* é uma aplicação ou tarefa realizada em um ambiente de recursos de computadores de alto desempenho. Ele pode ser composto por seções individuais escalonáveis OGF (ROEHRIG; ZIEGLER; WEDER, 2012).

Uma tarefa (*task*) pode ser definida de maneira análoga ao *job* como sendo uma parte específica deste, e pode ser entendida *como* uma unidade de *job*.

No contexto da proposta desse trabalho, o *job* é formado pelo conjunto de tarefas de uma aplicação divididas em unidades escalonáveis enviadas ao sistema *RMS* que recebe a submissão do processo.

Um exemplo básico dos componentes de submissão de um *job* em *Grid* Computacional é mostrado na **Figura 11**. Os escalonadores atribuem trabalho (*job*) para calcular os recursos e atender às exigências de trabalho, especificadas dentro das limitações de recursos disponíveis e suas características.

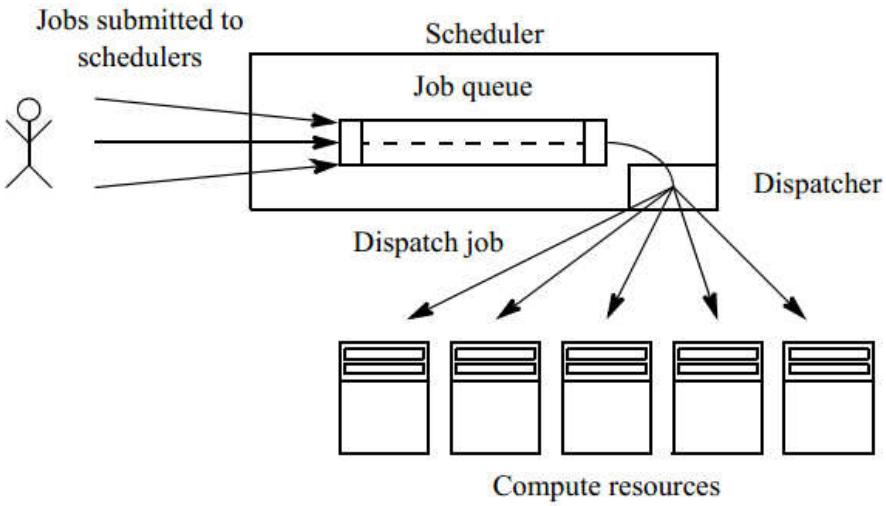


Figura 10 - Componentes básicos de submissão de um *Job*. Fonte: (WILKINSON, 2010).

Escalonamento é um problema de otimização. O objetivo básico da técnica é otimizar o desempenho dos *Jobs*. Cada um terá certamente características estáticas e dinâmicas e requisitos distintos de recursos. Assim, cada recurso computacional tem suas características estáticas e dinâmicas que irão afetar na maneira como os *Jobs* serão executados por eles.

Diferentes escalonadores podem ter comportamentos distintos quanto à maneira de alocar os *Jobs* aos recursos, mas geralmente eles entram em filas, como ilustra a **Figura 10**.

Os *Jobs* são colocados numa fila e enviados para um recurso computacional específico, baseado num algoritmo de escalonamento que leva em conta o *job* e suas características de recursos. Um componente chamado *Dispatcher* é responsável por tal operação.

Os escalonadores classificam-se quanto a:

- Política de escalonamento:
    - *First-in, first-out*: o primeiro *job* que entra é o primeiro a ser atendido;

- *Favor certain types*: política de favorecimento de *Jobs* de acordo com seu tipo, com base numa classificação de critério definido;
- *Shortest job first*: é prioritário o *job* menor;
- *Smallest* (*or largest*) *memory first*: é prioritário o *job* que ocupar maior (ou menor) quantidade de memória do ambiente;
- Short (*or long*) *running job first*: é priorizado nessa política o *job* que tem menor tempo de execução (*makespan*);
- *Priority based*: política baseada na atribuição de uma prioridade numérica para cada *job.*

Escalonadores de *Jobs* também podem incluir:

- Associação de recursos a *Jobs*;
- Escalonamento dinâmico baseado na carga;
- Escalonamento preemptivo com migração de processos entre outros.

Há uma particularidade relevante em ambientes heterogêneos como das Grids Computacionais a se destacar, que é a necessidade dos escalonadores de *Jobs* programarem um mecanismo de relatório de características dinâmicas dos recursos do ambiente computacional, pois nesses ambientes computacionais as características de recursos disponíveis mudam dinamicamente.

- Tipos de *Jobs*: os escalonadores esperam que o nome do *job* seja algo que já possa ser executado nos recursos de destino, possivelmente o nome de arquivos de entrada e saída em uma linha de comando. O *job* pode ser um *script* de sistema operacional ou também um comando a ser executado imediatamente. Eles podem conter também uma série de múltiplos executáveis ou até várias instâncias de executáveis com o mesmo nome. As

múltiplas instâncias de um mesmo executável com parâmetros diferentes são chamados também de *arrays* de *Jobs.* Essa solução é muito empregada para resolver aplicações científicas e chamadas de aplicações *parameter sweep* (item 2.2.1.1). A maioria dos *Jobs* é esperada pelos escalonadores como processos em *batch*. Um dos tipos mais esperados de *Jobs* são processos de longa duração. Padrões de resultados de entrada e saída são redirecionados para arquivos.

- Tipos de recursos computacionais: normalmente os recursos Computacionais consistem em um número de computadores individuais, algumas vezes centenas de computadores interligados formando um *cluster*. Esse ambiente é utilizado já há muitos anos e tem escalonadores para configurações em agrupamentos computacionais. Essas configurações recebem os *Jobs* por meio de uma interface de nó *front-end* e usualmente utilizam escalonadores para distribuir as tarefas aos recursos computacionais. São exemplos de escalonadores de *Jobs* Condor, SGE e LSF.

- Recursos computacionais escalonados: os recursos computacionais disponíveis para esse processo são os mais diversos e consideram algumas das seguintes características:

    - Características estáticas das máquinas: tipo de processador, número de cores, *threads,* velocidade, memória principal e memória cachê;

    - Características dinâmicas das máquinas: carga na máquina, disco avaliado para uso e carga de rede;

    - Preferências de usuários/requisitos de recursos computacionais;

    - Conexões de rede e características;

    - Características do *Job*: tamanho do código binário, dados, tempo de execução estimado, requisitos de memória, localização de arquivos de entrada e saída, estágios de entrada e saída de arquivos.

- *Job Matching*: esse recurso é o responsável por associar a quantidade de recursos computacionais ao *job* a ser executado. Algumas vezes um *job* será executado em múltiplas máquinas e em outras situações o recurso múltiplos *Jobs* será executado em uma máquina, ou ainda os recursos ociosos de algumas máquinas são utilizados para a execução do referido *job*.

A abordagem de submissão de *job* e de transferência de arquivos (*file staging*) da pesquisa tem por base o modelo do *middleware* de *Grid* Computacional Globus Toolkit (GT) devido à grande padronização desse ambiente e de sua grande aceitação na comunidade científica e organizações. Desse moco, os itens 2.3.1 e 2.3.2 são baseados nas técnicas de computação em *Grid* propostas por Wilkinson (2010).

2.3.1.1 Componentes de submissão de um job

1. Grid Resource Allocation Management (GRAM): este é o componente do Globus (Globus Toolkit) que recebe a submissão dos *Jobs*.

   Esse componente é um do projeto Globus, que produz tecnologias com as quais os usuários de um *Grid* que empregam o *middleware* Globus Tooolkit podem localizar, submeter, monitorar e cancelar a execução remota de um *job* que se baseia em recursos de um *Grid* Computacional.

   Ressalta-se que GRAM não é um escalonador de *Jobs,* apenas a camada de interface para agendamento de *Jobs* para os distintos escalonadores de um ambiente de *Cluster* ou *Grids*, oferecendo, pois, vários protocolos para comunicação com os escalonadores (*RMS*)

2. Escalonadores de *Jobs* (contidos em RMS): A **Figura 10** exibe os componentes básicos da submissão de um *job* em recursos de computação local.

Nessa figura, os *Jobs* são submetidos para um sistema de computação “front-end”, que, por sua vez, passa os *Jobs* para um *cluster* de nós de computação por

meio de um escalonador no modo de um *cluster* tradicional. O usuário envia sua submissão de *job* ao GRAM, o qual provê uma interface unificada para acessar os recursos do *Grid* que gerenciam os *Jobs.*

O GRAM é um serviço de *Grid*, instanciado como um *Grid* Service Container[5] disponível no *Globus Core*. Esse serviço pode fazer a submissão diretamente para local *host* (“fork”), mas ele tipicamente faz a submissão ao escalonador de *job* (*RMS*), e este encaminhará a execução ao *cluster* computacional, ou recurso computacional disponível no ambiente.

Os escalonadores de *Jobs* (Condor, PBS, LSF e etc.) têm interface com o GRAM por intermédio do componente GRAM *scheduler adapter.* Esses componentes estão disponíveis na versão 4 do Globus para Portable Batch System (PBS), *Loading Sharing Facility* (LSF) e Condor. Componentes *Adapter* de três camadas estão disponíveis para SunGrid Engine(SGE), IBM LoadLeveler e GridWay.

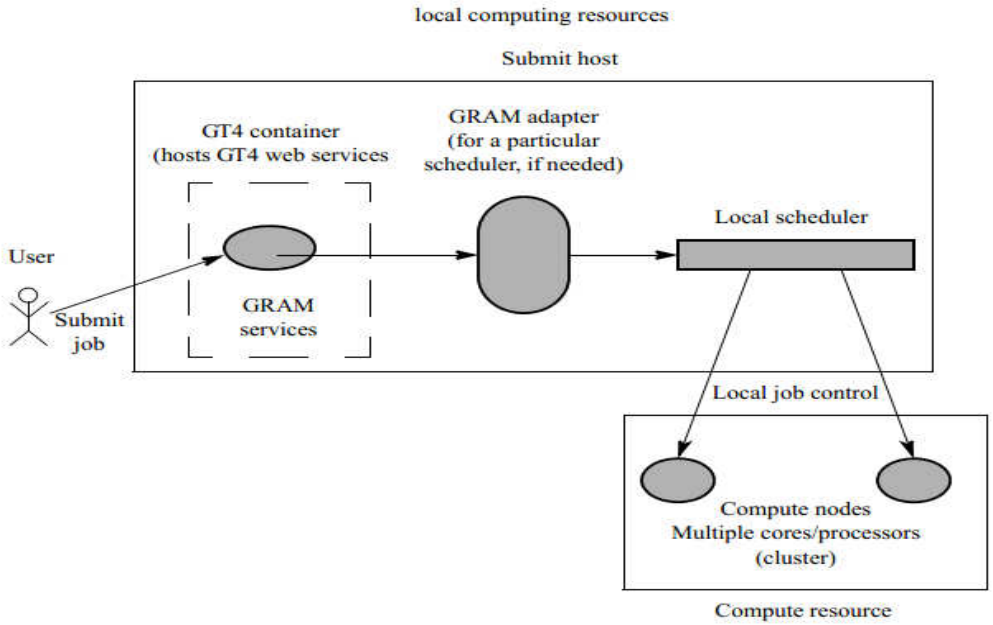


Figura 11 - Componentes básicos de submissão de *Job* em Grid – GRAM (WILKINSON, 2010).

[5] Grid Service Container: Serviço com especificação (OGSA), implementado no *middleware* Globus Toolkit, que enxerga os recursos de ambiente disponível num *Grid Computacional.*

2.3.1.2 Especificação de um *Job*

A submissão básica de um *job* é executada em Globus pelo ***globusrun-ws,*** que submete e monitora *Jobs* GRAM. Ele suporta múltiplas e simples submissões de *Jobs* e lida com gerenciamento de credenciais e *streming* de *Jobs stdout/stderr* durante a execução.

Há dois caminhos básicos para que um *job* seja especificado:

- Diretamente pelo nome do executável;
- Utilizando um arquivo de descrição de *job*.

Embora o *job* possa ser submetido diretamente por linha de comando, é muito mais flexível e poderoso fazê-lo, utilizando-se um arquivo de descrição de *job,* para descrever detalhes como: arquivos de entrada e saída, requisitos, memória necessária para execução.

Normalmente, um arquivo de descrição de *job* contém as seguintes informações:

- *Job Description File*
  - Nome do executável
  - Número de instâncias
  - Argumentos
  - Arquivos de Entrada
  - Arquivos de Saída
  - Diretórios
  - Variáveis de ambiente, paths etc.

- Requisitos de recursos
    - — Processador
        - Número de núcleos
        - Tipo
        - Velocidade
    - — Memory

2.3.1.3 Submetendo um *Job*

Para se realizar a submissão de um *job,* basicamente no G*lobus,* esta se efetuará pelo comando globusrun-ws, que fará a chamada ao escalonador responsável pela execução do *job* no ambiente.

Como visto no item 2.3.1, a submissão do *job*, além de poder ser feita em linha de comando com muitos dos parâmetros da tabela a seguir, pode também utilizar um arquivo de descrição de *job* (*job description file*) para configurar as características da submissão deste. Algumas das possibilidades de parâmetros para utilização com esse comando são:

- — *Output-mode*
- — Streaming
- — Batch Submissão
- — *Select Scheduler*: escolha do escalonador a utilizar, caso esteja disponível no ambiente -> Condor, LSF, SGE, LSF.

2.3.2 Transferência de Arquivos (File Staging)

Uma aplicação que submete um *job* a um sistema *RMS* – **Figura 10**, em muitos casos utiliza estágios de transferência de arquivo. Esta tarefa é definida neste item conforme o Workflow da **Figura 12**.

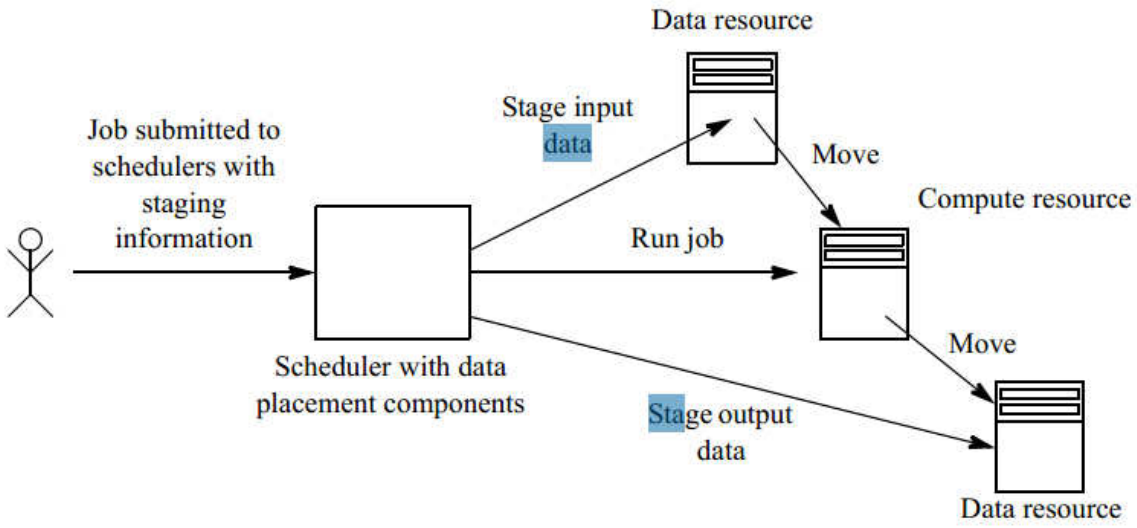


Figura 12 - Estágios de entrada e saída de arquivos. (WILKINSON, 2010).

Antes que um *job* possa ser executado em um recurso computacional (*Grids, Clusters*), há necessidade de que o executável e os arquivos de entrada possam ser acessados pelo ambiente computacional que o executará e também que o usuário que submeteu os arquivos tenha acesso aos que correspondem aos de saída do processo.

A maioria das plataformas de Grids Computacionais não utilizam unidades de armazenamento compartilhadas em rede, como *NFS* (Network File System) ou AFS, que geralmente não são disponíveis ao ambiente, principalmente por motivos de consumo de largura de banda. Isso ocorre devido ao fato desses ambientes serem plataformas geograficamente distribuídas.

Os arquivos são transferidos para o ambiente de execução basicamente de duas formas:

- Transferência por linha de comando: a movimentação de arquivos num ambiente de *Grid* Computacional requer de *Grid Data Transfer Services*. O Globus oferece muitos componentes de tratamento do gerencimanento de dados. O principal mecanismo de transferência de arquivos é em três

camadas e utiliza o controle de canais de FTP, mas opera sobre a *GSI (Grid Security Infraestructure),* que é um ambiente seguro.

A transferência de arquivos em três camadas pelo serviço GridFTP é ilustrado na próxima na **Figura 13**:

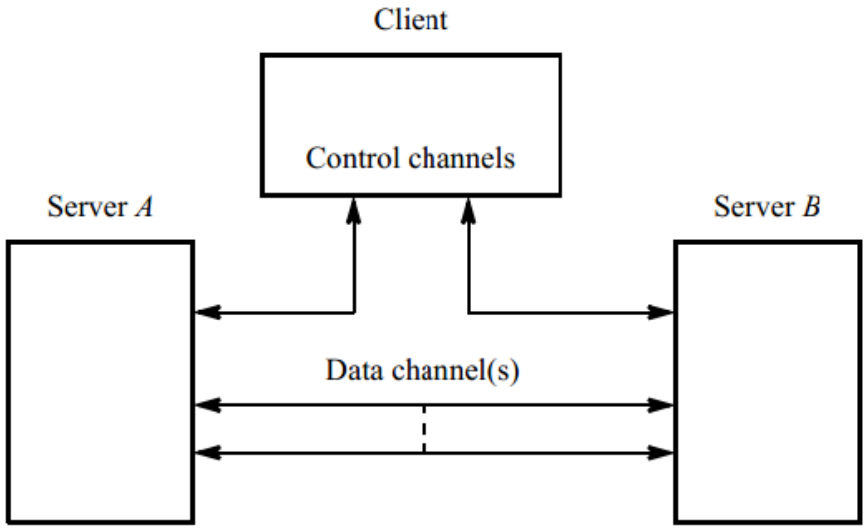


Figura 13 – GridFTP - Transferência em três camadas (WILKINSON,2010).

- *File Staging*: essa técnica refere-se a quando todos os arquivos são arranjados como coleções e movidos para onde eles são necessários. Muitos escalonadores contemplam esse mecanismo, tais como PBS, Condor, LSF, entre outros.

Muitas aplicações requerem etapas de *file staging* e podem ser alcançadas por mecanismos simples de *file transfer* via linha de comando, como na figura, mas com a técnica de *file staging* toda a sequência de movimentação de arquivos já pode fazer parte da especificação do *job,* como exibido na **Figura 13**.

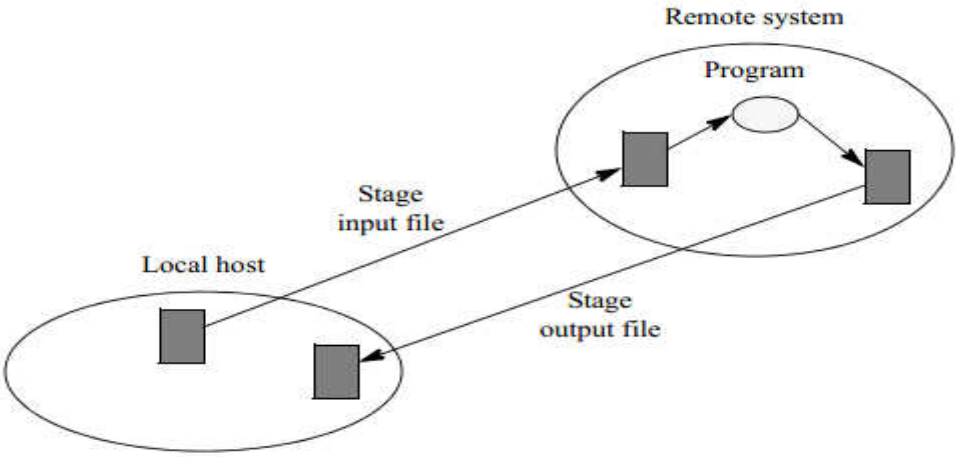


Figura 14 - *File Staging* (WILKINSON, 2010).

Os modelos de formatos de arquivos de descrição dos *file staging* são os seguintes:

- JDD: *Staging* pode ser especificada utilizando um arquivo de descrição JDD, que utiliza as tags <fileStageIn> e <fileStageOut>. Esses atributos têm subatributos <rftOptions>, <allOrNone>. O atributo (elemento) <transfer> especifica a origem e o destino como pares.

- JDSL: essa linguagem de descrição de *job* é bem similar à JDD e inclui a possibilidade de mecanismo de exclusão do arquivo após a transferência.

Seguem exemplos de uma JDD *description file transfer* e JDSL *description file transfer*:

1. JDD

```
<job>
    …
    <fileStageOut>
    <transfer>
     <sourceUrl>file:///prog1Out</sourceUrl>
     <destinationUrl>gsiftp://coit-grid05.uncc.edu:2811
                              /prog1Out</destinationUrl>
     </transfer>
</fileStageOut>
…
</job>
```

2. JDSL

```
<jsdl:DataStaging>
   <jsdl:FileName>/prog1Out</jsdl:FileName>
    <jsdl:CreationFlag>overwrite</jsdl:CreationFlag>
    <jsdl:Target>
        <jsdl:URI>gsiftp://coit-grid05.uncc.edu:2811/prog1Output</jsdl:URI>
   </jsdl:Target>
</jsdl:DataStaging>
```

2.4 Arquitetura Geral de Escalonamento em Grids Computacionais

Muitas aplicações estão se voltando para a computação em *Grid* com a finalidade de satisfazer as suas necessidades de armazenamento computacional e de dados.

Como *Sites* individuais simplesmente não são eficientes o suficiente para atender às necessidades de recursos das aplicações fim-a-fim[6], ao se utilizarem recursos distribuídos de alto desempenho, é possível dar a essas aplicações muitos benefícios. Computação em *Grid* eficaz é possível, no entanto, se os recursos forem bem escalonados.

Segundo Schopf (2002), o Escalonamento nas Grids Computacionais (*Grid Scheduling*) é definido como o processo de tomada de decisões que envolvem os recursos de escalonamento de múltiplos domínios administrativos.

Esse processo pode incluir uma busca em vários domínios administrativos para utilizar uma única máquina ou agendar um único trabalho para usar recursos em um único *site*, ou mesmo em vários deles.

A diferença entre Escalonador de *Grid* (*Grid Scheduler* ou Broker) e Escalonador de Recursos Locais (*Local Resource Scheduler*) é que o *Grid Scheduler* não é proprietário dos recursos locais, portanto não tem controle sobre eles.

[6] Aplicações fim-a-fim: este tipo de aplicação, muitas vezes denominada de P2P(fim-a-fim em português e confundida muitas vezes com ponto-a-ponto), permite que uma aplicação residente no computador do usuário possa submeter uma tarefa para um outro computador (ou até conjunto de computadores).

O escalonamento de tarefas envolve três fases (SCHOPF, 2002): fase 1 – descobrimento dos recursos, fase 2 – seleção do sistema e fase 3 – execução do *Job,* conforme a **Figura 15**.

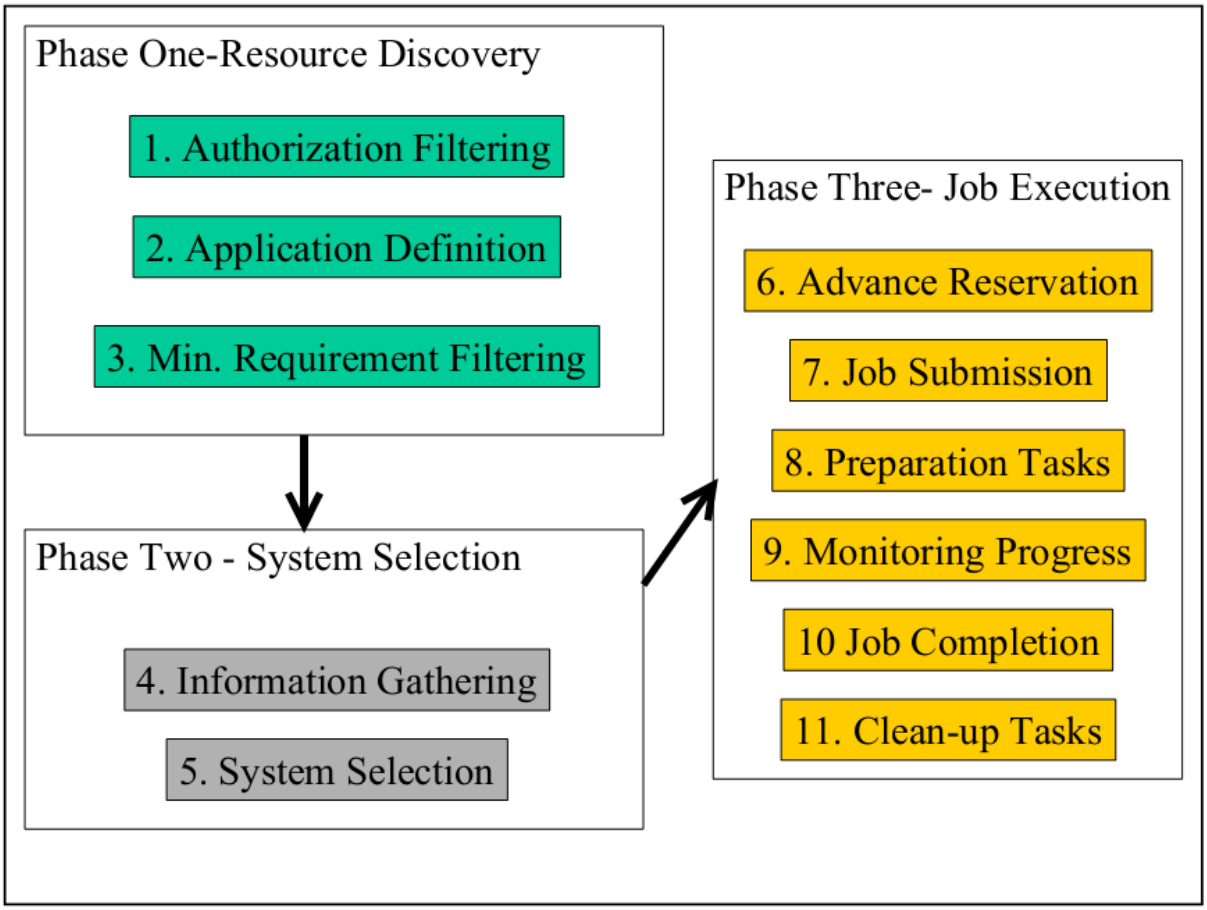


Figura 15 - Three-fase Archicteture for Grid Scheduling (SCHOPF, 2002)

Fase 1 - Descoberta dos Recursos (*Resource Discovery*): consiste na descoberta de quais são os recursos que estão disponíveis para um determinado usuário, sendo subdivida em três passos:

> Passo 1 - Autorização, que verifica a qual recursos o usuário tem acesso para a execução da aplicação (conjunto de tarefas);

Passo 2 - Definição de Requerimentos da Aplicação, que investiga quais são os requerimentos para a submissão da aplicação – quantidade de memória, sistema operacional, dentre outros;

Passo 3 - Filtragem dos Requerimentos, que retorna quais os recursos a que o usuário tem acesso e quais deles atendem aos requisitos da aplicação.

Fase 2 - Seleção do Sistema (*System Selection*): seleciona qual recurso (ou conjunto de recursos) será destinado para aplicação, e envolve dois passos (4 e 5):

Passo 4 - Reunião das Informações Dinâmicas (*Information Gathering*), que, a partir das informações disponíveis, procura detalhes sobre as informações dinâmicas dos recursos e os modos como os usuários podem acessá-los;

Passo 5 – Seleção do Sistema (*System Selection*), que escolhe os recursos para a aplicação com base nos dados coletados no passo 4;

Fase 3 - Execução da Aplicação (*Job Execution*), que é dividida em cinco passos (dos 6 ao 11):

Passo 6 – (passo opcional): é responsável pela reserva avançada dos recursos. Para fazer o melhor uso de determinados ambientes, é possível reservar parte do conjunto de recursos de maneira avançada (sensível às características do ambiente). Esse passo pode ser posto em prática, entre outros projetos, nos seguintes exemplos: *Globus Project by Roy* (*GARA*) e PBS (PBS, 2005);

Passo 7 – Submissão da Aplicação (*Job Submission*): é feita partindo da seleção dos recursos realizada anteriormente;

Passo 8 – Preparação das tarefas (*Preparation Tasks*): contém as ações necessárias para executar a aplicação;

Passo 9 – Monitoramento do Progresso (*Monitoring Progress*): passo necessário a fim de que o usuário possa saber o andamento da execução ou reescalonar (retornar para o passo 4 caso não se verifique progresso significativo);

Passo 10 – Finalização do Trabalho (*Job Completion*): acionado em tempo de finalização da aplicação e sobre o qual o usuário precisa ser notificado;

Passo 11 – Limpeza das Tarefas (*Clean-up Tasks*): é o último passo da proposta de planejamento e está relacionado à Limpeza de Tarefas, ou seja, a transferência dos resultados para o usuário, remoção do ambiente temporário (variáveis de ambiente, arquivos), dentre outras atividades referentes aos resultados do processo como um todo.

Capítulo 3: Proposta

Este capítulo apresenta a descrição da proposta de planejamento de execução de aplicações em Computação Distribuída, especialmente para o escopo dos ambientes de Grids Computacionais.

Nesta seção do trabalho, será exposta a metodologia de pesquisa em sete etapas, ordenadas sequencialmente e tem o objetivo de dar uma contribuição ao processo de tomada de decisão quanto à utilização de ambientes de computação distribuída de alto desempenho para a execução da aplicação em estudo.

As etapas do método são as seguintes:

- Estudo de possibilidade de utilização da aplicação em computação distribuída: o foco dessa etapa é investigar, com base na classe da aplicação (item 2.2.1), se é possível realizar o processamento desta em ambientes de computação distribuída;

- Estudo das métricas de avaliação: nessa etapa, a ideia básica é definir quais são as métricas de avaliação que se pretende extrair da proposta;

- Divisão da aplicação em tarefas menores a ser executadas: nessa etapa, é realizado o fracionamento, de fato, da aplicação em partes menores, com o objetivo de distribuí-las nos nós computacionais disponíveis na infraestrutura de experimento;

- Definição do método de transferência dos arquivos de entrada e de saída: nessa etapa descreve-se como deve ser realizada a transferência dos arquivos de entrada, bem como a forma de recebimento dos arquivos de resultados;

- Montagem dos lotes de execução das tarefas: faz-se o agrupamento das tarefas em lotes de execução de acordo com as características da aplicação e o *RMS* em que está sendo utilizado no ambiente (WILKINSON, 2010);

- Execução das tarefas e recebimentos dos resultados: etapa em que o usuário envia ao sistema *RMS* a sua aplicação para execução, aguardando os resultados do processo.
- Análise dos resultados: essa etapa é responsável pela transformação dos dados gerados pelos diversos tipos de arquivos de *log* (resultados) recebidos pelo ambiente de *RMS*, sendo utilizado na aplicação do método, criando-se planilhas e gráficos de visualização de acordo com as métricas definidas no segundo item da metodologia.

3.1 Estudo de possibilidade de utilização da aplicação em computação distribuída

Nessa etapa, tem-se o objetivo de realizar um estudo prévio das características da aplicação em questão, buscando, com base no item 2.2.1 desse trabalho, verificar a possibilidade de divisão da aplicação em partes menores para serem executadas pelo sistema *RMS* do ambiente, responsável pela alocação das tarefas no ambiente de computação distribuída.

A verificação da divisão da aplicação em tarefas paralelizáveis é baseada na classificação de tarefas discutidas no capítulo 2, itens 2.2.1, 2.2.2 e 2.2.3.

O resultado esperado dessa etapa será uma resposta positiva ou negativa para a seguinte pergunta: “A aplicação em questão é candidata à utilização em ambientes de computação distribuída, especialmente em Grids Computacionais?”.

3.2 Estudo das métricas de avaliação

O estudo das métricas é obtido com base em dois principais objetivos:

- Classificação da aplicação a utilizar no ambiente (item 2.2.1);
- Estudo da taxonomia de Krauter, Buyya e Maheswaran (2002) do item 1.4.3 e Figura 6.

A decisão das métricas pretendidas com a aplicação dessa proposta pode tomar por base os dois itens acima e, como exemplo pode-se considerar uma aplicação

classificada como *Bag-of-Tasks* (item 2.2.1). Esse tipo de aplicação pode ser dividido em várias tarefas menores independentes e sem comunicação entre elas. Aplicando-se a técnica de estudo proposta neste trabalho, é concretizada a aplicação das etapas 3.1.2 e 3.1.3 e, para o caso em estudo, verifica-se que é um tipo de aplicação viável.

Além da possibilidade de fracionamento e comunicação, o estudo da taxonomia também é aplicado para que seja decidido, por exemplo, se é esperado um ganho de tempo na soma dos tempos totais das tarefas, além de compará-lo com o tempo de execução da aplicação local (índice *makespan* que mede o tempo entre o início da execução da tarefa e a entrega da resposta pelo sistema de interface), ou deseja-se aplicar as duas métricas de CIRNE et al (2002) que propõem considerar o tempo de carga dos arquivos de entrada para os nós computacionais, para depois subtraí-los do tempo total de *makespan*.

3.3 Divisão da aplicação em tarefas menores a ser executadas (WILKINSON, 2010)

Nesta etapa acontece a divisão da aplicação em partes menores, seguindo o método de preparação das tarefas proposto pelo estudo de escalonamento geral de tarefas em ambientes de Grids Computacionais de Schopf (2002).

Com base na classificação do item 2.2.1, a aplicação em questão poderá ser dividida em *parameter sweep*, *bag-of-tasks* ou *workflow*. É nessa fase que as tarefas que compõem os *Jobs* (figura 8*)* são quebradas, para posterior agrupamento em unidades de execução da fase 3 do modelo de Schopf (2002) da Figura 15 do item 2.4.

3.4 Definição do método de transferência dos arquivos de entrada e saída

Esta etapa, também conhecida como *File Staging* no problema de escalonamento (seção 2.3.2), faz o planejamento dos dados de entrada e também como as saídas devem ser recebidas no ambiente computacional em utilização. O conjunto de arquivos planejados nesse contexto, para transferência, são requisitos para que a tarefa seja executada no ambiente computacional disponível.

A estratégia de transferência dos arquivos é definida no item 2.3.2. Dependendo do escalonador em uso, a política de transferência de arquivos de entrada necessários pode variar nos comandos necessários para a sua implementação.

Muitos escalonadores de *Grid (RMS)* suportam a utilização de serviços como GridFTP para permitir que os dados necessários sejam transferidos para o *site* onde a tarefa será executada. Nesse sentido, o pacote *RMS* Condor, utilizado nos experimentos dessa pesquisa, suporta vários tipos desses estágios de transferência (*file staging*) de arquivos, quer sejam antes, quer seja depois da aplicação executar.

O resultado dessa etapa é uma lista de unidades de rede que receberão os arquivos e os comandos de transferência. Para iniciar a próxima etapa, haverá necessidade do escalonador em uso receber os dados no passo número 8, que faz parte da terceira fase do modelo de escalonamento de Schopf (2002), figura 15.

3.5 Montagem dos lotes de tarefas para execução

Nesta etapa, com base nas políticas de escalonamento adequadas ao problema da aplicação em estudo e a aplicação da técnica de divisão das tarefas (item 3.1.3), as já “quebradas” tarefas devem ser agrupadas para execução. Seguindo a estratégia do item 2.3.1, há necessidade de delimitar a compatibilidade da proposta de agrupamento desse item com o gerenciamento de recursos (*RMS*) que está sendo utilizado.

Essa etapa da montagem dos lotes faz parte da fase 3 do modelo de Schopf(2002), figura 15. Como resultado dessa fase, espera-se o agrupamento de tarefas em lotes para a submissão, ou seja, a dos *Jobs* definidos no item 2.3.1.

3.6 Processamento das tarefas e recebimento dos resultados (*logs*)

Após a realização da melhor política de agrupamento das tarefas que irão compor os *Jobs*, nessa etapa, eles são submetidos ao sistema de interface (*user* da figura 8) com a camada dos recursos computacionais *RMS.*

Para tanto, basta fazer o agendamento das execuções dessas unidades de processamento, monitorar o processo de execução e aguardar os dados recebidos de resultados a serem compilados e estudados na próxima fase.

Essa etapa corresponde aos passos 7, 9 e 10 do modelo de escalonamento de Grid de Schopf (2002).

3.7 Análise dos resultados

Esta etapa é responsável pela transformação dos dados gerados pelos diversos tipos de arquivos de *log* (resultados) recebidos pelo ambiente de *RMS,* que estão sendo utilizados na aplicação do método, criando-se planilhas e gráficos de visualização de acordo com as métricas definidas no primeiro item dessa metodologia.

É nessa fase da metodologia proposta nesse trabalho que as tendências e conclusões do estudo de viabilidade apresentam sua principal contribuição.

Capítulo 4: O Caso de Estudo

Este capítulo apresenta, de maneira sucinta, um estudo de aplicação de técnicas de Mineração de Textos (MT) e Inteligência Artificial (IA) para a classificação das ementas da jurisprudência da Justiça do Trabalho de São Paulo proposta por Ferauche (2011). O trabalho aplica técnicas de MT e IA para auxiliar no processo de classificação de ementas.

Esse processo é realizado em três etapas básicas: a Fase de Extração das Ementas, na qual são capturados os conteúdos dos textos que serão minerados; o Pré-processamento das Ementas, em que os textos são transformados em valores que demonstram as características dos textos; e o Processamento das Ementas, etapa na qual as características descobertas na fase anterior são utilizadas para o aprendizado dos classificadores.

Essas fases são sequenciais e formam um método para realizar a classificação das ementas de maneira sistemática, apoiada pelo computador.

A presente dissertação faz a proposta de estudo de viabilidade de utilização de uma aplicação em ambientes de Grids Computacionais, com base no estudo do tipo de tarefas a aplicação.

Nesse sentido, a metodologia proposta no capítulo 4, constitui o estudo de viabilidade da fase de Pré-processamento das Ementas de Ferauche (2011), contendo algumas adaptações com foco na execução em ambientes de Computação Distribuída.

4.1 Justificativa da Escolha do Pré-processamento das Ementas

Iniciando o estudo de viabilidade com base na aplicação do método proposto no capítulo 3, possíveis é possível que sejam verificados na análise três fatores que apontam a fase de pré-processamento de ementas como uma das melhores candidatas ao pré-processamento em Grids Computacionais com a abordagem de execução de tarefas com sistemas *RMS*.

Segundo Wilkinson (2010), no planejamento de execução de uma tarefa em computação distribuída, há necessidade de que seja feito o estudo da aplicação, com o objetivo de verificar se esta pode ser dividida em tarefas menores para serem executadas em paralelo por esse ambiente.

O estudo sucinto da fase de “Pré-processamento das Ementas” passo a passo remete às seguintes justificativas de escolha para a aplicação do método da proposta:

- Passo de análise de viabilidade (seção 3.1.1): a tarefa de pré-processamento de ementas pode ser dividida facilmente em tarefas menores de processamento pelo próprio modelo de processamento desse capítulo, o que torna o resultado dessa análise de viabilidade afirmativo;

- Passo de divisão da aplicação em tarefas menores de execução (seção 3.1.3): como o estudo tem o objetivo de classificação de ementas, com base em aprendizado de classificadores binários, cada tarefa da aplicação é o pré-processamento de textos de uma categoria, e isso faz com que a unidade de tarefa seja o pré-processamento de uma categoria;

- Passo de definição dos métodos de transferência dos arquivos de entrada e saída (seção 3.1.4): considerando que foi possível selecionar a forma de transferência dos arquivos de entrada e saída para as coleções de textos a serem pré-processadas nos experimentos do capítulo 4.4, esse item é também atendido pela prática experimental;

- Passo de Montagem dos lotes de tarefas de execução: na seção 5.5 é feito o agrupamento das coleções em lotes de 5, 10 e 20 tarefas para serem pré-processadas numa única unidade de execução enviadas ao ambiente.

No período de pesquisa desse estudo, os pesquisadores desenvolveram trabalhos de classificação das ementas da Justiça do Trabalho de São Paulo, os quais resultaram na produção de artigos publicados em congressos de pesquisa, como, por exemplo, o Congresso Ibero-Americano e do Workshop de Pesquisa do Centro Paula Souza (CETEEPS).

Nesse sentido, é possível citar o trabalho de Aprendizado de Classificadores (FERAUCHE; ALMEIDA; ITO, 2011), Uma Proposta de Workflow para Pré-processamento de Documentos em Grid Computacional (BORGES; FERAUCHE; ALMEIDA, 2010), Categorizador de Documentos (FERAUCHE; ALMEIDA; ITO, 2010) e Pré-processamento Distribuído de Documentos para Algoritmo de Aprendizagem de Máquina (BORGES; ALMEIDA, 2011).

A proposta inicial desse trabalho seria buscar, com base em referências teóricas e também em estudo de caso, uma proposta de metodologia para estudar se uma determinada aplicação é uma boa candidata à execução em Grids Computacionais.

Os resultados iniciais das pesquisas desenvolvidas nos trabalhos citados permitiram a seleção de um bom caso de estudo.

Assim, a escolha dessa fase de Pré-processamento das Ementas justifica-se pelos resultados dos estudos citados e também pela grande aderência que esse processo tem com a metodologia de sete etapas, proposta na presente Dissertação de Mestrado.

4.2 Classificação de Ementas

O trabalho descrito nesta seção é uma proposta de utilização da abordagem de aprendizado de máquina, devido à alta complexidade de extrair o conhecimento dos especialistas para expressá-las dentro do código.

Desse modo, foi utilizada uma coleção de ementas classificadas pelos especialistas de Direito como exemplos pré-classificados para o treinamento dos algoritmos classificadores e outra coleção de ementas para a validação do aprendizado dos algoritmos classificadores.

Nesse sentido, a Mineração de Textos (MT) tem por objetivo descobrir informações relevantes por meio de dados não estruturados, contidos em formato texto. Assim, uma definição genérica inclui todos os tipos de processamento de texto que tratam de encontrar, organizar e analisar informação (KONCHADY, 2006).

Portanto, a aplicação de técnicas de MT e IA, quando utilizadas em conjunto, permitem a constituição de um mecanismo de auxílio à tarefa de classificação das ementas trabalhistas.

A classificação em Mineração de Textos visa identificar os tópicos principais de um documento e depois associá-lo a uma ou mais categorias predefinidas (EBECKEN; LOPES; COSTA, 2003).

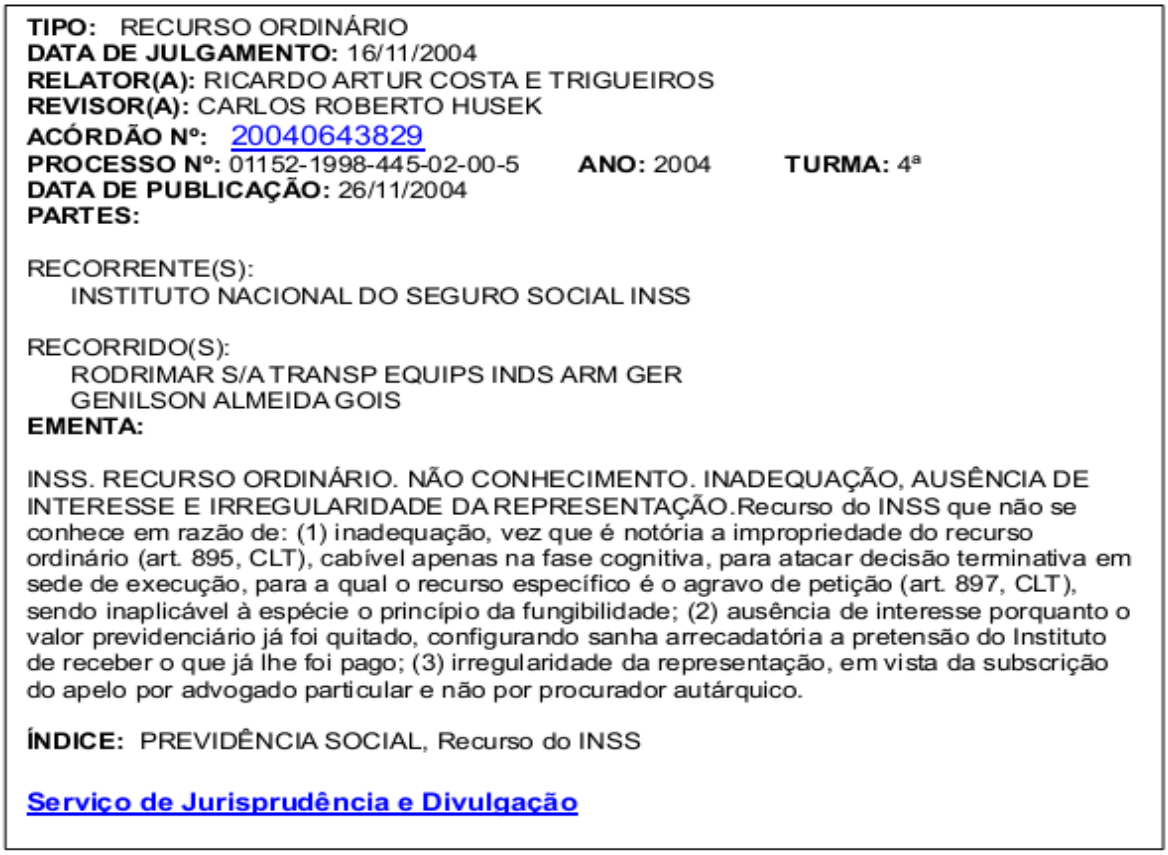
De acordo com Konchady (2006), o problema da classificação pode ser descrito como a separação de documentos em múltiplas categorias, em que se tem um conjunto de n categorias {C1, C2, . . Cn} para as quais são associados m documentos {D1, D2, . . Dm}.

Quanto à ementa, trata-se de um resumo de uma decisão (acórdão) tomada por um colegiado de desembargadores. Com a finalidade de facilitar a pesquisa jurisprudencial do Tribunal Regional do Trabalho da 2ª Região – São Paulo, um especialista em Direito do Trabalho realiza a tarefa de classificá-las seguindo a ontologia mantida pela Secretaria de Gestão da Informação Institucional, mas de maneira empírica e fortemente dependente do nível de conhecimento e experiência do especialista.

O grande número de ementas a serem classificadas, sobre os mais variados assuntos, faz com que o procedimento adotado seja bastante complexo, sobretudo sem auxílio computacional que ajude nessa tarefa.

A Jurisprudência, no domínio do Direito, desempenha um importante papel como fonte dessa ciência e o conteúdo dela auxilia a interpretação da lei e sua aplicação na solução de um problema no âmbito jurídico. Os documentos dessa prática são

gerados em formato de hipertexto, com base em informações do banco de dados, conforme o modelo da figura seguinte.



TIPO: RECURSO ORDINÁRIO
DATA DE JULGAMENTO: 16/11/2004
RELATOR(A): RICARDO ARTUR COSTA E TRIGUEIROS
REVISOR(A): CARLOS ROBERTO HUSEK
ACÓRDÃO Nº: 20040643829
PROCESSO Nº: 01152-1998-445-02-00-5 **ANO:** 2004 **TURMA:** 4ª
DATA DE PUBLICAÇÃO: 26/11/2004
PARTES:

RECORRENTE(S):
INSTITUTO NACIONAL DO SEGURO SOCIAL INSS

RECORRIDO(S):
RODRIMAR S/A TRANSP EQUIPS INDS ARM GER
GENILSON ALMEIDA GOIS
EMENTA:

INSS. RECURSO ORDINÁRIO. NÃO CONHECIMENTO. INADEQUAÇÃO, AUSÊNCIA DE INTERESSE E IRREGULARIDADE DA REPRESENTAÇÃO.Recurso do INSS que não se conhece em razão de: (1) inadequação, vez que é notória a impropriedade do recurso ordinário (art. 895, CLT), cabível apenas na fase cognitiva, para atacar decisão terminativa em sede de execução, para a qual o recurso específico é o agravo de petição (art. 897, CLT), sendo inaplicável à espécie o princípio da fungibilidade; (2) ausência de interesse porquanto o valor previdenciário já foi quitado, configurando sanha arrecadatória a pretensão do Instituto de receber o que já lhe foi pago; (3) irregularidade da representação, em vista da subscrição do apelo por advogado particular e não por procurador autárquico.

ÍNDICE: PREVIDÊNCIA SOCIAL, Recurso do INSS

Serviço de Jurisprudência e Divulgação

Figura 16 - Jurisprudência retirada do site do Tribunal Regional do Trabalho da 2ª Região.

O documento em análise segue uma estrutura na qual é possível identificar uma espécie de cabeçalho, com informações que identificam a origem da jurisprudência, tais como: Tipo do Processo, Data de Julgamento, Juiz Relator e Revisor do acórdão, Número do acórdão, Ano do acórdão, Turma do acórdão, Data de publicação, Número do processo e Partes envolvidas.

É possível, ainda, identificarem-se, nessa estrutura, mais duas partes: a ementa e o índice.

A ementa é a parte em que se encontra a síntese do que foi decidido no acórdão, premissas dele e justificativas; nela está concentrado todo o conhecimento da jurisprudência, e ela é digitada por servidores públicos das Secretarias das Turmas.

Por último, existe o índice, que é a classificação da jurisprudência, utilizado para organizar e facilitar a busca desta, preenchido pelos servidores públicos do Serviço de Jurisprudência e Divulgação, que podem ser identificados como os especialistas do sistema, pois são eles que leem a ementa, identificam relações na área do Direito e depois classificam a jurisprudência (FERAUCHE; ALMEIDA; ITO, 2010).

Para fins de melhor compreensão do experimento realizado nessa pesquisa, será apresentada a seguir a metodologia do caso de estudo de Ferauche (2011).

A proposta, objeto deste caso em estudo, concentra-se na tarefa de classificação de documentos. Para tal objetivo, o trabalho é dividido em três fases:

- Fase de Extração das ementas: faz a captura dos textos a serem minerados;
- Fase de Pré-processamento das ementas: no qual os textos são transformados em valores que demonstram as características dos textos;
- Fase de Processamento das ementas: em que as características são utilizadas para o aprendizado dos classificadores.

4.2.1 Extração das Ementas

Trata-se da fase em que se extrai o conteúdo das ementas a partir de arquivos de texto, que por padrão são gerados para envio às Editoras que compõem a revista jurisprudencial. Esses arquivos têm a extensão “.JUR” e correspondem às ementas dos acórdãos publicados de janeiro de 2008 a janeiro de 2011.

Um programa aplicativo foi desenvolvido para ler os arquivos citados anteriormente e extrair apenas o conteúdo da ementa. O texto entre os delimitadores “..EMEN:” e “..DECI:” forma o conteúdo do resultado da decisão.

Há, no conteúdo do arquivo, um texto entre o delimitador “..INDE:”, que indica a categoria da jurisprudência a que pertence a ementa, criando o aplicativo uma estrutura de diretórios, além de copiar os arquivos de cada categoria em seu respectivo diretório.

Portanto, são geradas coleções de documentos, cada coleção de uma determinada categoria com tamanho em *bytes* (quantidade de informação) diferentes. Tal organização permite entender as particularidades de cada categoria e o conhecimento dessas características da coleção forma uma base para a tomada de decisão das etapas subsequentes.

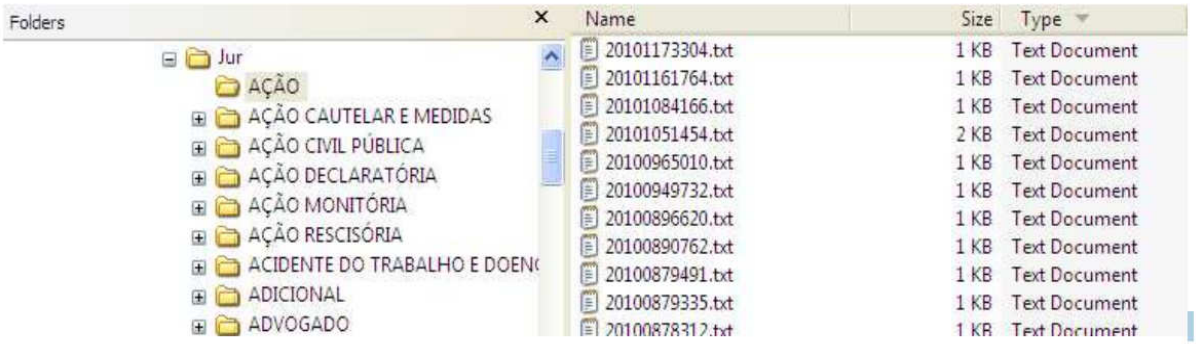


Figura 17 - Estrutura de diretórios das categorias e suas ementas (FERAUCHE; ALMEIDA, 2011).

O resultado dessa fase é uma coleção de documentos textuais, sem as informações de cabeçalho, que são irrelevantes. Assim, os documentos contêm apenas o conteúdo puro do texto da ementa, organizados em diretórios, sendo que cada diretório é uma categoria da jurisprudência trabalhista da 2ª Região – São Paulo.

4.2.2 Pré-processamento das Ementas

Esta é a fase em que ocorre a preparação dos documentos e extração de um conjunto de características destes, chamado de vetor atributo-valor, na qual cada termo é um atributo do vetor, com um valor para cada atributo. Essa parte do processo pode ser dividida em valorização dos atributos e seleção de exemplos do treinamento.

A primeira ação do pré-processamento das ementas foi definir qual a técnica de valorização dos atributos a ser utilizada, e a opção pela utilização ou não de critérios de suavização e normalização. Sendo assim, houve a necessidade de se verificar a distribuição da quantidade de documentos dentro de cada categoria e a quantidade de informações.

As categorias das ementas apresentaram uma distribuição irregular quanto à relação entre a quantidade de documentos e o tamanho de *bytes* de cada categoria, ou seja, existem categorias com menos documentos, mas com mais informações, assim como existem categorias com menos informação, porém com mais documentos, como visualizado na **Tabela 3**.

CATEGORIA	QUANTIDADE DE DOCUMENTOS	TAMANHO (Bytes)
PREVIDENCIA SOCIAL	12865	10.250.190
EXECUÇÃO	5370	4.191.805
MÃO-DE-OBRA	4308	4.743.027
EMBARGOS DECLARATÓRIOS	4248	2.561.049
PROVA	3689	2.867.747
RELAÇÃO DE EMPREGO	2922	2.583.514
PRESCRIÇÃO	2834	2.728.237
DANO MORAL E MATERIAL	2532	2.529.515
COMPETÊNCIA	2151	2.397.916
SINDICATO OU FEDERAÇÃO	2094	2.079.449

Tabela 3 - Exemplo de 10 categorias e a distribuição de seus documentos e seu tamanho em bytes (FERAUCHE; ALMEIDA, 2011)

Para a extração dos termos e atribuição de valor a eles foi utilizada a ferramenta PRETEXT II (MONARD;PRATI; SOARES, 2008), tendo em vista que ela utiliza a técnica de *bag of words* e faz uso de cortes de palavras baseados em frequência, utilizando a lei de Zipf (SOARES, 2009) e os cortes de Luhn (SOARES, 2009) para

restringir o problema da alta dimensionalidade do vetor atributo valor, que geralmente ocorre na Mineração de Textos (MT).

Como métricas, foram utilizados o Frequency - Inverse Document Frequency (tf-idf) e critérios de suavização quadrática por atributo (coluna), com o objetivo de reduzir o problema da irregularidade da distribuição da quantidade de documentos e de informação nas categorias, capturando-se o máximo das características relevantes nos documentos.

A ferramenta PRETEXT II é do tipo modular de pré-processamento de textos, desenvolvida em linguagem PERL pelo LABIC-ICMC na USP, disponível para Linux e Windows, de configuração flexível o suficiente para a transformação de coleção de textos para tabelas atributo x valor, que podem ser utilizados pelos algoritmos de aprendizado de máquina, pois esses algoritmos necessitam geralmente que os dados estejam representados de uma maneira estruturada.

Quanto à transformação de textos não estruturados em dados estruturados, esse procedimento requer o pré-processamento dos textos. Essa tarefa não tem a necessidade de interface com o usuário, tornando-se uma boa candidata ao processamento em lotes, normalmente utilizados como a divisão mínima da tarefa para execução em ambientes de computação distribuída, como é o caso dos Grids Computacionais.

A teoria do aprendizado computacional, conhecida como PAC-learning, criada por Leslie Valient, em 1984, mostra a importância da complexidade do relacionamento entre aprendizado computacional e a complexidade dos exemplos utilizados no conjunto de treinamento.

Essa perspectiva teórica em síntese, leva em consideração a distribuição dos exemplos positivos e negativos dentro do conjunto de treinamento, para o caso de uma predição booleana, de modo que a quantidade de exemplos deve ser restrita e distribuída de maneira proporcional, sem haver uma diferença grande entre exemplos positivos e negativos, senão a complexidade dos exemplos fará com que o

algoritmo não seja capaz de aprender, daí a importância de restringir o espaço de exemplos (RUSSEL; NORVIG, 2004).

Com o objetivo de restringir o espaço de exemplos para esta pesquisa, das 187 categorias, foram selecionadas 10 categorias que são constituídas de, no mínimo, 500 documentos, distribuídos da seguinte maneira: uma categoria que contém até 1000 documentos, duas categorias que contêm entre 1000 e 2000 documentos, duas categorias que têm entre 2000 e 3000 documentos, uma categoria que tem entre 3000 e 4000 documentos, duas categorias que têm entre 4000 a 5000 documentos e duas categorias que têm acima de 5000 documentos.

Não foram utilizadas todas as categorias nem selecionados todos os documentos das categorias escolhidas, pois não houve poder computacional disponível para tanto. Desse modo, pelo mesmo motivo, foi decidido construir classificadores binários para cada uma das 10 categorias.

Nesse sentido, foram selecionados aleatoriamente 500 documentos de uma categoria, confrontados com mais 500 documentos de 5 das 177 categorias restantes, selecionadas também aleatoriamente, respeitando-se a distribuição proporcional da quantidade real de documentos das 187 categorias, tendo por base a teoria PAC-learning [6], compondo-se um conjunto de exemplos para treinamento que contenham uma distribuição de exemplos positivos (da categoria que se pretende aprender) e de exemplos negativos (das outras categorias diversas), conforme pode ser visto na Tabela 4.

Como consequência, foram gerados os vetores atributo-valor de cada categoria a ser aprendida, em conjunto com outras categorias selecionadas de maneira aleatória.

Faz-se necessário fazer a tradução do formato próprio de tabela atributo-valor do PRE-TEXT II, para servir de entrada para a fase seguinte, a qual espera o formato ARFF (Attribute-Relation File Format).

Categoria	Real[7]	Selecionado[8]	Outras	Real	Selecionado
EXECUÇÃO	5370	500	EMBARGOS DECLARATÓRIOS	4248	181
			RELAÇÃO DE EMPREGO	2922	125
			SINDICATO OU FEDERAÇÃO	2094	89
			MANDADO DE SEGURANÇA	1612	69
			RESPONSABILIDADE SOLIDÁRIA/SUBSIDIÁRIA	895	39
			Total de Outros	**11771**	**503**
MÃO-DE-OBRA	4308	500	PROVA	3689	212
			SINDICATO OU FEDERAÇÃO	2094	121
			RECURSO	1297	75
			RESPONSABILIDADE SOLIDÁRIA/SUBSIDIÁRIA	895	52
			PROCESSO	726	42
			Total de Outros	**8701**	**502**
EMBARGOS DECLARATÓRIOS	4248	500	PRESCRIÇÃO	2834	171
			SINDICATO OU FEDERAÇÃO	2094	126
			CONCILIAÇÃO	1377	83
			HORAS EXTRAS	1136	69
			NORMA COLETIVA (EM GERAL)	885	54
			Total de Outros	**8326**	**503**

Tabela 4 - Exemplo de 3 categorias utilizadas e a quantidade de exemplos selecionados (FERAUCHE; ALMEIDA, 2011).

4.2.3 Processamento das Ementas

Os vetores atributos-valor, após a tradução para o formato ARFF (Attribute-Relation File Format) foram inseridos na ferramenta WEKA - Waikato Environment for Knowledge Analysis, para que os dados fossem processados por algoritmos de aprendizado de máquina, e assim fossem criados modelos de conhecimento.

Foi utilizado o algoritmo J4.8 como implementação do algoritmo de árvore de decisão disponível por meio da ferramenta WEKA. Trata-se de uma implementação posterior, com poucas melhorias do algoritmo C4.5 revision 8.

Com relação à ferramenta WEKA, esta contém também a implementação do classificador probabilístico Naive Bayes, utilizando a distribuição normal para modelar os atributos.

[7] Quantidade real de exemplos presentes na categoria
[8] Quantidade de exemplos selecionados aleatoriamente

Dessa forma, uma variante do algoritmo SVM, denonimada SMO (Sequential Minimal Optimization), foi utilizada como algoritmo classificador SVM, sendo implementada por meio da ferramenta WEKA. Conforme Park (PARK, 2010) o SMO surgiu da necessidade de implementação de um algoritmo SVM de maneira rápida, simples e capaz de tratar conjuntos de dados mais extensos. Além disso, tem a capacidade de tratar um conjunto de dados esparsos, que contém um número substancial de elementos com valor zero.

Park (PARK, 2010) afirma que a otimização realizada no SMO encontra-se na programação quadrática analítica, ao invés da abordagem numérica tradicional. Portanto, foram montados 3 modelos de aprendizado, utilizando-se as 3 implementações de algoritmos de aprendizado (J4.8, Naive Bayes e SMO), para cada uma das 10 categorias selecionadas.

A técnica utilizada para o treinamento dos algoritmos foi o *cross-validation*. Essa técnica quebra o conjunto de exemplos em dois, um conjunto usado para treinar o algoritmo e outro utilizado para testá-lo, de forma a poder avaliar a precisão do algoritmo treinado.

Quanto à escolha dos exemplos para cada conjunto, esta realiza-se de forma aleatória, e para que o algoritmo aprenda com uma diversidade maior de exemplos, e possa ir ajustando sua taxa de erro, é recomendado repetir o processo várias vezes, alternando os exemplos dos conjuntos (WITTEN; FRANK, 2000).

Desse modo, é possível fixar o número de folds, ou partições dos exemplos a serem utilizados. Foram utilizados 3 folds para o treinamento dos algoritmos. Portanto, os exemplos foram divididos em 3 partes aproximadamente iguais, e uma por vez foi utilizada para testar, enquanto o restante foi utilizado para treinar, ou seja, foram utilizados dois terços para treinar e um terço para testar, sendo repetido o processo três vezes, para que no final cada parte fosse utilizada para teste. Essa maneira de treinar é conhecida como *three fold cross-validation* (PARK, 2010).

Capítulo 5: Aplicação do Método Proposto ao Caso de Estudo

O objetivo deste capítulo é demonstrar como foram aplicadas as sete etapas do método de análise de viabilidade, escolhido para a pesquisa e proposto nesse trabalho ao domínio de aplicação, conforme Capítulo 4. Assim, a estrutura do capítulo segue com a explicação da aplicação de cada etapa ao estudo de caso selecionado.

5.1 Aplicando o estudo de possibilidade de execução da aplicação em computação distribuída

Ao iniciar o estudo do domínio da aplicação de pré-processamento de documentos, especialmente da etapa de pré-processamento das ementas de Ferauche e Almeida (2011), é possível verificar que a aplicação de estudo tem natureza heterogênea.

O pré-processamento é utilizado com o propósito de extrair termos relevantes dos documentos da coleção em estudo, os quais são utilizados como exemplos de treinamento para a criação de classificadores binários de documentos.

Essa aplicação pode ser divida em várias etapas de pré-processamento de categorias. A possibilidade de divisão em tarefas independentes configura a aplicação em estudo em *Bag-of-tasks*.

Cada classificador binário obtido na etapa de aprendizado de classificadores tem objetivos distintos, podendo-se afirmar, neste estudo, que cada uma das tarefas independentes entre si não são utilizadas para resolver um problema único, mas problemas distintos.

5.2 Aplicando o estudo das métricas de avaliação

A métrica escolhida para a avaliação aplicada no presente estudo é o tempo em segundos, sendo possível obter os índices *makespan e Speedup*[9]*.* Nessa etapa, são

[9] A variável *makespan* é utilizada no escopo desta dissertação como a medida de tempo de intervalo entre o início da execução de uma tarefa até sua finalização. Já o *Speedup* é o menor tempo de execução atingido na execução de uma tarefa ou de um lote de tarefas, podendo também ser entendido como a tarefa de execução mais rápida.

consideradas as características de análise de resultados para o apoio à decisão pretendida na investigação.

Para o caso em estudo, buscou-se basicamente obter dados de economia de recursos computacionais, o que justifica a opção pela variável tempo em segundos na definição das métricas.

É importante observar também que os experimentos computacionais foram realizados em ambiente de computação local, ou seja, sem interconexões a redes distribuídas remotamente, portanto, não foi de interesse do estudo a medição de métricas de consumo de largura de banda.

5.3 Aplicando a técnica de divisão da aplicação em tarefas menores

A aplicação da técnica dá-se após o descobrimento da unidade mínima de processamento de tarefas permitidas por meio do estudo do domínio da aplicação que foi necessário na etapa 1.

Essa fase permitiu a classificação da aplicação em várias etapas distintas e independentes, portanto uma aplicação do tipo *bag-of-tasks*. A unidade mínima de divisão é o pré-processamento de uma coleção de documentos de uma determina categoria. Um exemplo dessa divisão é descrito na **Tabela 5**.

Tarefa	Exemplos (+)	Exemplos (-)	Total de documentos por categoria
EXECUCAO	5370	5373	10743
MAO DE OBRA	4308	4311	8619
EMBARGOS	4248	4250	8498
PROVA	3689	3691	7380
PRESCRICAO	2834	2836	5670
Total de Documentos	20449	20461	40910

Tabela 5 - Exemplo de 5 tarefas do domínio da aplicação

As tarefas descritas na **Tabela 5** são compostas pela seleção de documentos, seguindo a técnica de distribuição de quantidades de documentos descritas no Capítulo 4.

Desse modo, cada categoria tem uma coleção de documentos para a realização da etapa de pré-processamento de documentos e, assim, no exemplo, temos cinco unidades de processamento computacionais, podendo serem distribuídas para o ambiente computacional em utilização.

5.4 Realizando a definição do método de transferência dos arquivos

O estudo dos arquivos de entrada e saída necessários para a execução de cada tarefa é o escopo desta etapa.

No caso em estudo, a ferramenta utilizada para o pré-processamento das ementas trabalhistas permite, por meio de arquivo de configuração e passagem de parâmetros de linha de comando de sistema operacional (modelo de arquivos de configuração de *job* e ambiente são descritos nos apêndices), que seja utilizado um diretório específico para a leitura e gravação desses arquivos textos.

Aproveitando esse recurso, é possível utilizar tanto unidades locais como também unidades de rede para a leitura e escrita dos dados de entrada e saída, respectivamente.

Nos experimentos desse trabalho, foram utilizados, para a leitura e gravação dos arquivos, unidades de compartilhamento de arquivos do tipo NFS (Network File System).

Dessa maneira, não houve necessidade de uma transferência de arquivos a cada execução de tarefa porque toda a coleção de textos necessária para a leitura do pré-processador de textos foi transferida no início do processo para essa unidade de rede compartilhada descrita acima.

Ressalta-se que, para a aplicação da etapa, é necessário que seja feito um estudo detalhado dos recursos que estão disponíveis no ambiente de execução das tarefas.

Devido à disponibilidade e à facilidade de utilização de unidades de rede compartilhadas pelo sistema *RMS* escolhido para os experimentos (item 3.4), na realização do planejamento dessa etapa, não houve necessidade de complexos estágios de *File Stage*, como os descritos na **Figura 14 - *File Staging* (WILKINSON, 2010).**.

A decisão pela utilização de unidades NFS enquanto projeto justifica-se pela facilidade que é encontrada sem necessidade de configuração e montagem de complexas etapas como as descritas acima.

É também importante reiterar que essa decisão também somente se torna viável em ambientes de computação local e onde os ambientes *RMS* e *Middleware* dispõem desses sofisticados recursos.

Entende-se que essa análise de utilização seria avaliada com maiores detalhes nos casos de ambiente multi-institucionais, onde os recursos de largura de banda são muito mais críticos que no caso do ambiente em estudo.

5.5 Montando os lotes de execução das tarefas

Nesta etapa foram montados 3 tipos de lotes de execução diferentes, seguindo a política de categorias com maior números de documentos a serem pré-processados. Desse modo, foram obtidos lotes conforme a distribuição das tabelas 6, 7 e 8, respectivamente.

Tipo de Lote 1 – Tarefa	Exemplos (+)	Exemplos (-)	Total de doc. categoria
EXECUCAO	5370	5373	10743
MAO DE OBRA	4308	4311	8619
EMBARGOS	4248	4250	8498
PROVA	3689	3691	7380
PRESCRICAO	2834	2836	5670
Total de Documentos	20449	20461	40910

Tabela 6 - Lote com 5 tarefas

Tipo de Lote 2 – Tarefa	Exemplos (+)	Exemplos (-)	Total de documentos por categoria
EXECUCAO	5370	5373	10743
MAO DE OBRA	4308	4311	8619
EMBARGOS	4248	4250	8498
PROVA	3689	3691	7380
PRESCRICAO	2834	2836	5670
DANO MORAL E MATERIAL	2532	2534	5066
COMPETENCIA	2151	2153	4304
SINDICATO OU FEDERAÇÃO	2094	2096	4190
SERVIDOR PÚBLICO EM GERAL	2020	2023	4043
JORNADA	1979	1982	3961
Total de Documentos	31225	31249	62474

Tabela 7 - Lote com 10 tarefas

Tipo de Lote 3 – Tarefa	Exemplos (+)	Exemplos (-)	Total de documentos por categoria
EXECUCAO	5370	5373	10743
MAO DE OBRA	4308	4311	8619
EMBARGOS	4248	4250	8498
PROVA	3689	3691	7380
PRESCRICAO	2834	2836	5670
DANO MORAL E MATERIAL	2532	2534	5066
COMPETENCIA	2151	2153	4304
SINDICATO OU FEDERAÇÃO	2094	2096	4190
SERVIDOR PÚBLICO EM GERAL	2020	2023	4043
JORNADA	1979	1982	3961
ASSISTENCIA JUDICIÁRIA	1564	1567	3131
HONORARIOS	1559	1561	3120
CONCILIACAO	1377	1379	2756
MULTA	1321	1324	2645
RECURSO	1297	1299	2596
INSALUBRIDADE OU PERICULOSIDADE	1260	1261	2521
APOSENTADORIA	1248	1251	2499
ESTABILIDADE OU GARANTIA DE EMPREGO	1184	1187	2371
NULIDADE PROCESSUAL	1160	1162	2322
Total de Documentos	43195	43240	86435

Tabela 8 - Lote com 20 tarefas

Aplicando essa técnica e executando-se parte dos experimentos, observou-se que havia uma categoria de textos denominada “PREVIDÊNCIA”, cuja quantidade de documentos era superior ao dobro da segunda maior categoria em quantidade de documentos: “EXECUÇÃO”.

Essa observação gerou uma situação não desejada na execução dos lotes, ou seja, todas as vezes que a categoria foi incluída num lote de execução, o escalonador de recursos do sistema sempre processou-a em prioridade, prejudicando o restante do processamento das tarefas.

Devido a essa particularidade do *corpus*[10], foi feita a opção de não considerá-la para a montagem dos lotes e consequente execução.

5.6 Processando as tarefas e recebendo os resultados

Após a realização do agrupamento das tarefas nos lotes da etapa anterior, realiza-se a criação de arquivos de lote para execução, respeitando-se o modelo de processamento e arquivos de descrição de lotes de execução do ambiente que está sendo utilizado nos experimentos.

A **Figura 18** é um exemplo de arquivo de descrição de lote que é o modelo do *RMS* Condor e foi um dos arquivos dos experimentos realizados na dissertação, o qual contempla o processamento de cada tarefa relacionada nas tabelas da etapa anterior.

O resultado dessa etapa é a preparação de arquivos, como o exemplo do modelo da Figura 18, agendamento de execução das tarefas, monitoramento e posterior recebimento dos *logs* de resultados a serem explorados na próxima etapa.

[10] A palavra *corpus* é utilizada em tarefas de mineração de textos como sendo uma coleção de documentos ou dados textuais

```
1 ## 08-12
2 ## Job que executa cinco categorias utilizando cinco máquinas
3 Executable = /usr/bin/perl
4 Universe = vanilla
5 Log     = /mnt/SW_UTIL/Labgrid/corpus/log/tst.job.5Cat.5maquinas.log.$(Process).log
6 Output  = /mnt/SW_UTIL/Labgrid/corpus/log/tst.job.5Cat.5maquinas.out.$(Process).log
7 Error   = /mnt/SW_UTIL/Labgrid/corpus/log/tst.job.5Cat.5maquinas.err.$(Process).log
8 Requirements = machine == "cachorro.labgrid.br" || \
9              machine == "macaco.labgrid.br" || \
10             machine == "tigre.labgrid.br"  || \
11             machine == "girafa.labgrid.br" || \
12             machine == "urso.labgrid.br"
13
14 arguments = -I /mnt/SW_UTIL/Labgrid/corpus/pretext1 /mnt/SW_UTIL/Labgrid/corpus/pretext01/Start.pl
15 InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext01
16 Queue
17
18 arguments = -I /mnt/SW_UTIL/Labgrid/corpus/pretext02 /mnt/SW_UTIL/Labgrid/corpus/pretext02/Start.pl
19 InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext02
20 Queue
21
22 arguments = -I /mnt/SW_UTIL/Labgrid/corpus/pretext03 /mnt/SW_UTIL/Labgrid/corpus/pretext03/Start.pl
23 InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext03
24 Queue
25
26 arguments = -I /mnt/SW_UTIL/Labgrid/corpus/pretext04 /mnt/SW_UTIL/Labgrid/corpus/pretext04/Start.pl
27 InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext04
28 Queue
29
30 arguments = -I /mnt/SW_UTIL/Labgrid/corpus/pretext05 /mnt/SW_UTIL/Labgrid/corpus/pretext05/Start.pl
31 InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext05
32 Queue
```

Figura 18 - Arquivo de descrição de *job* de um lote com 5 tarefas

5.7 Planejando os resultados esperados para a análise

Essa fase foi utilizada para o recebimento dos arquivos de resultados gerados pelo ambiente computacional.

Os arquivos de resultados são arquivos do tipo texto e foram consolidados visando à obtenção de um arquivo de resultado para cada execução experimental. Após o recebimento e leitura desses arquivos, foram gerados tabelas e gráficos que sintetizassem os resultados dos experimentos e permitissem, a partir da leitura dos tempos de processamento, a principal conclusão do objeto de estudo da pesquisa, que é a análise de desempenho da execução de uma aplicação em ambientes de computação distribuída, especialmente nas Grids Computacionais.

Nesse sentido, alguns exemplos de arquivos textos de *log* de *resultados* podem ser consultados nos apêndices dessa dissertação.

Capítulo 6: Resultados Experimentais

Este capítulo descreve os resultados obtidos durante os testes de laboratório realizados no decorrer da pesquisa.

Conforme se pode obervar, foram realizados diferentes tipos de testes, utilizando-se os lotes de processamento definidos com a aplicação das técnicas descritas detalhadamente no capítulo 5, de acordo com as características de ambiente computacional e distribuição de tarefas descritas na **Tabela 9**.

Número de nós	Número de Tarefas	Número de Tarefas	Número de Tarefas
2	5	10	20
4	–	10	20
5	5	10	20
6	–	10	20
10	5	10	20

Tabela 9 - Descrição dos tipos de testes experimentais realizados

6.1 Ambiente de execução dos testes experimentais

Para os experimentos do trabalho de pesquisa, foi utilizado um *Desktop Grid* contendo 10 máquinas. Estas pertencem a um dos laboratórios do Programa de Mestrado do Centro Paula Souza (CEETEPS), e elas contêm um processador *Core* 2 *Duo* e as configurações de *hardware* e aplicativos descritos conforme as tabelas 10 e 11.

Máquina	IP	Processador	Memória	HD
cachorro.labgrid.br	192.168.100.1	3 GHz	3 GB	320 GB
macaco.labgrid.br	192.168.100.2	3 GHz	3 GB	320 GB
tigre.labgrid.br	192.168.100.3	3 GHz	3 GB	320 GB
girafa.labgrid.br	192.168.100.4	3 GHz	3 GB	320 GB
leopardo.labgrid.br	192.168.100.5	3 GHz	3 GB	320 GB
urso.labgrid.br	192.168.100.6	3 GHz	3 GB	320 GB
cavalo.labgrid.br	192.168.100.7	3 GHz	3 GB	320 GB
onca.labgrid.br	192.168.100.8	3 GHz	3 GB	320 GB
panda.labgrid.br	192.168.100.9	3 GHz	3 GB	320 GB
leao.labgrid.br	192.168.100.10	3 GHz	3 GB	320 GB

Tabela 10 - Características dos computadores utilizados

Software	Versão	Descrição
Debian Ubuntu	2.6.38	Sistema Operacional
Perl	5.10	Linguagem de programação para processamento de textos
Pretext II	Atual	Ferramenta para pré-processamentos de dados textuais do LABIC-USP
Condor HTC	7.2.4	Escalonador de recursos e tarefas e middleware para Grid
Java JDK	1.6	Compilador e JVM

Tabela 11 – Características dos softwares utilizados em cada nó

6.2 Planejamento dos lotes de execução

As tarefas foram divididas, conforme a distribuição de tarefas do item 3 da metodologia de pesquisa descritas no capítulo 5, de forma que cada lote de tarefas correspondesse a um *job* enviado ao sistema de interface.

Nos experimentos, foram realizados o agendamento e as execuções de treze lotes de tarefas (**Tabela 12**). O planejamento para a execução dos experimentos contou com as seguintes fases:

1. Definição dos lotes a ser executados;
2. Definição da quantidade de nós para a execução de cada lote;
3. Escolha da quantidade de tarefas para a execução com base no item 2;
4. Criação de um arquivo de agendamento de tarefas para cada quantidade de nós desejados, semelhantes ao modelo da **Figura 18**, de acordo com a distribuição da **Tabela 12**;
5. Agendamento da execução dos 13 *Jobs,* monitoramento e compilação dos resultados.

Número Lote	Lote(Descrição)	Número nós
1	05 Tarefas	2
2	05 Tarefas	5
3	05 Tarefas	10
4	10 Tarefas	2
5	10 Tarefas	4
6	10 Tarefas	5
7	10 Tarefas	6
8	10 Tarefas	10
9	20 Tarefas	2
10	20 Tarefas	4
11	20 Tarefas	5
12	20 Tarefas	6
13	20 Tarefas	10

Tabela 12 - Distribuição dos lotes do experimento

6.3 Resultados

Os resultados obtidos com a aplicação do método proposto na presente dissertação são apresentados nesta seção.

Para tanto, realiza-se a exibição das tabelas e gráficos de resultados, comentários destes e proposta de tendências verificadas pela pesquisa.

6.3.1 Resultados gerais

A **Tabela 13** apresenta o quadro comparativo entre os resultados obtidos por meio da realização de cada tarefa executada individualmente e os resultados gerados com o processamento dos experimentos.

Número Lote	Lote(Descrição)	Tempo(1)	Tempo(2)	Número nós	Desempenho
1	05 Tarefas	4485	1830	2	40,80%
2	**05 Tarefas**	**4485**	**1623,6**	**5**	**36,20%**
3	05 Tarefas	4485	1657,8	10	36,96%
4	10 Tarefas	6406	2706	2	42,24%
5	10 Tarefas	6406	2056,2	4	32,10%
6	10 Tarefas	6406	1987,2	5	31,02%
7	**10 Tarefas**	**6406**	**1816,2**	**6**	**28,35%**
8	10 Tarefas	6406	1890,6	10	29,51%
9	20 Tarefas	8440	3312	2	39,24%
10	20 Tarefas	8440	2491,8	4	29,52%
11	20 Tarefas	8440	2223	5	26,34%
12	20 Tarefas	8440	2054,7	6	24,34%
13	**20 Tarefas**	**8440**	**2117,4**	**10**	**25,09%**

Tabela 13 - Comparação dos resultados locais com as execuções distribuídas experimentais

São apresentados, nessa tabela, os resultados de todos os experimentos realizados, os quais são comparados aos resultados da execução das unidades de tarefas em processamento local.

A primeira coluna é número do lote de execução em ordem de execução dos testes. Na segunda coluna, é apresentada a descrição da quantidade de tarefas que compõem cada lote.

Nas colunas nomeadas “Tempo(1)” e “Tempo(2)” são apresentados, respectivamente, os tempos de processamento das tarefas individualmente de cada lote e o tempo de execução no ambiente de computação distribuída de laboratório.

O número de nós é a quantidade de máquinas utilizadas no processamento de cada lote e, por fim, a última coluna apresenta o desempenho de cada teste obtido pelo percentual de tempo gasto que a solução apresenta em relação ao tempo do processo local. Exemplo: no lote 1 o tempo(2) representa 40,80%do tempo(1).

As três linhas em destaque (lotes 2, 7 e 13) da tabela representam o *Speedup,* métrica de melhor tempo, para cada um dos tipos de lotes, independente do número de nós utilizados no processo.

6.3.2 *Speedup*

A **Tabela 14** apresenta os valores dos melhores tempos em segundos obtidos em cada tipo de lote de tarefas, executados nas configurações de ambiente de processamento local, 2 nós, 4 nós, 5 nós, 6 nós e 10 nós, respectivamente.

Tarefas	Local	2 nós	4 nós	5 nós	6 nós	10 nós
5 Tarefas	4485	1830	-	1623,6	-	1657,8
10 Tarefas	6406	2706	2056,2	1987,2	1816,2	1890,6
20 Tarefas	8440	3312	2491,8	2223	2054,7	2117,4

Tabela 14 - Resumo dos resultados com menores tempos nos testes

Na **Tabela 15**, é possível a visualização da quantidade de tempo percentual gasto pelas configurações de 2, 5 e 10 nós para cada tipo de lotes de tarefas.

Veja o exemplo: a coluna “2 nós” representa o percentual de tempo gasto no processamento de 5, 10 ou 20 tarefas, comparado em relação ao tempo gasto pelo processamento local.

Tarefas	Local	2 nós	5 nós	10 nós
5	4485	40,80%	36,20%	36,96%
10	6406	42,24%	31,02%	29,51%
20	8440	39,24%	26,34%	25,09%

Tabela 15 - Resumo dos menores tempos em percentuais

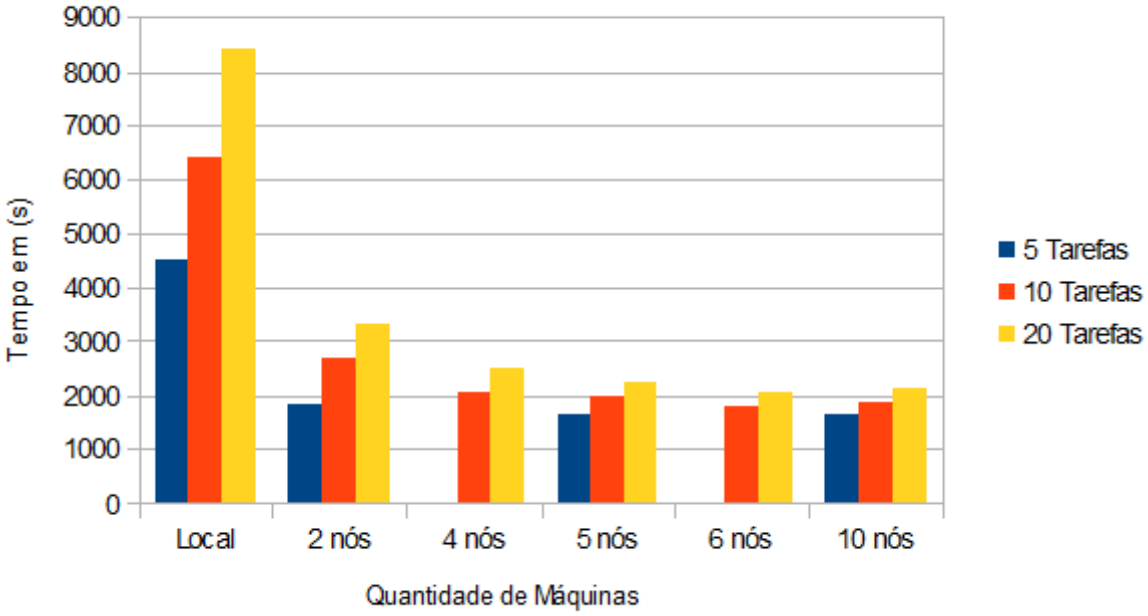


Figura 19 - Tempo de execução para lotes de 5, 10 e 20 tarefas nas configurações: Local, 2, 4, 5, 6 e 10 nós.

A **Figura 20** apresenta um gráfico, no qual é possível a visualização do *Speedup* apresentado na tabela 12.

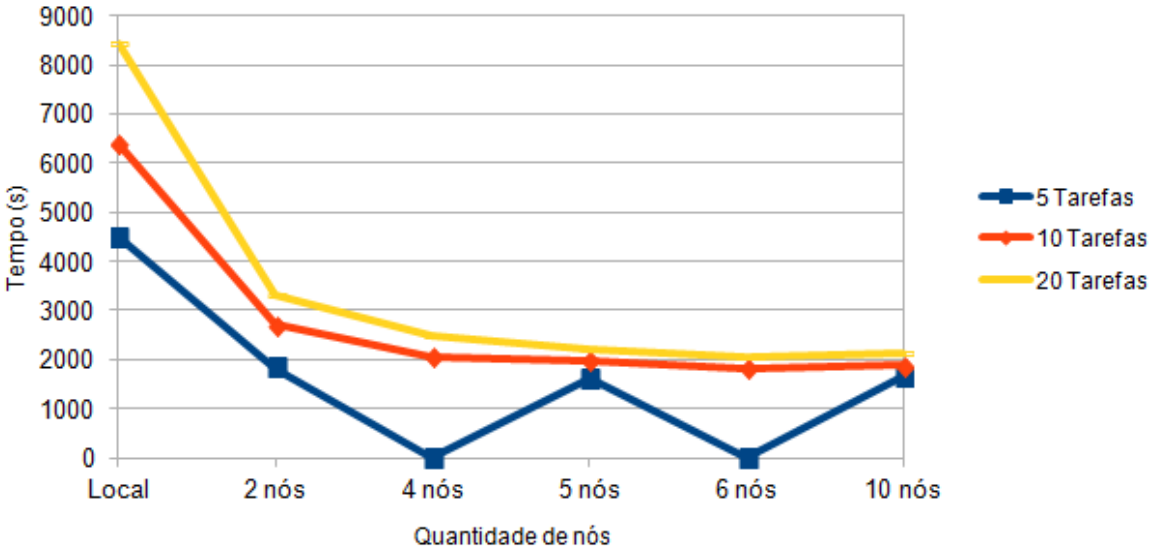


Figura 20 - *Speedup* para os lotes de 5, 10 e 20 tarefas nas configurações: Local, 2, 4, 5, 6 e 10 nós.

O gráfico da **Figura 20** permite a visualização dos pontos ótimos na execução de cada lote de processamento.

Analisando cada um dos tempos em segundos obtidos no processamento de cada lote de tarefas nas configurações de 2, 4, 5, 6 e 10 nós de processamento, é possível afirmar:

1. Há um ganho da ordem de aproximadamente 60% quando é realizado o mesmo processamento dos 3 tipos de lotes na configuração de 2 nós computacionais;

2. Ao migrar os lotes de tarefas de 2 nós para 4 nós, observa-se ainda um ganho de processamento de aproximadamente 11% para o lote de 5 tarefas, em torno de 20% para a execução de lotes de 10 tarefas e um ganho de aproximadamente 32% para lotes de 20 tarefas;

3. Os ganhos percentuais ficam bem menos expressivos quando as tarefas são executadas nas configurações de 5 e 6 nós, comparadas às configurações de

4 nós, atingindo, na maioria dos lotes de tarefas, ganhos bem inferiores a 10%;

4. Quando os lotes de tarefas são executados em ambientes computacionais acima de 5 nós, verificam-se margens muito pequenas de ganhos, tendendo a valores próximos de 0(zero).

6.4 Análise e interpretação dos resultados

Os resultados verificados no item 6.3 permitem a visualização dos ganhos de tempo de processamento quando executados os lotes de tarefas, descritos no capítulo 5, em ambientes de Grids Computacionais, onde as atividades são executadas em paralelo e contam com escalonadores de recursos para a submissão de tarefas e acompanhamento e monitoramento delas.

Além da visualização desses ganhos, pela análise dos gráficos das figuras 19 e 20, é possível fazer a verificação dos pontos ótimos de execução de cada lote de tarefas, o que possibilita o estudo de desempenho da execução de cada tipo de lote em ambientes de 2 a 10 nós computacionais, além da análise de tendências tal qual aquela realizada a partir do gráfico da **Figura 20**.

O estudo de desempenho permite a elaboração de perguntas para apoio à decisão de investimento financeiro em ambientes computacionais. Uma dessas questões, a título de exemplificação, pode ser: “Para o caso em estudo, até quantas máquinas é viável comprar, para a execução de lotes de 20 tarefas?”.

Prosseguindo na análise, é possível verificar também, pelos gráficos da seção anterior, que é viável a compra de máquinas para a execução de qualquer um dos tipos de lotes de tarefas apresentados em até 4 máquinas.

Ao se analisarem os tempos obtidos de *Speedup* no gráfico, é verificada a tendência de que até 5 máquinas torna-se viável a aquisição de máquinas para a montagem de Grids Computacionais dedicados, todavia, acima de 6 máquinas, apesar de ainda ser viável o processamento por haver um ganho pequeno com aumento das máquinas, não é interessante comprá-las, tornando-se recomendável a execução

dos lotes de tarefas em *Grids* Oportunistas, ambiente computacional em que as máquinas de processamento ociosos são utilizadas para os processamento, não havendo,portanto, a necessidade de aquisição de mais computadores.

Capítulo 7: Conclusão e Trabalhos Futuros

A presente dissertação apresentou a proposta e a aplicação de um método de planejamento de estudo de viabilidade de execução de tarefas, aplicável em ambientes de computação distribuída, especialmente às Grids Computacionais.

O método foi proposto com base no estudo do problema de escalonamento de tarefas e recursos desses ambientes e as técnicas aplicadas à etapa de pré-processamento de ementas da justiça trabalhista do Estado de São Paulo.

Para a coleta e análise dos dados experimentais, foi utilizado um *desktop Grid* composto por 10 máquinas de um laboratório, interligadas em rede local, conforme a descrição do item 6.1.

Os resultados obtidos demonstram que, por meio da aplicação do método proposto no trabalho, é possível obter uma consistente análise de desempenho da execução de uma aplicação de um determinado domínio em ambientes de computação distribuída.

Observa-se, ainda, que é possível afirmar que, efetivamente, pela aplicação da metodologia desse trabalho de sete etapas propostas, quando aplicadas ao caso de estudo, torna-se possível comprovar que o método se aplica em sua totalidade.

Considera-se que, como proposta de trabalhos futuros, a serem realizados de modo a dar continuidade à presente pesquisa, é possível destacar os seguintes itens:

1. Aplicar a metodologia em ambientes computacionais que utilizem unidades NFS, também nos processamentos das tarefas locais;

2. Utilizar as etapas descritas neste trabalho em ambientes distribuídos geograficamente, com o foco de mensurar o impacto de largura de banda nos experimentos e visando a adequação da solução aos *Grids* geograficamente distribuídos;

3. Criar um experimento que venha a desconsiderar o tempo de carga dos dados textuais nas métricas de tempo de processamento;

4. Utilizar o mesmo experimento aplicado ao estudo de caso do capítulo 5, mas fazendo o uso dos dados brutos (ementa trabalhista) ao invés do uso de resumo para verificar se seria obtido um aumento na acuidade do aprendizado dos classificadores.

5. Aplicar a metodologia proposta em diferentes configurações de ambiente computacional com o objetivo de conhecer como seria o comportamento das tendências verificadas neste estudo.

Referências

ABRAMSON, D.; GIDDY, J.; KOTLER, L. **High performance parametric modeling with nimrod/g: Killer application for the global Grid?** Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS 2000), April 2000; 520–528.

ALMASI, G.S.; GOTTILIEB, A. **Highly Parallel Computing**. Redwood City: Benjamin. Cummings, 1994.

AMORIM, C. L.; BARBOSA, V. C.; FERNANDES, E. S. T. **Uma Introdução a Computação Paralela e Distribuída**, UNICAMP, Campinas, 1988.

ANDERSON, D. P.; COBB, J.; KORPELLA, E.; LEBOFSKY, M.; WERTHIMER, D., **SETI@home: An experiment in public-resource computing, Communications of the ACM45(11)**, 56–61, 2002.

ANDRADE, N.; CIRNE, W.; BRASILEIRO, B.; ROISENBERG, P. **OurGrid: An Approach to Easily Assemble *Grids* with Equitable Resource Sharing**. *Job* Scheduling Strategies for Parallel Processing, 2003.

ASSIS, L., Nóbrega-Júnior, N., BRASILEIRO, F., CIRNE, W. **Uma heurística de particionamento de carga divisível para Grids Computacionais**. XXIV Simpósio Brasileiro de Redes de Computadores. 2006

BAKER, M. "**Cluster Computing Trends**," Physics Seminar, Liverpool University, 2000.

BERMAN, **F. High-performance schedulers**. In: FOSTER, I; KESSELMAN, C. The Grid: Blueprint for a new computing infrastructure. San Francisco – CA USA: Morgan Kaufmann Publishers Inc, 1999. p. 279-309.

BORGES, E. S.; ALMEIDA, M. A. **Mineração de Textos: Pré-processamento Distribuído de Documentos para Algoritmos de Aprendizado de Máquina**. In: VI WORKSHOP DE PÓS-GRADUAÇÃO E PESQUISA, São Paulo, 2011.

BORGES, E. S.; ALMEIDA, M. A.; FERAUCHE, T. **Mineração de Textos: Uma proposta de Workflow para Pré-processamento de Documentos em Grid Computacional**. In: V WORKSHOP DE PÓS-GRADUAÇÃO E PESQUISA, São Paulo. ANAIS DO V WORKSHOP DE PÓS-GRADUAÇÃO E PESQUISA, 2010

BUDENSKE, J. R.; RAMANUJAN, R. S. **On-line use of off-line derived mappings for iterative automatic target recognition tasks and a particular class of hardware platforms**. In: PROCEEDINGS OF THE 6TH HETEROGENEOUS COMPUTING WORKSHOP, 6., 1997, Washington, DC, USA. (HCW 97): IEEE Computer Society, 1997. p. 96 -.

BUYYA, R.; ABRAMSON, D.; GIDDY, J. **Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid**. Proceedings of the International Conference on High Performance Computing in Asia–Pacific Region (HPC Asia 2000), 2000.

CASANOVA, H. et al. **Heuristics for Scheduling Parameter Sweep Applications** in: Grid Environments. 9th Heterogeneous Computing Workshop, Cancun Mexico, p. 349-363. 01 maio 2000.

CASANOVA, H.; BERMAN, F. **Parameter Sweeps on the Grid with APST**. In: FOX, G.; HEY, T. (Org). Grid Computing: Making the Global Infrastructure a Reality. West Sussex-England: John Wiley & Sons, 2003. p. 773-787.

CASANOVA, H.; HAYES, J.; YANG, Y. **Algorithms and Software to Schedule and Deploy Independent Tasks in Grid Environments**. Workshop On Distributed Computing, Metacomputing, And Resource Globalization, Aussois France, p. 3-17. dez. 2002 .

CASAVANT, T. L.; KUHL, J. G. **A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems**. IEEE Transactions On Software Engineering, Iowa City, p. 141-154. 1988.

CHEDE, C.T. **Grid Computing: um novo paradigma computacional**. Rio de Janeiro: Brasport, 2004.

CIRNE, W., PARANHOS, D., COSTA, L., SANTOS-NETO, E., BRASILEIRO, F., Sauvé, J., Silva, F. A. B., Barros, C. O., Silveira , C. **Running Bag-of-Tasks applications on Computational *Grids*: The mygrid approach**. Parallel Processing, International Conference on, 0:407. 25, 26.

CONDOR, Disponível em: <http://www.cs.wisc.edu/condor>. Acesso em: 10 de janeiro de 2012.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. **Distributed Systems: Concept and Design (3rd Edition)**, Addison Wesley, 2001.

DANTAS, M. **Computação Distribuída de Alto Desempenho: Redes, *Clusters* e Grids Computacionais**. Rio de Janeiro: Axcel Books do Brasil Editora, 2005. 278 p.

DANTAS, M.A.R.; HOSKEN, A. **"Computação Oportunista de Alto-Desempenho: Características, Desafios e Performance"**, II Encontro de Ciência e Tecnologia de Lages, 2003.

DQS (Distributed Queueing System) da Universidade da Flórida. Disponível em <http://www.research.fsu.edu/techtransfer/showcase/dqs.html>. Acesso em 10 de janeiro de 2012.

DONG, F.; AKL, S. G. **Scheduling Algorithms for Grid Computing: State of the Art and Open Problems**. 2006-504 Kingston - Ontario: School Of Computing, Queen´s University, 2006. 55 p.

EBECKEN, N. F. F.; LOPES, M. C. S.; COSTA, M. C. A. **Mineração de Textos**. In: REZENDE, S. O. Sistemas Inteligentes: fundamentos e aplicações. Barueri, SP: Manole, 2003.

FELDMAN, R; SANGER, J. **The Text Mining Handbook**: Cambridge University Press, 2007.

FERAUCHE, T. **APLICAÇÃO DE TÉCNICAS DE MINERAÇÃO DE TEXTOS PARA CLASSIFICAÇÃO DAS EMENTAS DA JURISPRUDÊNCIA DA JUSTIÇA DE TRABALHO DE SÃO PAULO**. 2011. 86 f. Dissertação (Mestrado) - Curso de Tecnologia de Informação Aplicada, Departamento de Programa de Mestrado em Tecnologia: Tecnologia de Informação Aplicada, CEETEPS, São Paulo, 2011.

FERAUCHE, T.; ALMEIDA, M. A. **Aprendizado de Classificadores**. In: VI WORKSHOP DE PÓS-GRADUAÇÃO E PEQUISA DO CENTRO PAULA SOUZA, São Paulo: CEETEPS, 2011.

FERAUCHE, T.; ALMEIDA, M. A.; ITO, M. **Categorizador de Ementas Trabalhista**. In: V WORKSHOP DE PÓS-GRADUAÇÃO E PEQUISA DO CENTRO PAULA SOUZA, São Paulo: CEETEPS, 2010.

FOSTER, I. **Globus Toolkit Version 4: Software for Service-Oriented Systems**. Internacional Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005.

FOSTER, I. T.; KESSELMAN, C.; TSUDIK, G., TUECKE, S. **"A Security Architecture for Computational *Grids*"**, ACM Conference on Computer and Communications Security, 1998.

FOSTER, I. **What is the Grid ? The Three Point Checklist.** GRIDToday, July 2002.

FOSTER, I.; KESSELMAN, C.; NICK, J. M.; TUECKE, S. **The Anatomy of the Grid: Enabling Scalable Virtual Organizations**. The International Journal of High Performance Computing Applications. 2001

FOSTER, I.; KESSELMAN, C. **Globus: A metacomputing infrastructure toolkit**. International Journal of Supercomputer Aplications 11, 2, 115-128. 2005

FOSTER, I; KESSELMAN, C. The **Grid 2, 2nd Edition Blueprint for a New Computing Infrastructure**. Elsevier, 2004.

FOSTER, I; KESSELMAN. **The Grid: Blueprint for a New Computing Infrastructure**. Morgan Kaufmann: San Fransisco, CA. 1999.

gLite, Disponível em <http://glite.cern.ch/> . Acesso em agosto de 2011.

GOLDCHLEGER, A; KON, F. GOLDMAN, A.; FINGER, M.; BEZERRA, G. C. **InteGrade: object-oriented Grid middleware leveraging idle computing power of desktop machines**. Concurrency and Computation: Practice and Experience, 2004.v.16, N. 5, p. 449-459.

GRAHAM, R L; LAWLER, E L; LENSTRA, J K. **Optimization and approximation in deterministic sequencing and scheduling: a survey**. Annals Of Discrete Mathematics, North-holland, p. 287-326. 1979.

GRAM (Globus Resource Allocation Manager). Disponível em: <http://dev.globus.org/wiki/GRAM>. Acesso em 4 jun. 2011.

GRIDCAFE (Org.) **GridCafe**. Disponível em: <http://www.gridcafe.org>. Acesso em: 21 maio 2012.

IBM. **Tivoli Workload Scheduler LoadLeveler**. Disponível em: <http://www-03.ibm.com/systems/software/loadleveler/>. Acesso em: 10 de janeiro 2012.

KONCHADY, M. **Text Mining Application Programming**: Charles River Media, 2006. ISBN 1-58450-460-9.

KRAUTER, K.; BUYYA, K.; MAHESWARAN, M. **A taxonomy and Survey of Grid Resource Mangment Systems for Distributed Computing. Software – Practice and Experience**, v.32, n.2, p.135-164. 2002.

LEGION: Disponível em <http://legion.virginia.edu/FAQ.html>. Acesso em 10 de janeiro de 2012.

LSF: Load Sharing Facility. Disponível em: <http://www.platform.com/workload-management/high-performance-computing>. Acesso em: 10 janeiro de 2012.

LÜLING, R.; MONIEN, B. A . **Dynamic Distributed Load Balancing Algorithm with Provable Good Performance**. In: Proceedings of ACM Symposium on Parallel Algorithms and Architectures (SPAA-93), 1993.

LÜLING, R.; MONIEN, B.; RAMME, F. **Load Balancing in Large Networks: A Comparative Study. Relatório Técnico**, Departamento de Matemática e Ciência de Computação, Universidade de Paderborn, Alemanha, 1993.

MAHESEARAN, M. **Quality of service driven resource management algorithms for network computing**. Proceedings of the 1999 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA ’99), June 1999; 1090– 1096.

MONARD, M. C.; PRATI, R. C.; SOARES, M. V. B. **PreText II: Descrição da Reestruturação da Ferramenta de Pré-processamento de Textos: Relatórios Técnicos do Instituto de Ciências, Matemática e de Computação, Universidade de São Paulo**, São Paulo 2008. Disponível em <http://sites.labic.icmc.usp.br/pretext2/>. Acesso em setembro/2011.

MURPHY, Michael A. **Virtual Organization *Clusters*: Self-Provisioned Clouds on the Grid. 2010. 172 f. Tese (Doutorado)** - Department Of Computer Science, School Of Clemson University, Clemson, SC 29634, 2010.

MYRINET. Disponível em: <http://http://www.myricom.com>. Acesso em: 15 out. 2011.

NASSIF, L. N. **Seleção distribuída de recursos em grades Computacionais, Tese de Doutorado da Universidade Federal de Minas Gerais**, 2006.

PARK, A. F. M I. (2010), **Aplicação de Técnicas de Mineração de Textos para categorização de eventos de Segurança no CITR Gov.,** Dissertação de Mestrado, UnB, Brasília, 82p.

PBS: Portable Batch System. Disponível em: <http://www.nas.nasa.gov/hecc/support/kb/Portable-Batch-System-(PBS)-Overview_126.html>. Acesso em: 10 de janeiro de 2012.

PITANGA, M. **"Computação em Cluster"**, 1ª ed., Rio de Janeiro: Brasport, 2004.

REIS, Q. V. ; SANTANA, M. J. **Escalonamento em Grids Computacionais: estudo de caso 2005. 94 f. Dissertação (Mestrado)** – Ciências da Computação e Matemática Computacional, Instituto de Ciências Matemáticas e Computação – ICMC-USP.

RISTA, C.; PINTO, A. R.; DANTAS, M. A. R. **"OSCAR: Um Gerenciador de Agregado para Ambiente Operacional Linux"**, 4ª Escola Regional de Alto Desempenho, 2004.

ROEHRIG, M.; ZIEGLER, W.; WEDER, P. **Grid scheduling dictionary of terms and keywords**. Disponível em: <http://www.gridforum.org/>. Acesso em: 09 fev. 2012.

RUSSELL, S.; NORVIG, P. **Inteligência Artificial: trad. da 2ª ed. Rio de Janeiro: Elsevier**, 2004.

S. Smallen, H. Casanova, and F. Berman. **Applying Scheduling and Tuning to On-line Parallel Tomography**. Proceedings of Supercomputing 01, Denver, Colorado, USA, November 2001.

SCHOPF, J M. **A General Architecture for Scheduling on the Grid: Special Issue on Grid Computing**, J. Parallel And Distributed Computing, abr. 2002.

SHIRAZI, B. A., HURSON, A. R., KAVI, K. M. (1995). **Scheduling and Load Balancing in Parallel and Distributed Systems**. IEEE Computer Society Press. 1995.

SMALLEN, S. et al. **Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience**. In: HETEROGENEOUS COMPUTING WORKSHOP, 9., 2000, Cancun Mexico. 9th Heterogeneous Computing Workshop. Cancun Mexico: IEEE Computer Society, 2000. p. 1 - 12.

SOARES, M. V. B. **Aprendizado de máquina parcialmente supervisionado multidescrição para realimentação de relevância em recuperação de informação para a WEB, Dissertação de Mestrado**, ICMC / USP, São Paulo, 2009, 95p.

SOUZA, P.S.L. **AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos. Tese (Doutorado)**, ICMC-USP, São Carlos, 2000.

STALLINGS, W. **Computer Organization and Archicteture**, Prentice Hall, 5 Edição, 2000.

STILES, J. R. , BARTOL, T. M., SALPETER, E. E. , SALPETER, M. **M. Monte Carlo Simulation of Neuromuscular Transmitter Release Using MCell, a General Simulator of Cellular Physiological Processes**. Computational Neuroscience, pages 279-284, 1998.

TANENBAUM, A. S. Sistemas Operacionais Modernos. 2. ed. São Paulo - SP: Prentice Hall do Brasil. 1995.

TANENBAUM, A.S. **"Sistemas Operacionais Modernos"**, Rio de Janeiro: Livros Técnicos e Científicos Editora, 1999.

TANENBAUM, A.S.; VAN STEEN, M. **Distributed Systems: Principles and Paradigms**. Prentice Hall; 1st edition, ISBN: 030888931, 2002

Tribunal Superior do Trabalho (TST). **"Instrução Normativa Nº 30 de 2007"**. http://www.tst.gov.br/DGCJ/instrnorm/30.htm. Acessado em 23 de setembro de 2011.

UNICORE. Disponível em <http://www.unicore.eu/unicore>. Acesso em agosto de 2010.

UNIVA CORPORATION. **UNIVA GRID ENGINE**. Disponível em: <http://www.univa.com/products/grid-engine>. Acesso em: 10 janeiro de 2012.

WILKINSON, B. **Grid Computing: Techniques and Application**. USA: Taylor and Francis Group, 2010. (ISBN: 9781420069532).

WITTEN, I. H.; FRANK E. Data Mining – **Pratical Machine Learning Tools e Tecnhiques with JAVA Implementations**: Morgan Kaufmann, 2000.

XU, Chengzhong, LAU, Francis C.M., DIEKMANN, Ralf. **Decentralized Remapping of Data Parallel Applications in Distributed Memory Multiprocessors**. 1997.

Apêndice 1 – Modelos de configuração de ambiente

Arquivo /etc/hosts utilizado nas estações de trabalho Linux do LabGrid.

```
   # Arquivo /etc/hosts de cachorro.labgrid.br
 1 127.0.0.1localhost
 2 127.0.1.1ubuntu
 3 192.168.100.1   cachorro.labgrid.br cachorro
 4 192.168.100.2   macaco.labgrid.br macaco
 5 192.168.100.3   tigre.labgrid.br tigre
 6 192.168.100.4   girafa.labgrid.br girafa
 7 192.168.100.5   leopardo.labgrid.br leopardo
 8 192.168.100.6   urso.labgrid.br urso
 9 192.168.100.7   cavalo.labgrid.br cavalo
10 192.168.100.8   onca.labgrid.br onca
11 192.168.100.9   panda.labgrid.br panda
12 192.168.100.10  leao.labgrid.br leao
13
14 # The following lines are desirable for IPv6 capable hosts
15 ::1     ip6-localhost ip6-loopback
16 fe00::0 ip6-localnet
17 ff00::0 ip6-mcastprefix
18 ff02::1 ip6-allnodes
19 ff02::2 ip6-allrouters
```

Exemplo de Arquivo de configuração do Condor estação central: somente linhas alteradas.

```
 1 ##################################################################
 2 ##  condor_config [LabGridPool]
 3 ##  Data de atualização: 19-11-2011
 4 ##
 5 ##  This is the global configuration file for condor.  Any settings
 6 ##  made here may potentially be overridden in the local configuration
 7 ##  file.  KEEP THAT IN MIND!  To double-check that a variable is
 8 ##  getting set from the configuration file that you expect, use
 9 ##  condor_config_val -v <variable name>
10 ##################################################################
11
12 ##  What machine is your central manager?
13 #CONDOR_HOST= $(FULL_HOSTNAME)
14 CONDOR_HOST     = 192.168.100.1
15
16 ##  Where is the local condor directory for each host?
17 ##  This is where the local config file(s), logs and
18 ##  spool/execute directories are located
19 LOCAL_DIR= /var/lib/condor
20 #LOCAL_DIR= $(RELEASE_DIR)/hosts/$(HOSTNAME)
21
22 ##  Where is the machine-specific local config file for each host?
23 LOCAL_CONFIG_FILE= /etc/condor_config.local
24 #LOCAL_CONFIG_FILE= $(RELEASE_DIR)/etc/$(HOSTNAME).local
25
26 ##  This macro is used to specify a short description of your pool.
27 ##  It should be about 20 characters long. For example, the name of
28 ##  the UW-Madison Computer Science Condor Pool is ``UW-Madison CS".
29 COLLECTOR_NAME   = LabGridPool
30
31 ##  Write access.  Machines listed here can join your pool, submit
32 ##  jobs, etc.  Note: Any machine which has WRITE access must
33 ##  also be granted READ access.  Granting WRITE access below does
34 ##  not also automatically grant READ access; you must change
35 ##  HOSTALLOW_READ above as well.
36 ##
37 ##  You must set this to something else before Condor will run.
38 ##  This most simple option is:
39 ##    HOSTALLOW_WRITE = *
40 ##  but note that this will allow anyone to submit jobs or add
41 ##  machines to your pool and is serious security risk.
42 HOSTALLOW_WRITE = $(FULL_HOSTNAME), *.labgrid.br
43 #HOSTALLOW_WRITE = *.your.domain, your-friend's-machine.other.domain
44 #HOSTDENY_WRITE = bad-machine.your.domain
```

Apêndice 2 - Modelos de arquivos de agendamento de tarefas

Agendamento de para submissão de um lote com 5 tarefas.

```
## 08-12
## Job que executa cinco categorias utilizando cinco máquinas
Executable = /usr/bin/perl
Universe = vanilla
Log     = /mnt/SW_UTIL/Labgrid/corpus/log/tst.job.5Cat.5maquinas.log.$(Process).log
Output  = /mnt/SW_UTIL/Labgrid/corpus/log/tst.job.5Cat.5maquinas.out.$(Process).log
Error   = /mnt/SW_UTIL/Labgrid/corpus/log/tst.job.5Cat.5maquinas.err.$(Process).log
Requirements = machine == "cachorro.labgrid.br" || \
           machine == "macaco.labgrid.br" || \
           machine == "tigre.labgrid.br"  || \
           machine == "girafa.labgrid.br" || \
           machine == "urso.labgrid.br"

arguments = -l /mnt/SW_UTIL/Labgrid/corpus/pretext1 /mnt/SW_UTIL/Labgrid/corpus/pretext01/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext01
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus/pretext02 /mnt/SW_UTIL/Labgrid/corpus/pretext02/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext02
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus/pretext03 /mnt/SW_UTIL/Labgrid/corpus/pretext03/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext03
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus/pretext04 /mnt/SW_UTIL/Labgrid/corpus/pretext04/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext04
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus/pretext05 /mnt/SW_UTIL/Labgrid/corpus/pretext05/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus/pretext05
Queue
```

Agendamento para a submissão de um lote com 10 tarefas em 10 máquinas.

```
## 08-12
## Job que executa 10(DEZ) categorias utilizando 10(dez) máquinas
Executable = /usr/bin/perl
Universe = vanilla
Log      = tst.job.10Cat.10maquinas.log43.$(Process).log
Output   = tst.job.10Cat.10maquinas.out43.$(Process).log
Error    = tst.job.10Cat.10maquinas.err43.$(Process).log
Requirements = machine == "cachorro.labgrid.br" || \
               machine == "macaco.labgrid.br" || \
               machine == "tigre.labgrid.br"  || \
               machine == "girafa.labgrid.br" || \
               machine == "leopardo.labgrid.br" || \
               machine == "urso.labgrid.br" || \
               machine == "cavalo.labgrid.br" || \
               machine == "onca.labgrid.br" || \
               machine == "panda.labgrid.br" || \
               machine == "leao.labgrid.br"

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext1 /mnt/SW_UTIL/Labgrid/corpus2/pretext1/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext1
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext2 /mnt/SW_UTIL/Labgrid/corpus2/pretext2/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext2
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext3 /mnt/SW_UTIL/Labgrid/corpus2/pretext3/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext3
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext4 /mnt/SW_UTIL/Labgrid/corpus2/pretext4/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext4
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext5 /mnt/SW_UTIL/Labgrid/corpus2/pretext5/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext5
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext6 /mnt/SW_UTIL/Labgrid/corpus2/pretext6/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext6
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext7 /mnt/SW_UTIL/Labgrid/corpus2/pretext7/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext7
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext8 /mnt/SW_UTIL/Labgrid/corpus2/pretext8/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext8
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext9 /mnt/SW_UTIL/Labgrid/corpus2/pretext9/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext9
Queue

arguments = -l /mnt/SW_UTIL/Labgrid/corpus2/pretext10 /mnt/SW_UTIL/Labgrid/corpus2/pretext10/Start.pl
InitialDir = /mnt/SW_UTIL/Labgrid/corpus2/pretext10
Queue

```

Apêndice 3 - Modelo de arquivos de configuração do PRETEXTII

Modelo de arquivo de configuração padrão do PRETEXTII.

```
<?xml version="1.0" encoding="utf-8"?>
<pretext
   lang="pt"
   dir="textos"
   log="pretext.log"
   silence="off">

   <maid>
      <number/>
      <html/>
      <simbols/>
      <stoplist dir="stoplist">
         <stopfile>port.xml</stopfile>
         <stopfile>ingl.xml</stopfile>
      </stoplist>
      <stemming dir="steminfo"/>
   </maid>

   <ngram dir="ngraminfo">
      <gram n="1"/>
      <gram n="4"/>
      <gram n="9"/>
   </ngram>

   <report
      ngramdir="ngraminfo"
      discover="discover"
      graphics="graphics"
      taxonomy="taxonomia.txt"
      transpose="disabled">

      <gram n="1"
         max="500"
         min="10"
         measure="tf"
         smooth="disabled"/>
      <gram n="4"
         maxfiles="100"
         minfiles="5"
         measure="tfidf"
         normalize="qua"
         normalizetype="c"/>
      <gram n="9"
         std_dev="0.5"
         measure="tflinear"
         smooth="enabled"
         normalize="lin"
         normalizetype="l"/>
   </report>
</pretext>
```

Exemplo de arquivo de configuração do PRETEXTII utilizado nos experimentos

```
<?xml version="1.0" encoding="utf-8"?>
<pretext
   lang="pt"
   dir="textos"
   log="pretext.log"
   silence="off">

   <maid>
      <number/>
      <simbols/>
      <stoplist dir="stoplist">
         <stopfile>port.xml</stopfile>
         <stopfile>ingl.xml</stopfile>
      </stoplist>
      <stemming dir="steminfo"/>
   </maid>

   <ngram dir="ngraminfo">
      <gram n="1"/>
   </ngram>

   <report
      ngramdir="ngraminfo"
      discover="discover"
      graphics="graphics"
      taxonomy="taxonomia.txt"
      transpose="disabled">

      <gram n="1"
         measure="tfidf"
         smooth="enabled"
         normalize="qua"/>
   </report>
</pretext>
```

Apêndice 4 – Modelo de arquivo de saída de processamento de lotes de tarefas

```
000 (029.000.000) 12/02 15:17:50 Job submitted from host: <192.168.100.2:33080>
...
001 (029.000.000) 12/02 15:17:50 Job executing on host: <192.168.100.2:51642>
...
006 (029.000.000) 12/02 15:17:58 Image size of job updated: 4140
...
006 (029.000.000) 12/02 15:22:58 Image size of job updated: 26264
...
006 (029.000.000) 12/02 15:27:58 Image size of job updated: 34412
...
006 (029.000.000) 12/02 15:32:58 Image size of job updated: 118284
...
006 (029.000.000) 12/02 15:37:58 Image size of job updated: 121980
...
006 (029.000.000) 12/02 15:47:58 Image size of job updated: 122288
...
005 (029.000.000) 12/02 15:52:42 Job terminated.
(1) Normal termination (return value 0)
Usr 0 00:21:18, Sys 0 00:00:30  -  Run Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
Usr 0 00:21:18, Sys 0 00:00:30  -  Total Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
0  -  Run Bytes Sent By Job
0  -  Run Bytes Received By Job
0  -  Total Bytes Sent By Job
0  -  Total Bytes Received By Job
...
000 (029.001.000) 12/02 15:17:50 Job submitted from host: <192.168.100.2:33080>
...
001 (029.001.000) 12/02 15:17:50 Job executing on host: <192.168.100.2:51642>
...
006 (029.001.000) 12/02 15:22:58 Image size of job updated: 28136
...
006 (029.001.000) 12/02 15:27:58 Image size of job updated: 34968
...
006 (029.001.000) 12/02 15:32:58 Image size of job updated: 115044
...
006 (029.001.000) 12/02 15:37:58 Image size of job updated: 119268
...
006 (029.001.000) 12/02 15:42:58 Image size of job updated: 119572
...
005 (029.001.000) 12/02 15:47:24 Job terminated.
(1) Normal termination (return value 0)
Usr 0 00:16:56, Sys 0 00:00:16  -  Run Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00  -  Run Local Usage
Usr 0 00:16:56, Sys 0 00:00:16  -  Total Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00  -  Total Local Usage
0  -  Run Bytes Sent By Job
0  -  Run Bytes Received By Job
0  -  Total Bytes Sent By Job
0  -  Total Bytes Received By Job
...
```