Fábio Campos Lourenço

UMA ABORDAGEM *BRANCH AND BOUND* PARA O RCPSP EM UM AMBIENTE DE COMPUTACAO COLABORATIVA

IM – NCE – UFRJ – Mestrado em Informática

Éber Assis Schmitz
Ph.D., Imperial College

Felipe Maia Galvão França
Ph.D., Imperial College

Rio de Janeiro
Janeiro, 2005

FICHA CATALOGRÁFICA

L292 Lourenço, Fábio Campos.
Uma abordagem branch and bound para o RCPSP em um ambiente de computação colaborativa / Fábio Campos Lourenço. – Rio de Janeiro, 2004.
x, 64f.; il.

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

Orientadores: Éber Assis Schmitz; Felipe Maia Galvão França

Escalonamento de Projetos – Teses. 2. RCPSP – Teses. 3. P2P – Teses. I. Éber Assis Schmitz (Orient.). II. Felipe Maia Galvão (Orient.). II. Universidade Federal do Rio de Janeiro. Instituto de Matemática. Núcleo de Computação Eletrônica. III.Título

CDD

UMA ABORDAGEM *BRANCH AND BOUND* PARA O RCPSP EM UM AMBIENTE DE COMPUTACAO COLABORATIVA

Fábio Campos Lourenço

Dissertação submetida ao corpo docente do Núcleo de Computação e Eletrônica / Instituto de Matemática da Universidade Federal do Rio de Janeiro – UFRJ, como parte dos requisitos necessários à obtenção do grau de Mestre em Ciências em Informática.

Aprovada por:

Prof. Éber Assis Schmitz, Ph.D.

Prof. Felipe Maia Galvão França, Ph.D.

Profª. Érika Fernandes Cota, D.Sc.

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

Rio de Janeiro – RJ

Janeiro de 2005

AGRADECIMENTOS

Aos meus pais, Albino Lourenço e Terezinha Lourenço, que me deram o apoio necessário para que eu conseguisse chegar até aqui.

À minha esposa Fátima por compreender as tardes e os finais de semana de ausência necessários para que este trabalho pudesse ser concluído.

Ao Prof. Éber Schmitz por ter sido um excelente orientador, sempre dando novas idéias e ajudando em tudo que fosse possível. Também agradeço o grande incentivo dado para que eu ingressasse no mestrado.

Ao meu co-orientador Prof. Felipe França cujos conselhos foram essenciais para o refinamento técnico deste trabalho.

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

Aos meus amigos que me incentivaram a entrar no mestrado e a concluir mais esta etapa da minha vida, com destaque especial ao Prof. José Roberto Blaschek, que nunca desistiu de me convencer da necessidade de continuar sempre estudando.

A todas as outras pessoas que de uma forma ou de outra contribuíram para a execução deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AOA	ACTIVITY ON ARC
AON	ACTIVITY ON NODE
API	APPLICATION PROGRAMMING INTERFACE
B&B	BRANCH AND BOUND
CPU	CENTRAL PROCESSING UNIT
DNS	DOMAIN NAME SERVER
DTD	DOCUMENT TYPE DEFINITION
FIFO	FIRST IN FIRST OUT
GRPW	GREATEST RANK POSITIONAL WEIGHT
IP	INTERNET PROTOCOL
JSP	JOB SCHEDULING PROBLEM
MBPS	MEGA-BITS PER SECOND
NPV	NET PRESENT VALUE
P2P	PEER TO PEER
PSP	PROJECT SCHEDULING PROBLEM
RAM	RANDOM ACCESS MEMORY
RCPSP	RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM
UML	UNIFIED MODEL LANGUAGE
XML	EXTENDED MARKUP LANGUAGE

RESUMO DA TESE

UMA ABORDAGEM *BRANCH AND BOUND* PARA O RCPSP EM UM AMBIENTE DE COMPUTACAO COLABORATIVA

Fábio Campos Lourenço

Orientador: Prof. Éber Assis Schmitz
Co-orientador: Prof. Felipe Maia Galvão França

Departamento: Informática

Um projeto pode ser representado por uma rede de atividades formando um grafo de precedência, direcionado e acíclico. Quando a quantidade de recursos existentes é limitada, o problema de determinação do menor tempo de realização do projeto é conhecido como RCPSP (*Resource Constrained Project Scheduling*).

O problema RCPS é reconhecidamente NP-hard. Esta tese mostra um algoritmo distribuído para a solução ótima do problema RCPS usando uma abordagem *branch and bound*. Este algoritmo foi implementado e avaliado em um ambiente de computação colaborativa, do tipo *peer-to-peer*, com escalonamento adaptativo distribuído para balanceamento de carga nos nós computacionais.

Os resultados sugerem a escalabilidade do algoritmo apenas com a adição de nós computacionais.

ABSTRACT OF THESIS

A branch and bound approach for RCPSP in a collaborative computing environment

Fábio Campos Lourenço

Supervisor: Prof. Éber Assis Schmitz, Ph.D.
Co-supervisor: Prof. Felipe Maia Galvão França, Ph.D.

Department: Computer Science

A project can be represented by a network of activities forming a directed acyclic graph. When the amount of resources available is limited, the problem of finding the minimal time of the project is known as RCPSP (Resource Constrained Project Scheduling Problem).

The optimal solution to the RCPSP is NP-hard. This thesis shows a distributed algorithm for the RCPSP optimal solution, using a branch and bound approach. This algorithm was implemented and tested in a peer-to-peer collaborative computing environment, with distributed adaptative scheduling to the load balancing in the computational nodes.

The results suggest the scalability of the algorithm just by adding more computational nodes.

Capítulo 1

INTRODUÇÃO

Neste capítulo o tema do trabalho é apresentado, seguido da motivação, o objetivo esperado e a organização dos capítulos seguintes.

1.1 – Apresentação

O escalonamento de atividades de um projeto é um problema bastante estudado. Os primeiros trabalhos datam da década de 60. Cada novo trabalho publicado apresenta um novo método de abordar a complexidade do problema.

O grande interesse em encontrar um método rápido e preciso para solucionar o escalonamento de um projeto está ligado à competição entre as organizações, que procuram reduzir o tempo e o custo de seus projetos, conseguindo assim uma vantagem em relação aos seus competidores.

1.2 – Motivação

A publicação de estudos e pesquisas sobre computação colaborativa e trabalhos baseados em redes *peer-to-peer* traz uma nova possibilidade para a resolução ótima de problemas de escalonamento. Utilizando um grupo de computadores pessoais, é possível obter um alto ganho no processamento computacional resultante permitindo realizar trabalhos que só seriam possíveis em computadores de grande porte. Esta nova abordagem é o ponto de partida deste trabalho, considerando: (1) que a computação colaborativa permite “atrelar” uma grande massa de poder computacional a um baixo custo e (2) a necessidade de obter soluções ótimas para problemas complexos para efeitos de *benchmark* de heurísticas, considerando a complexidade da solução do RCPSP.

1.3 – Objetivo

O objetivo deste trabalho é apresentar um algoritmo distribuído e escalável para a resolução de problemas RCPSP, que é uma classe de problemas de escalonamento de projetos. Este procedimento terá como base uma arquitetura de computação colaborativa, utilizando um grupo de computadores interligados. A escalabilidade do algoritmo é alcançada através da adição de mais computadores na arquitetura.

A busca de soluções ótimas para problemas de escalonamento é importante para a criação de *benchmarks* utilizados nas avaliações de heurísticas de solução aproximada.

1.4 – Contribuições do trabalho

Este trabalho traz como contribuições: (1) um algoritmo *branch and bound* para P2P e (2) controle dinâmico de carga entre os nós computacionais. Também serviu de base para elaboração dos artigos "Uma Abordagem *Branch and Bound* Para o RCPSP em um Ambiente de Computação Colaborativa", submetido e aceito para publicação no *XII Congreso Latino Iberoamericano de Investigación de Operaciones* (CLAIO 2004), e "Uma Arquitetura XML para Computação Colaborativa P2P" submetido e aceito no 5º Workshop em Sistemas Computacionais de Alto Desempenho (WSCAD 2004).

1.5 – Plano do trabalho

Este trabalho está estruturado em sete capítulos. O capítulo 2 apresenta uma visão geral do problema de escalonamento de projetos e os fundamentos teóricos envolvidos no desenvolvimento este trabalho. O capítulo 3 faz uma revisão geral sobre computação colaborativa. O capítulo 4 apresenta o método de solução ótima para o RCPS proposta nesta tese. No capítulo 5 encontram-se os resultados obtidos do experimento realizado. O capítulo 6 contém a análise dos resultados. O capítulo 7 finaliza a tese apresentando as conclusões finais do trabalho e sugere algumas propostas para trabalhos futuros com o objetivo de dar continuidade a este projeto.

Capítulo 2

ESCALONAMENTO DE PROJETOS

Neste capítulo serão apresentados os conceitos relacionados ao escalonamento de projetos, a partir de uma conceituação básica e descrição de métodos e processos pesquisados até o momento, fornecendo uma visão panorâmica do tema. Pretende-se com isso situar o trabalho no contexto geral de problemas de escalonamento de projetos.

2.1 – Problema de escalonamento de projetos (PSP)

O problema de escalonamento de projetos tem sido tema de diversas pesquisas nos últimos anos. Estas pesquisas enfatizam a modelagem do problema e algoritmos de solução, principalmente para o problema de minimização da duração total do projeto (*makespan minimization*). Em paralelo a estes desenvolvimentos está a pesquisa na área de suporte a decisão de escalonamento de projetos, com ênfase em conjunto de dados, métodos de geração de dados (essenciais para a geração de *benchmarks*) avaliações e comparações de novos modelos, algoritmos e heurísticas. Problemas de escalonamento de tarefas para a minimização da duração total (*makespan*) são reconhecidos como *NP-hard* [4].

2.1.1 Descrição e representação

Problemas de escalonamento de projeto (*Project Scheduling Problems* – PSP) são compostos por atividades, recursos, relações de precedência entre atividades e medidas de desempenho [3]. A seguir, cada componente do problema é descrito.

Projeto

Projeto é uma seqüência de atividades com começo e fim limitadas por tempo, recursos e resultados. De acordo com Martins[2], a melhor definição para projeto talvez seja "um esforço temporário empreendido para criar um produto – ou serviço – que é único". Projetos são componentes críticos da estratégia de negócios de uma organização.

Atividades ou tarefas

Não há um consenso em relação à utilização dos termos atividade e tarefa. Neste documento, ambos os termos são usados indistintamente.

Para um projeto terminar com sucesso, cada atividade deve ser processada em um dos vários modos, quando houver. Cada modo representa uma maneira distinta de realizar uma atividade. O modo determina a duração da atividade, medida em número de períodos de tempo, requerimentos de quantidade e tipos de recursos e possíveis fluxos de caixa ocorrendo no início, durante o processamento ou no final de uma atividade[3].

Recursos

Recursos utilizados por atividades são classificados de acordo com categoria, tipo e valor [2]. A classificação por categoria inclui recursos renováveis, não renováveis, parcialmente renováveis e duplamente restringido.

Renováveis: são restringidos apenas por um período de tempo, ou seja, independente do tamanho do projeto, cada recurso renovável está disponível para cada período de tempo. Exemplos: máquinas, equipamentos e pessoas.

Não renovável: são limitados para todo o planejamento do projeto, sem restrições dentro de cada período. Exemplo: orçamento monetário do projeto.

Parcialmente renovável: limita a utilização de recursos dentro de um subconjunto do planejamento. Exemplo: em um planejamento de um mês com trabalhadores com tempo semanal de trabalho limitado por contrato.

Duplamente restritos: são limitados em um horizonte de planejamento e dentro de um período de tempo[1]. Exemplo: restrições de orçamento que limitam o gasto total do projeto e o gasto em cada período de tempo.

A classificação por tipo distingue cada categoria de acordo com sua função. Exemplo: no desenvolvimento de *software* podemos ter os seguintes tipos de recursos renováveis: desenvolvedor, analista, *designer*, administrador de banco de dados, etc.

Cada tipo de recurso possui um valor associado a ele que representa a disponibilidade do recurso para utilização em atividades concorrentes.

Sempre que existir pelo menos uma categoria desses recursos, o PSP resultante é denominado RCPSP (*resource-constrained project scheduling problem*).

Relações de precedência

Freqüentemente a realização de atividades implica a existência de um encadeamento entre elas, fazendo com que umas devam terminar antes que outras possam começar. Esta característica pode ser explicada representando-se o projeto como um grafo direcionado, onde cada atividade é um nó e a relação de precedência é representada por um arco direcionado entre as duas atividades. Esta representação é chamada de AON (*activity-on-node*), que veremos mais detalhadamente na seção 2.2 –

Critérios de desempenho – Função objetivo

A seguir são enumerados alguns dos mais comuns entre os vários critérios de desempenho empregados para o PSP:

Makespan minimization – provavelmente o mais pesquisado e largamente aplicado objetivo do escalonamento de tarefas de um projeto. O *makespan* é definido como o tempo decorrido entre o início da primeira tarefa e o fim da última, sendo, portanto, a duração de todo o projeto. Uma vez que usualmente se assume o instante de tempo t=0 para o início do projeto, minimizar o *makespan* reduz-se a minimizar o máximo tempo de conclusão de todas as tarefas e tomar o tempo de conclusão da última como o valor do *makespan*.

Net present value maximization – quando o nível significante de fluxo de caixa está presente, representando desembolso para cobrir despesas de iniciação e progresso do projeto, o valor presente líquido (*net present value*, NPV) é a medida mais apropriada como critério de desempenho do projeto. O NPV refere-se ao balanço entre as despesas para conclusão da tarefa e a receita auferida após sua conclusão. A solução de modelos matemáticos deste tipo resulta nos tempos ótimos de início para cada tarefa e também no NPV ótimo do projeto. Dada a complexidade desse problema combinatório, métodos exatos de solução se aplicam somente a pequenas instâncias.

Quality maximization – Icmeli e Ron [7] examinaram, empiricamente, o ambiente de projetos nos EUA e detectaram que o objetivo mais importante, na expectativa do gerente de projeto, é maximizar a qualidade. Com base nisso, em um estudo pioneiro, os autores introduzem um modelo de programação linear mista dedicado a maximizar a qualidade do projeto. A qualidade de um projeto é mensurada com base na quantidade de tempo e dinheiro despendido em refazer atividades que não atendem as especificações dos patrocinadores do projeto.

Cost minimization – em vista de sua larga aplicação prática, este objetivo tem atraído enorme atenção dos pesquisadores nos últimos tempos. Os objetivos baseados em custo apresentam duas vertentes: (a) custo das atividades e (b) custo dos recursos. Em objetivos visando ao primeiro caso, a via pela qual as atividades são executadas, a saber, o tempo de início e/ou os modos escolhidos, resulta em custos diretos que são minimizados. Exemplos

são os tradicionais problemas contínuos do tipo de compromisso tempo/custo. Com objetivos voltados para o custo dos recursos, o escalonamento das atividades influencia o custo indiretamente, via recursos. Como exemplo, pode-se citar o caso clássico de nivelamento de recursos, onde se minimiza o desvio entre o estado de distribuição de recursos contra um estado desejável.

2.1.2 Classificação de problemas de escalonamento

Graham [5] e Pinedo [6] propõem o seguinte esquema de classificação para JSP (*job shop problem*) e pode ser considerado como uma generalização de problemas de escalonamento. JSP é definido por um conjunto finito de tarefas, cada uma consistindo em uma cadeia de operações e um conjunto finito de máquinas. Cada operação deve ser executada durante um período ininterrupto de tempo em uma dada máquina. Segundo Graham e Pinedo um problema de escalonamento é descrito pelo terno $\alpha \mid \beta \mid \gamma$. O campo α diz respeito ao ambiente das máquinas que processarão as tarefas e contém um único dado; o campo β refere-se a características do processamento e restrições e pode conter nenhum, um ou mais dados; o campo γ refere-se aos detalhes acerca do critério de desempenho e, geralmente, contém um único dado. Segue-se uma seleção de modelos JSP, com as correspondentes entradas, entre parêntesis, para o campo α:

Job shop (Jm) – Cada tarefa tem sua própria rota a percorrer através das m máquinas. São de dois tipos: uma tarefa pode visitar qualquer máquina apenas uma vez ou pode ser re-processada em mais de uma máquina, assim permitindo-se recirculação.

Flow shop (fm) – As máquinas são dispostas em série e todas as tarefas têm de ser processadas em cada uma das m máquinas. Todas as tarefas têm o mesmo itinerário e devem ser processados em seqüência, primeiro na máquina 1, depois na máquina 2 e assim por diante. Após completada em um processador, uma tarefa deve aguardar na fila. Via de regra, usa-se o padrão FIFO (*first in first out*) para a fila.

Flexible flow shop (FFs) – Trata-se da generalização do ambiente *flow shop*. Em vez de *m* máquinas em série, há *n* estágios em série, contendo, cada um deles, determinado número de máquinas em paralelo. Em cada estágio, uma tarefa requisita uma única máquina e, na maior parte das vezes, qualquer máquina pode processar qualquer tarefa.

Open shop (Om) – Análogo ao *flow shop*, exceto que agora as tarefas podem ter tempo de processamento nulo e, também, seguir qualquer rota no ambiente de máquinas, quer dizer, diferentes tarefas podem ter diferentes rotas.

Project Scheduling (PS) – Cada tarefa possui duração fixa com um único modo de execução. Há um encadeamento de tarefas fazendo com que umas só possam ser executadas após outras.

Multimode Project Scheduling (MPS) – Análogo ao *project scheduling*, exceto que cada tarefa pode ser executada de vários modos podendo ter duração diferente para cada modo.

Algumas possíveis entradas para o campo β, que se referem às restrições do processamento são:

Preempções (prmp) – O processamento da tarefa pode ser interrompido em qualquer tempo e retomado mais tarde, conforme conveniência do escalonamento. Enquanto isso, uma outra tarefa toma lugar na máquina.

Restrições de precedências (prec) – Obriga que um ou mais tarefas tenham que ser completados antes que determinada tarefa possa ter permissão para iniciar seu processamento.

Para o campo γ, referente ao objetivo a ser otimizado, algumas entradas possíveis podem ser:

Makespan (C_{max}) – Considerando-se *n* tarefas e C_j o tempo que a tarefa *j* termina seu processamento, o *makespan* é definido como *max*(C_1, ..., C_n), equivalente ao tempo de conclusão da última tarefa a deixar o sistema.

Tempo total ponderado de finalização ($\Sigma w_j C_j$) – A soma dos tempos ponderados de finalização fornece uma indicação do custo de se manter as tarefas no sistema em dado escalonamento. O peso w_j é um fator de prioridade relativa entre a tarefa j e as demais.

Esta nomenclatura utilizada para o JSP é adaptada e ampliada de modo a abranger os casos do PSP e identificar precisamente seus problemas. Além de uma notação comum, o ambiente de recursos é descrito (campo α) , características das atividades (campo β) e medidas de desempenho (campo γ), com o qual se pode enquadrar os modelos mais importantes [2]. Como exemplo, podemos citar:

- PS|prec|Cmax: RCPSP, i.e., PSP com *makespan* como função objetivo;
- MPS|prec|Cmax: MRCPSP, i.e., PSP multimodo com *makespan* como função objetivo;

2.2 – Representação gráfica

Com o objetivo de representar graficamente as atividades de um projeto e suas restrições de execução foram criados diagramas que permitissem a visualização rápida das atividades de um projeto e suas dependências.

Além do gráfico de Gantt (Figura 1a), proposto em 1919, que mostra a execução das tarefas em função do tempo, duas outras representações são utilizadas para traduzir a idéia de diagrama de rede de projetos: atividade-no-arco (*activity-on-arc* – AOA) (Figura 1b) e atividade-no-vértice (*activity-on-node* – AON) (Figura 1c).

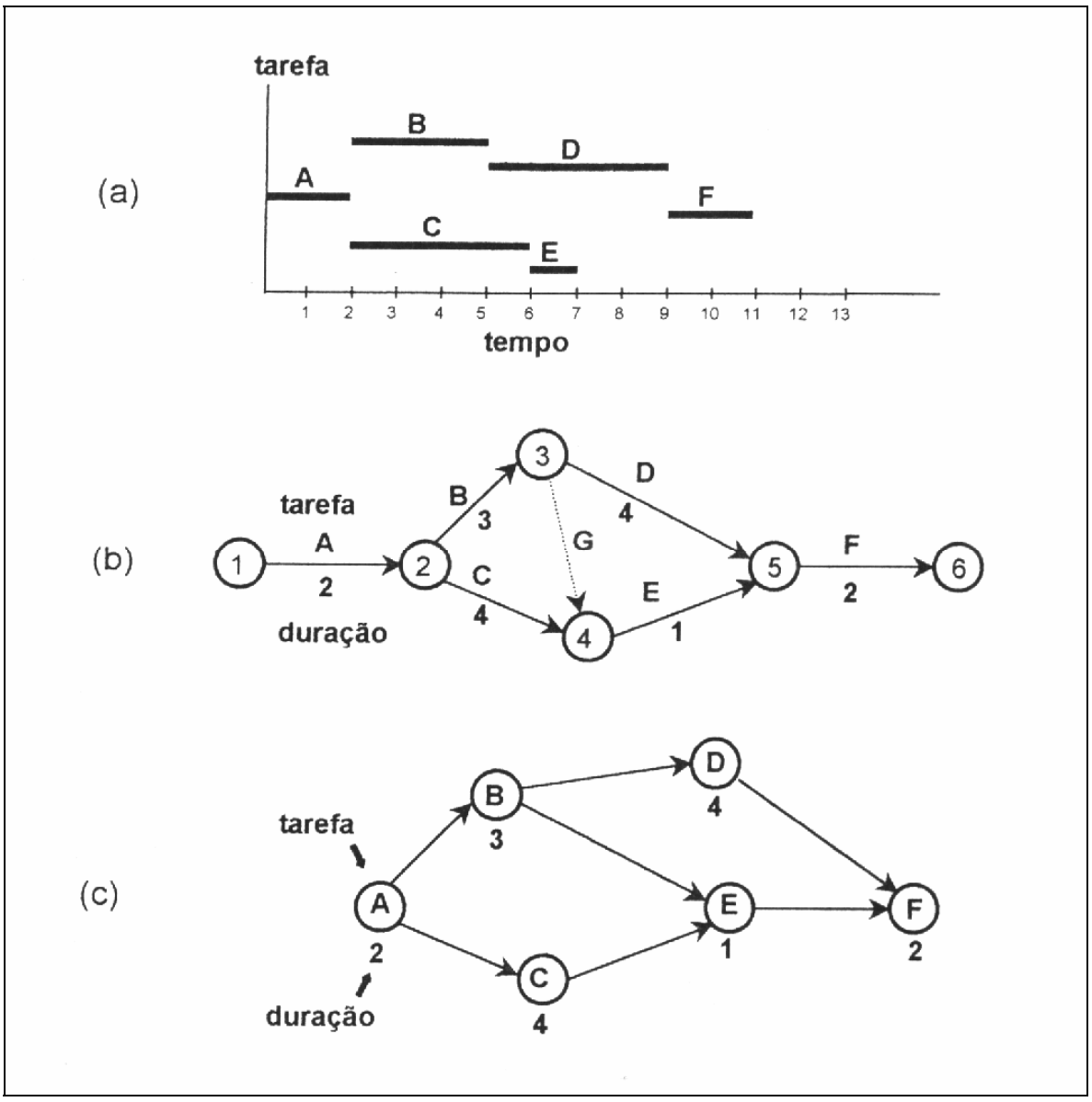

Figura 1 Representações para o PSP: (a) gráfico de Gantt; (b) rede AOA; (c) rede AON[3].

Na representação AOA, os nós representam eventos e os arcos representam as atividades. Atividades fictícias são acrescentadas para representar o começo e o término de um projeto, alem de servir para preservar as relações de precedências entre as tarefas, como é o caso do arco G na Figura 1b. Em redes tipo AON, as atividades e seus parâmetros são representados pelos nós enquanto arcos direcionados indicam suas relações de precedência.

Problemas cujo objetivo seja minimizar o *makespan* quase sempre adotam redes do tipo AON porque dispensa acréscimos de atividades fictícias para registrar a relação de precedência, o que contribui para reduzir o esforço computacional. Esta vantagem pode ser decisiva quando se consideram grandes projetos. Em verdade, o problema de modelar uma rede AOA de mínimo número de atividades fictícias que corresponda a um dado projeto é *NP-hard* [8]. Razões da superioridade da representação AON sobre a AOA podem ser encontradas em [9].

Caminho Crítico

Caminho crítico em um diagrama de rede do PSP é definido como o caminho no grafo de precedência de maior comprimento entre as atividades de início e fim do projeto quando as restrições de recursos são relaxadas. Seu comprimento corresponde ao *makespan* para o caso do PSP de recursos ilimitados calculado com o método CPM (*Critical Path Method*). Nesse caso, todas as atividades que possuam valores diferentes para o tempo de início mais cedo e tempo de início mais tarde apresentam as chamadas folgas. Uma atividade no caminho crítico é aquela que apresenta folga nula.

2.3 – Representação matemática

Dado um grafo acíclico AON direcionado onde os nós representam as atividades e os arcos representam a relação de precedência entre as atividades, a relação de precedência entre os nós pode ser representada pela relação *H(i,j)*, significando que a atividade *i* deve ser finalizada antes que a atividade *j* possa iniciar. É importante notar que se *H(i,j)* e *H(j,z)* então *H(i,z)*. O RCPSP pode ser formulado da seguinte maneira:

Minimizar f_n sujeito à: (1)

$f_i \leq f_j - d_j$ *para todo H(i,j)* (2)

$f_1 = 0$ (3)

$\sum_{i \in S_t} r_{ik} \leq a_k$ *para k=1,...m e t=1,...,*f_n (4)

onde:
n = quantidade de atividades no projeto
m = total de tipos de recursos renováveis
f_i = tempo de término da atividade $i=1,...,n$
H = conjunto da relação de precedência
d_i = duração da atividade i
a_k = quantidade de recursos do tipo k disponível
r_{ik} = quantidade de recursos do tipo k necessários para executar a atividade i
S_t = atividades em execução no instante t

A função objetivo (1) minimiza o tempo de término da atividade fictícia final. Eq. (2) expressa a relação de precedência, enquanto que a Eq. (3) força que a atividade fictícia inicial termine no tempo zero. Eq. (4) expressa que em nenhum instante de tempo durante o projeto a restrição de recurso é violada.

Este trabalho trata apenas do RCPSP cujas atividades possuem duração determinística. Uma outra classe do RCPSP considera que a duração de cada atividade é uma variável aletória com determinada função de distribuição de probabilidade. Neste caso, a solução obtida também é uma função de distribuição. Esta classe de problemas RCPS não é abordada neste trabalho.

2.4 – Algoritmos para solução de problemas RCPS

Os algoritmos para solução do RCPS podem ser enquadrados em duas vertentes: algoritmos de solução aproximada e de solução exata.

2.4.1 Solução aproximada

Os procedimentos cujo propósito é o cômputo de soluções tão próximas quanto possível da solução exata dizem-se métodos de solução aproximada. Freqüentemente, tais métodos são referidos meramente como heurísticas.

Heurística é uma técnica que tenta a resolução de problemas matemáticos via busca de soluções sub-ótimas, em um razoável custo computacional, sem ser capaz de garantir a otimalidade nem indicar o quão próximo do ótimo a solução encontrada está. Anos atrás, decidir-se por abordagem heurística era encarado como admissão de derrota. Mas já se sabe que a maioria dos problemas combinatórios de escalonamento são intratáveis [8].

Estratégias de heurísticas para o RCPS, tendo *makespan* como objetivo, basicamente enquadram-se em quatro metodologias: escalonamento baseado em regra de prioridade, *branch and bound* truncado, conceitos de arcos disjuntivos e abordagens meta-heurísticas (procedimentos que exploram o conhecimento adquirido com a avaliação de soluções previamente visitadas) [3].

Métodos baseados em regras de prioridade

Representam a classe mais importante, a despeito de serem já bastante antigos. Isso se deve, entre outras razões, à sua facilidade na elaboração, implementação e baixo custo computacional, em termos de tempo e memória. Esta característica permite integrá-los como sub-rotinas rápidas em abordagens meta-heurísticas mais sofisticadas. Em verdade, a imensa maioria de programas comerciais de gerenciamento de projeto baseia-se em simples regras de prioridades [10]. Uma estratégia deste tipo para a solução do RCPS é formada por dois componentes: uma regra de prioridade e um esquema de escalonamento. O objetivo do esquema de escalonamento é gerar um escalonamento viável, à medida que acrescenta atividades a um escalonamento parcial. A cada estágio, o esquema constrói o conjunto de todas as atividades escalonáveis chamado de conjunto de decisão. Uma atividade é então tomada desse conjunto, utilizando como critério de escolha uma regra de prioridade.

Branch & bound truncado

Análogo ao esquema enumerativo *branch and bound* (cf. seção 2.4.2), em vez de rastrear toda a árvore de enumeração, conduz apenas uma exploração parcial, baseando-se em algum critério de decisão que limita o espaço de busca do problema. Desse modo, o processamento é interrompido tão logo seja atingido um critério de parada e retorna a melhor solução[22].

Algoritmo de arcos disjuntivos

A idéia básica por trás dessa abordagem é estender as relações de precedência (conjunto de arcos conjuntivos) acrescentando arcos adicionais (arcos disjuntivos) de maneira que conjuntos mínimos proibitivos (i.e. conjuntos de atividades tecnologicamente independentes), que não podem ser escalonadas simultaneamente por limitação de recursos, sejam destruídos e então o escalonamento de menor tempo se mantém viável em precedência e recursos [22].

Meta-heurística

Estratégias de meta-heurísticas exploram o espaço viável de modo global e surgiram em anos recentes. As de maior destaque são *Tabu Search* [23], *Simulated Annealing* [24] [25], algoritmos genéticos[26] e *Ant Colony Optimization*[27].

2.4.2 Solução exata

Neste caso, a meta é o cálculo da solução ótima da função objetivo. Os métodos de solução exata têm sido usados mais freqüentemente para geração de soluções *benchmark*, em face da natural dificuldade de cálculo da solução ótima do problema.

As estratégias até agora utilizadas para a minimização do *makespan* são do tipo: programação dinâmica [28], programação zero - um [29] e esquemas de enumeração implícita com *branch and bound* [13] [30], sendo esta última a estratégia mais utilizada e tema deste trabalho.

2.5 – Branch and Bound

Branch and bound, ou *B&B*, é “um técnica algorítmica para encontrar a solução ótima, utilizando a melhor solução encontrada para limitar a busca pela solução ótima. Se uma solução parcial não puder melhorar a melhor solução, ela é abandonada” [12].

A técnica consiste em 2 passos básicos:

(i) a partir de uma solução parcial, gerar todas as alternativas possíveis para o refinamento da solução (*branching*);

(ii) para cada alternativa de solução, descartar as soluções que não oferecerem possibilidade de melhorar a melhor solução obtida até o momento (*bounding*).

A aplicação sucessiva do passo (i) organiza o espaço de busca do problema em uma árvore de solução que é explorada em busca do resultado ótimo para o problema. O descarte de alternativas de soluções inviáveis é realizado comparando-se o melhor resultado encontrado até o momento com a estimativa do limite superior, para a alternativa de solução parcial em questão.

Esta técnica não é muito útil se todas as soluções forem aproximadamente iguais ou se as soluções iniciais forem muito ruins e melhores soluções apenas são encontradas gradualmente, próximas às folhas da árvore de solução. Em ambos os casos, o descarte de soluções inviáveis é reduzido e o custo é similar a explorar todo o espaço de solução.

2.5.1 *Branch and bound* para o RCPS

Suponhamos o problema de sequênciar *n* tarefas independentes em um processador, dispondo de uma função com a qual se possa calcular o valor numérico da solução, i.e, a qualidade do escalonamento. O método *branch and bound* (B&B) inicia por gerar uma árvore de solução com *n* ramos, representando as *n* opções possíveis para a execução da

primeira tarefa, compondo assim o primeiro nível. Ato contínuo, cada ramo do primeiro nível se ramifica (*branch*) também, nesse caso *n-1* vezes, construindo o segundo nível, para cobrir todas as possíveis alternativas das *n-1* tarefas remanescentes como a segunda a ser processada. Prosseguindo-se dessa forma, sucessivamente até o *n*-ésimo nível, a árvore poderia atingir *n!* ramos, um número que pode ser gigantesco a depender do número de tarefas (Figura 2a) [2].

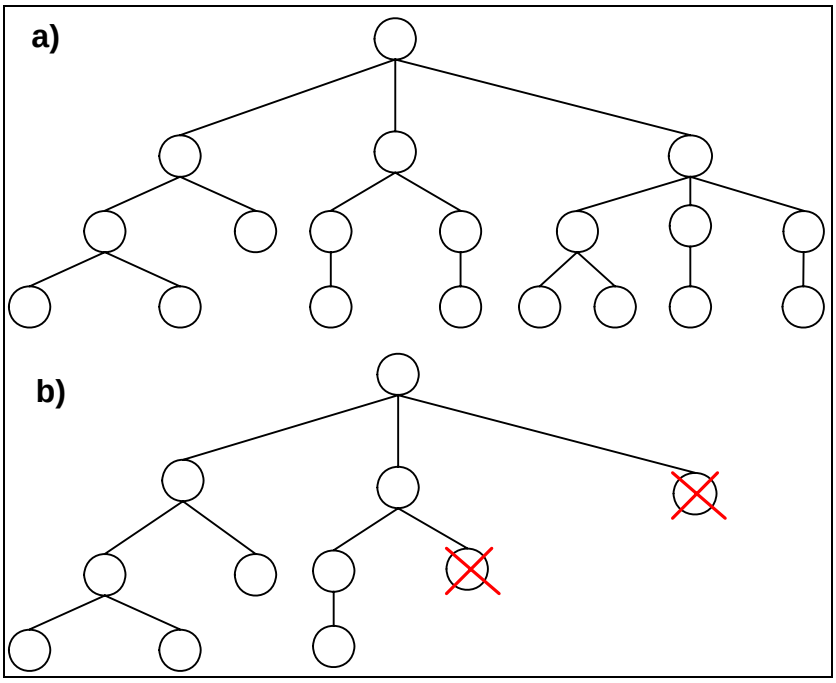

Figura 2 - Exemplo de uma árvore de solução sem poda (a) e com poda (b)

Ao invés de avaliar todas as soluções, o procedimento identifica e suprime (poda) regiões da árvore nas quais se pode provar não haver solução ótima e, assim, diminui o espaço de enumeração (Figura 2b). Essa fase está associada à operação de fixação de limites (*bounding*), ao envolver o cálculo de limitantes inferiores/superiores para a solução ótima em cada um dos nós gerados na fase anterior (*branching*). Sucessivamente analisando e podando partes da árvore, o método encontra a solução exata, se não esgotar o tempo de computação pré-estabelecido.

Para o RCPS, o *branching* é feito em função do grafo de precedência e da disponibilidade de recursos. Normalmente é utilizada uma atividade fictícia, com duração zero e sem consumo de recursos, no início do grafo, garantindo assim que a árvore de solução gerada possui uma única raiz. A partir daí, encontram-se todas as configurações de escalonamento possíveis combinando todas as atividades sucessoras diretas entre si e removendo as combinações que violem a restrição de recursos disponíveis. Sobre os escalonamentos resultantes para fase de *branching*, é aplicado um procedimento de *bounding*. Este procedimento é baseado em uma função que calcule o limite inferior de uma configuração de escalonamento. Se este limite for maior que o limite superior conhecido do problema, a configuração de escalonamento é descartada.

A eficiência da poda tem uma relação direta com a performance do método. A maioria dos trabalhos realizados busca melhorar as estimativas de limites superiores e inferiores para o RCPS [14].

2.5.2 *Branch and bound* paralelo

Existem alguns trabalhos relacionados a implementações *branch and bound* para máquinas paralelas, sendo o trabalho de Simpson e Patterson [13] o mais citado na literatura. Nesta implementação, os processadores paralelos realizam simultâneamente a busca por soluções ótimas em diversas árvores de solução. Informações de limites globais superiores e inferiores são armazenados em uma memória comum, acessível a todos os processadores. A existência de um *hardware* apropriado para a interconexão entre os processadores e a memória compartilhada faz com que seja possível disponibilizar para todos os processadores informações atualizadas sobre os limites do problema, melhorando o processo de poda.

2.6 – Considerações finais

Este capítulo apresentou uma visão geral sobre o escalonamento de projetos – sua classificação e representação. Em seguida, descrevemos os principais trabalhos relacionados a busca de solução para o RCPSP e a aplicação do método *branch and bound* para este problema.

Capítulo 3

COMPUTAÇÃO COLABORATIVA P2P

A computação *peer-to-peer* — P2P — pode ser definida como uma classe de aplicações que, baseadas em um controle descentralizado, i.e., distribuído entre todas as máquinas participantes, usufruem de recursos de armazenamento, processamento, conteúdo e presença humana disponíveis nas fronteiras da Internet. Segundo o estudo de taxonomia de Krauter[15], P2P pode ser classificada como uma forma de organização plana das máquinas, em que todas se comunicam com qualquer outra (*overlay*) sem a necessidade de uma máquina servidora intermediária. Aplicações como o Gnutella[16] popularizaram o P2P na Internet onde, dado que operar em um ambiente de conectividade instável e endereços IPs imprevisíveis, projetos P2P são, geralmente, independentes de DNS e de servidores centrais.

3.1 – Taxonomia

Na perspectiva P2P, a taxonomia dos sistemas de computação é apresentada na Figura 3. Todos os sistemas distribuídos podem ser classificados no modelo *client-server* e no modelo P2P. O primeiro pode ser plano, onde todos os clientes apenas se comunicam com um único servidor central, ou hierárquico, ampliando a escalabilidade. No modelo hierárquico, os servidores de um nível agem como clientes dos servidores do nível superior. O modelo P2P, por sua vez, pode ser puro ou híbrido. Em um modelo puro, não existe um servidor central. No modelo híbrido, um nó do sistema primeiro conecta-se a um servidor a fim de buscar meta-informação, como a identidade de um nó que possua a informação desejada, ou verificar as credenciais de segurança. Após esse passo, uma comunicação P2P

é feita. Também existem soluções intermediárias em que super-nós contêm informações que outros não possuem. Outros nós procuram tais informações nos super-nós.

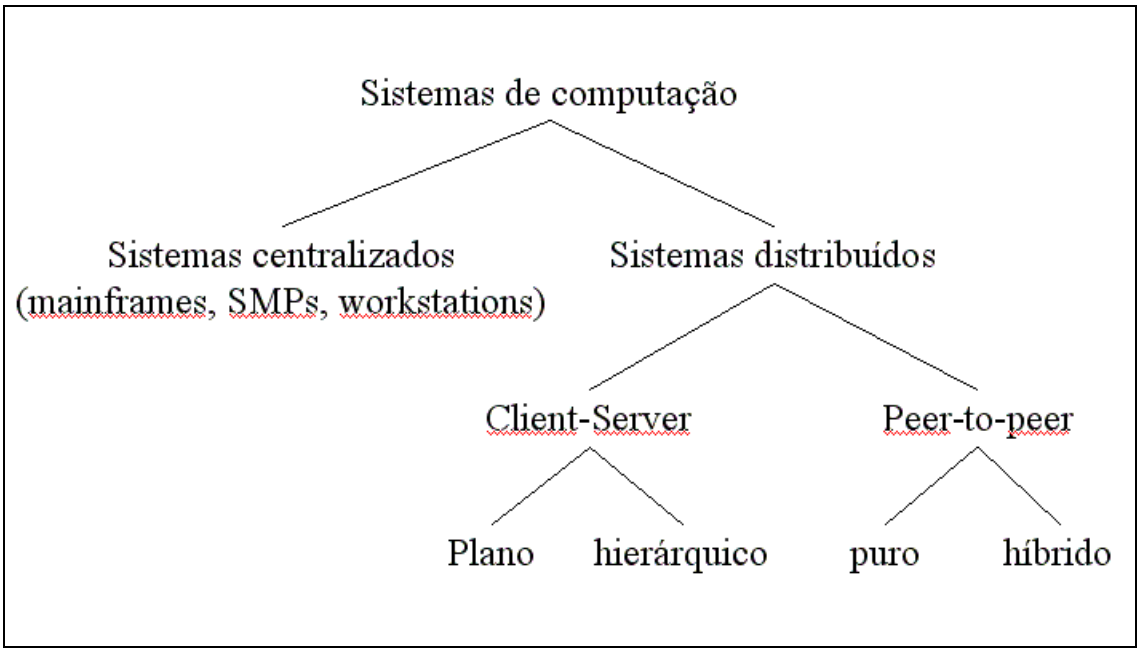

Figura 3 - Taxonomia dos sistemas de computação na perspectiva P2P [19]

3.2 – Características do P2P

P2P é freqüentemente confundido com outros termos, tais como computação distribuída tradicional, *grid computing*, e *ad-hoc networking* (P2P pode ser um meio utilizado para implementar *grid computing*). Para melhor definir P2P, essa seção introduz as características do P2P, que são[17]:

- **Redução do custo de compartilhamento** – Sistemas centralizados que servem muitos clientes, tendem a ter um custo extremamente alto no nó central, o servidor. Uma arquitetura P2P pode ajudar a espalhar esse custo entre todos os *peers* (pares) como, por exemplo, o espaço para armazenamento dos arquivos compartilhados;

- **Aperfeiçoar a escalabilidade/confiabilidade** – Com a carência de uma autoridade central para os nós, aperfeiçoar a escalabilidade e a confiabilidade é uma importante característica. Como resultado, inovações em algoritmos na área de descoberta e disseminação de recursos [18] é uma importante área de pesquisa;
- **Agregação de recursos e interoperabilidade** – Numa abordagem descentralizada, cada nó no sistema traz consigo uma certa quantidade de recursos, tais como potência computacional ou espaço em disco. Aplicações que se beneficiam dessa enorme quantidade de recursos, tais como simulações com uso intensivo de computação, ou sistemas de arquivos distribuídos, naturalmente dependem de uma estrutura P2P que seja capaz de agregar tais recursos;
- **Anonimato/privacidade** – Um usuário pode não querer que outro usuário ou provedor de serviço conheça sobre ele ou seu envolvimento com o sistema. Com um servidor central, é difícil ter certeza do anonimato porque, tipicamente, o servidor será capaz de identificar o cliente, pelo menos pelo endereço da Internet. Implementando uma estrutura P2P em que atividades são feitas localmente, usuários podem evitar ter de prover qualquer informação sobre eles mesmos para outra pessoa. FreeNet é um exemplo de como o anonimato pode ser construído em aplicações P2P. Este usa um esquema que repassa mensagens para ter certeza que o requerente do serviço não possa ser facilmente rastreado;
- **Dinamismo** – Sistemas P2P assumem que o ambiente computacional é altamente dinâmico. Isto é, recursos, tais como nós computacionais, entrarão e deixarão o sistema continuamente. Quando uma aplicação pretende rodar em um ambiente altamente dinâmico, o P2P se encaixa perfeitamente. Programas de comunicação instantânea são usados para informar aos usuários quando as pessoas às quais eles desejam se comunicar estão disponíveis. Sem esse suporte, usuários precisariam testar a disponibilidade das pessoas, enviando mensagens periódicas para estas. Do mesmo modo, aplicações computacionais distribuídas como o SETI@Home, um experimento científico que usa computadores ligados na Internet na busca de inteligência extraterrestre, precisam se adaptar à mudança dos participantes. Estas têm de reenviar trabalhos computacionais para outros participantes para ter certeza que estes não foram

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

- **Habilitando comunicação *ad-hoc* e colaboração –** Relacionado ao dinamismo está a noção de suporte a ambientes *ad-hoc*. Chamam-se ambientes ad-hoc aqueles onde os membros vão e voltam de acordo com sua localização física ou seus interesses correntes.

A Tabela 1 apresenta uma comparação entre os sistemas centralizados, clientes-servidores e P2Ps, indicando os pontos fortes e fracos em cada tipo de sistema.

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

Tabela 1 - Comparação de soluções. Células mais escuras representam os pontos mais fortes

3.3 – Aplicações colaborativas

Idealmente, as aplicações, para rodarem em sistemas P2P, devem ser paralelas, divididas em partes totalmente independentes e possuírem unidades de trabalho de elevada complexidade. Tais características visam garantir a eficiência do sistema como um todo. Na maior parte das vezes, os recursos utilizados pela aplicação estão em máquinas distintas, conectadas em redes que não foram desenvolvidas para apoiar o processamento paralelo, tais como a Internet, que possui largura de banda limitada e alta latência. Portanto, o uso de comunicação excessiva deve ser evitado, pois seu uso pode fazer com que o sistema gaste mais tempo na comunicação do que no trabalho efetivo nos processos. Operações de envio de tarefas para máquinas na rede e troca de mensagens entre estas devem ser usadas com cautela a fim de não degradar a performance do sistema. Os processos devem possuir baixa ou nenhuma sincronização e terem uma complexidade que faça com que o tempo de execução destes torne o tempo de comunicação desprezível [17].

3.3.1 Modelo Master-Worker

O modelo *Master-Worker* [19] [20] é ideal para as aplicações desenvolvidas para rodarem em ambientes P2P. O modelo consiste em duas entidades, o mestre (*master*) e os trabalhadores (*workers*). O mestre é responsável em decompor o problema em vários sub-problemas, distribuir estes entre o conjunto de trabalhadores, garimpar os diversos resultados e, finalmente, uni-los para alcançar a solução do problema. Cada trabalhador recebe mensagens contendo a tarefa que deve ser feita, realiza o trabalho, e envia uma mensagem contendo o trabalho feito [17].

3.4 – IeC – Infra-estrutura Colaborativa

Para o desenvolvimento deste trabalho, utilizamos como base a IeC (Infra-estrutura Colaborativa) [17], desenvolvida no Programa de Pós-Graduação em Informática NCE/UFRJ, que é uma arquitetura colaborativa P2P descentralizada. Esta arquitetura tem como objetivo facilitar a criação de aplicações colaborativas, possibilitando que os desenvolvedores destas aplicações concentrem-se nas particularidades de seus aplicativos, deixando com a IeC os serviços de troca de mensagens e distribuição de carga entre os *peers*.

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

Cada nó computacional que participa do sistema colaborativo deve possuir o módulo IeC instalado. Este módulo é responsável por [17]:

- escalonar as requisições recebidas de aplicações produtoras entre os diversos nós que possuem aplicações registradas para consumi-las;
- receber as respostas enviadas por aplicações consumidores e entregá-las às respectivas aplicações produtoras;
- fornecer, para as aplicações, informações sobre a distribuição de carga da rede colaborativa;
- reenviar requisições não atendidas no tempo máximo configurado (*time-out*).

A interação entre a aplicação P2P e a arquitetura IeC é realizada através de uma API que conta com as seguintes primitivas [17]:

- Registro (DoRegister) – Através desse método, uma aplicação se habilita a utilizar os serviços oferecidos. Caso seja uma aplicação consumidora, o tipo de requisição que ela responde (serviço) é informado como parâmetro. A aplicação recebe um identificador de aplicação (*AppID*) que deverá ser utilizado para identificá-la nas chamadas seguintes à API do IeC;

- Requisitar serviço (SendRequest) – Este método permite que uma aplicação produtora requisite a solução de algum trabalho. É passado como parâmetro o serviço referente ao trabalho e o trabalho a ser processado. A aplicação recebe um identificador para o trabalho solicitado (*ReqID*);

- Buscar resposta (GetResponse) – Esse método permite que uma aplicação verifique se alguma requisição feita anteriormente foi respondida, sendo entregue caso exista. O *ReqID* identifica a requisição que foi respondida. A IeC não avisa à aplicação que existe novas respostas para serem recebidas. A aplicação deve, periodicamente, verificar através desta função se existem novas respostas para o trabalho solicitado;

- Buscar requisição de serviço (GetRequest) – Através deste método, uma aplicação consumidora busca uma requisição para resolver. O retorno será vazio se não existir uma requisição pendente. A IeC não avisa à aplicação que existe novas requisições serem atendidas. A aplicação deve, periodicamente, verificar através desta função se existem novas requisições de trabalho;

- Enviar resposta (SendResponse) – Uma aplicação servidora utiliza este método para enviar a resposta para um trabalho feito, atendendo a uma requisição recebida;

- Buscar distribuição (GetDistribution) – Informa para a aplicação qual a dispersão de pacotes estimada na rede. A IeC, baseada em cálculos estatísticos sobre a quantidade de pacotes em cada nó da rede, retorna para a aplicação 3 níveis possíveis de dispersão: alta, média ou baixa. Esta informação pode ser utilizada pela aplicação para influir no balanceamento de carga do sistema, porém sua utilização não é obrigatória. Maiores detalhes sobre a avaliação da distribuição podem ser encontrados em [17].

A Figura 4 mostra os fluxos de mensagens que circulam no sistema colaborativo. Uma aplicação produtora envia pacotes de requisição para o módulo da infra-estrutura, que envia pacotes de comunicado para outros nós da rede. Aplicações consumidoras recebem pacotes de requisições da arquitetura IeC, processam e enviam os pacotes de resposta de volta para a aplicação produtora. É possível manter, na mesma máquina, aplicações produtoras e consumidoras. Maiores detalhes sobre a IeC podem ser encontradas em [17].

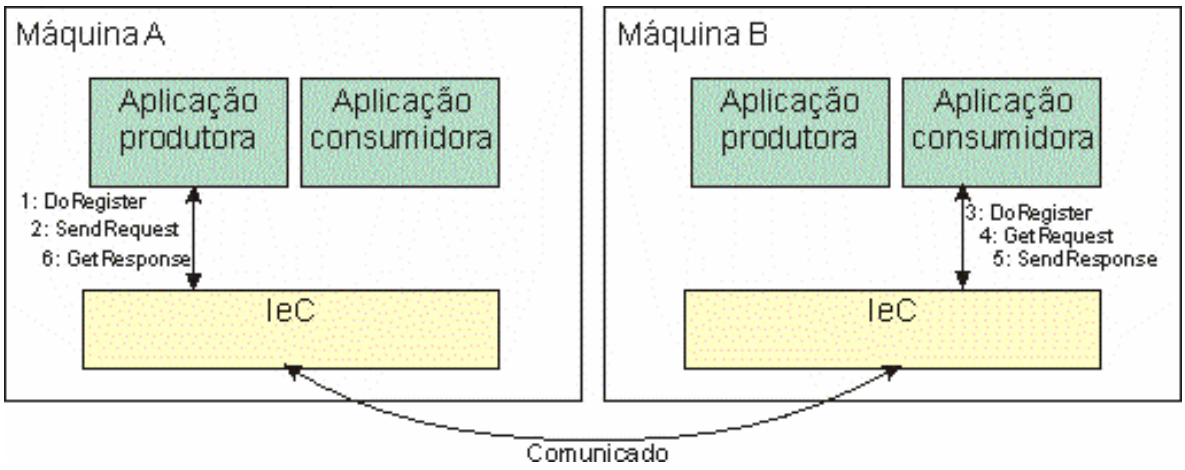

Figura 4 - Relacionamento das aplicações com a infra-estrutura IeC e entre os nós computacionais.

3.5 – Considerações finais

Neste capítulo abordamos o conceito de P2P, suas características e classificação. Posteriormente, citamos características necessárias para aplicações colaborativas que utilizam o poder de processamento coletivo visando obter ganho de eficiência.

Por fim, introduzimos o conceito da arquitetura colaborativa IeC, sobre a qual foi desenvolvido o restante deste trabalho.

Capítulo 4

BRANCH AND BOUND PARALELO PARA O RCPSP USANDO P2P

Este capítulo detalha a abordagem proposta neste trabalho. Aqui é explicado detalhadamente o algoritmo desenvolvido, os problemas encontrados e as ações tomadas para resolvê-los.

4.1 – Abordagem proposta

Como alternativa de solução para o RCPSP, este trabalho apresenta a utilização de um procedimento *branch and bound* (B&B) para a busca de resultados ótimos (Figura 6). O ponto principal desta abordagem é a utilização de uma infra-estrutura colaborativa P2P, que funciona como uma máquina paralela virtual, de baixo custo, simples configuração e escalável.

Em máquinas paralelas tradicionais, a existência de *hardware* apropriado para a interconexão entre os processadores e utilização de uma memória compartilhada entre eles fazem com que seja possível disponibilizar para todos os processadores, informações atualizadas sobre os limites globais do problema, melhorando o processo de descarte (poda). A arquitetura que utilizamos impõe restrições para a comunicação entre processadores e questões como essas devem ser levadas em consideração.

O principal problema da arquitetura utilizada a ser solucionado é a velocidade de interconexão entre os processadores (nós de processamento). Devido à baixa velocidade de transmissão de dados, se comparada com a velocidade de processamento, somos obrigados a minimizar a comunicação entre os nós. Neste aspecto, há uma grande afinidade entre a

arquitetura utilizada e o procedimento B&B, já que o B&B permite particionar o problema gerando pacotes de trabalho independentes para os processadores, conforme veremos a seguir.

Outro problema comum quando utiliza-se processamento paralelo é a geração e distribuição de trabalho entre os processadores. Utilizamos uma variação do modelo *master-worker*, chamado de *master-worker* hierárquico [11], permitindo assim uma maior flexibilidade para a distribuição de pacotes de trabalho, evitando sobrecarga no processador responsável por gerar e distribuir trabalho.

A solução destes problemas faz com que seja possível utilizar uma arquitetura colaborativa para encontrar o escalonamento ótimo para o RCPSP.

4.2 – Particionamento do Problema

Organizando o espaço de busca de uma rede de projeto (Figura 5) através de uma árvore de solução cada caminho da raiz à folha representa um escalonamento válido para o projeto e a altura do caminho é o tempo gasto para a realização do projeto para o escalonamento em questão. A solução ótima está associada ao caminho de menor altura (Figura 7).

Utilizaremos a Figura 5 para exemplificar o particionamento do problema realizado pelo algoritmo. No exemplo, temos dois tipos de recursos, sendo que temos dois recursos disponíveis do tipo um e um recurso disponível do tipo dois. No diagrama de rede, tipo AON, os números dentro dos nós identificam as atividades, os números acima dos nós representam o tempo de duração de cada atividade e o par numérico abaixo de cada nó identifica a quantidade de recursos do tipo um e dois necessários para executar a atividade.

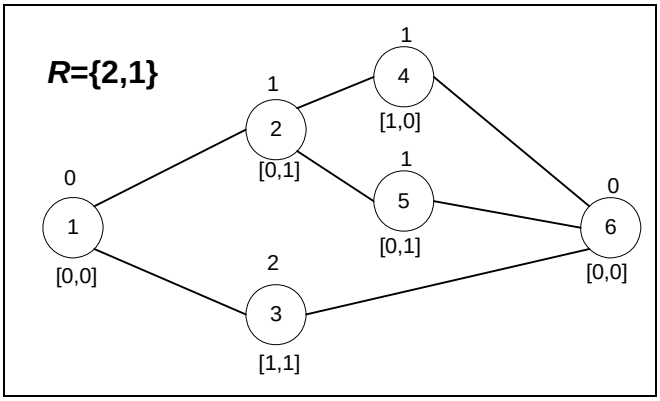

Figura 5 - Exemplo de um diagrama de rede de projeto

O algoritmo inicia colocando a atividade fictícia {1} em execução no instante t=0 e a ramifica (chamada ao procedimento *Branch* do algoritmo da Figura 6). As atividades {2} e {3} são incluídas no conjunto AVL, pois após o término da atividade {1} elas deixam de ter atividades precedentes. As atividades {2} e {3} são incluídas no conjunto AVL. Neste ponto, são gerados 4 subconjuntos de AVL: {}, {2}, {3}, {2,3}, que significam, respectivamente, não escalonar nenhuma tarefa, escalonar a tarefa 2, escalonar a tarefa 3 ou escalonar a tarefa 2 e 3 simultaneamente. O conjunto vazio é descartado pois, como não existe nenhuma tarefa executando, não escalonar nenhuma tarefa não iria levar a uma redução do *makespan*. Escalonar as tarefas 2 e 3 simultaneamente levaria a uma violação da restrição de recursos já que ambas as tarefas necessitam de uma unidade do recurso 2 e só há uma unidade disponível. Então, para t=1, ficamos com duas possibilidades de escalonamento (ESC): ESC[1] = {2} e ESC[2] = {3}. Para cada uma dessas opções é executado o procedimento de poda (*Bound*) para descarte de caminhos considerados inviáveis. No exemplo, a poda não é realizada e obtemos a árvore de solução completa da Figura 7.

```
t ← tempo atual (nível da árvore de solução).
ACT ← lista de atividades em execução no instante t (nó da árvore de solução).
Procedimento Branch (ACT)
    AVL = conjunto de atividades que não possuem precedentes ou cujas precedentes já
          foram finalizadas até o instante t
    Gerar todos os subconjuntos ESC de AVL que não violem o limite de recursos no
          instante t+1.
    t ← t+1
    ACT ← lista de atividades em execução no instante t
    Se o numero de elementos de ACT e AVL = 0   então
        Retorna t
    Senão
        Para cada ESC [i]
            Se Bound(ACT + ESC[i]) > limite_superior
                Descartar ESC [i]
            Senão
                Branch(ACT + ESC[i])
Procedimento Bound (ACT)
    Calcular para o escalonamento parcial definido em ACT, o limite inferior do problema a
          partir do instante t.
```

Figura 6 – Algoritmo *B&B para RCPSP*

O paralelismo no processamento é obtido particionando o problema pelos ramos da árvore de busca. Como não há necessidade de troca de informações entre os nós de ramos diferentes, temos uma grande redução da comunicação entre os diferentes processadores. No algoritmo da Figura 6, conseguiríamos obter este paralelismo se cada chamada recursiva ao procedimento *Branch* fosse executada em um processador distinto.

Cada conjunto de cores exemplifica a alocação dos nós em processadores. No exemplo, supomos a utilização de apenas três processadores. Os nós em cinza foram processados pelo processador número um, os nós em branco pelo número dois e os nós listrados foram processados pelo processador número três.

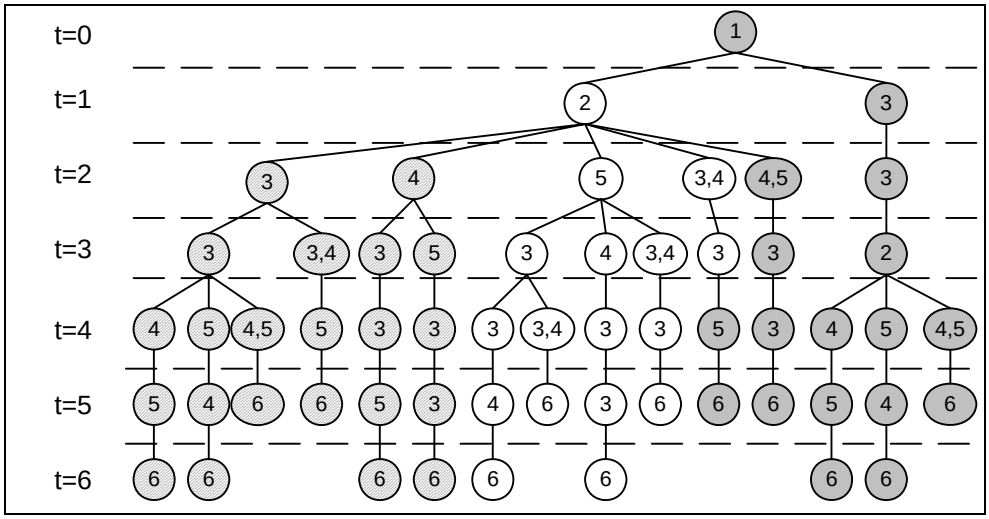

Figura 7 - Árvore de solução com a identificação de particionamento

O processador que recebe um nó deverá gerar (ramificar) todas as configurações possíveis de escalonamento para o intervalo de tempo seguinte, levando em consideração a restrição de recursos. O particionamento é feito dinamicamente, de forma que não é possível identificar no início do problema como será a alocação das atividades do projeto. Informações de carga colhidas durante o processamento definem para onde será enviado o próximo nó da árvore, por isso temos a situação do caminho {(1),(2),(3,4),(3),(5),(6)} que, após a geração do nó (3), é enviado para o processador um para o término do processamento.

Para o exemplo da Figura 7, temos que o menor *makespan* possível é de 4 unidades de tempo, existindo várias configurações de escalonamento onde se pode obter este tempo. O *makespan* ótimo pode ser dado pelo comprimento do menor caminho entra a raiz e a folha. Como sempre trabalhamos com atividades fictícias – duração zero – no início e fim do problema, subtraímos 2 unidades do tamanho do caminho para obter o *makespan*. Neste exemplo, como a variação entre a pior solução para a duração total do projeto e a solução ótima é de apenas uma unidade, provavelmente não haveria redução do espaço de busca aplicando-se algum mecanismo de poda.

4.3 – *Master-worker* hierárquico

No modelo *master-worker* tradicional um processador (*master*) é responsável por quebrar o problema em pacotes de trabalho e distribuir para os demais processadores (*workers*) que processam os pacotes e retornam a resposta sempre para o nó *master*.

Com o objetivo de aumentar o paralelismo e evitar a situação de que um único processador seja responsável por gerar todo o trabalho, decidimos utilizar o modelo *master-worker* hierárquico para a geração e alocação dos pacotes de trabalho. Neste modelo, cada processador pode atuar como *master* ou *worker*. Dessa maneira, utilizando como exemplo a Figura 7, o processador que recebeu o nó (4) no nível t = 2 da árvore de solução deverá responder ao seu respectivo *master* o menor escalonamento a partir deste nó. Como este nó irá ramificar, o processador *worker* pode solicitar que outros processadores ajudem a encontrar a solução deste sub-problema. O processador *worker* age então como *master* para este sub-problema, conforme diagrama da Figura 8.

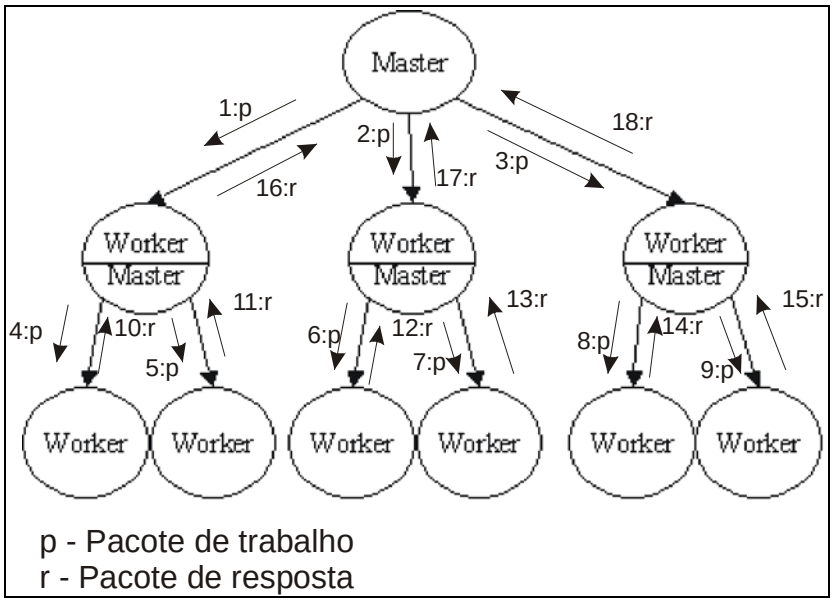

Figura 8 - Master-Worker com hierarquia

Cada processador *worker* tem a tarefa de receber um nó da árvore de solução, expandi-lo e disponibilizar as alternativas viáveis de solução para os demais processadores

da rede. Cada processador *worker* conhece o trabalho realizado até o nó sendo processado. Ao encontrar um nó folha, o processador *worker* irá calcular o tempo total como sendo a distância da raiz até a folha e irá retornar ao seu *master* imediato o menor resultado encontrado. O processador *master* após receber a resposta de todos os seus *workers*, identifica a menor resposta e repassa para seu *master* imediato. Repetindo o processo sucessivamente até que o processador *master* que iniciou o processamento seja alcançado, obtemos o escalonamento ótimo para um problema RCPS.

Baseado no algoritmo da Figura 6, temos o algoritmo da Figura 9 para realizar o *branch* do RCPSP no modelo *master-worker* hierárquico.

```
END – conjunto de atividades finalizadas até o instante t
ACT – conjunto de atividades em execução no instante t
ESC – conjunto de escalonamento de atividades possíveis para o instante t+1
Procedimento MenorMakespan (t, END, ACT)
    END ← END + lista de atividades finalizadas até o instante t+1.
    AVL ← atividades que não possuem atividades precedentes ou que todas as atividades
          precedentes pertençam a END.
    Se ACT e AVL forem vazios então
        Retorna t
    Senão
        Branch(AVL)
        Para cada subconjunto ESC de AVL resultante do branch faça
            EnviarParaWorker (MenorMakespan (t+1, ENC, ACT + ESC[i]))
        T[i] ← resposta do worker para o i-ésimo envio
        Quando todas as respostas T[i] forem recebidas
            Retorna Min(T[i])
```

Figura 9 - Algoritmo RCPSP Hierárquico

4.4 – Algoritmo colaborativo

Para prover o ambiente de computação colaborativa, usamos o IeC – um *middleware* desenvolvido na UFRJ para prover comunicação entre os processadores. O IeC fornece

uma API (*Application Program Interface*) que torna transparente para a aplicação o controle da comunicação e o balanceamento de carga.

O programa que roda em cada processador é idêntico – a única diferença de execução em cada processador se dá pelo fato que o programa que inicia a busca pela solução de um problema RCPS executa alguns passos a mais que os demais. A Figura 10 mostra o diagrama de atividades, baseado na UML, referente aos passos executados pelo algoritmo.

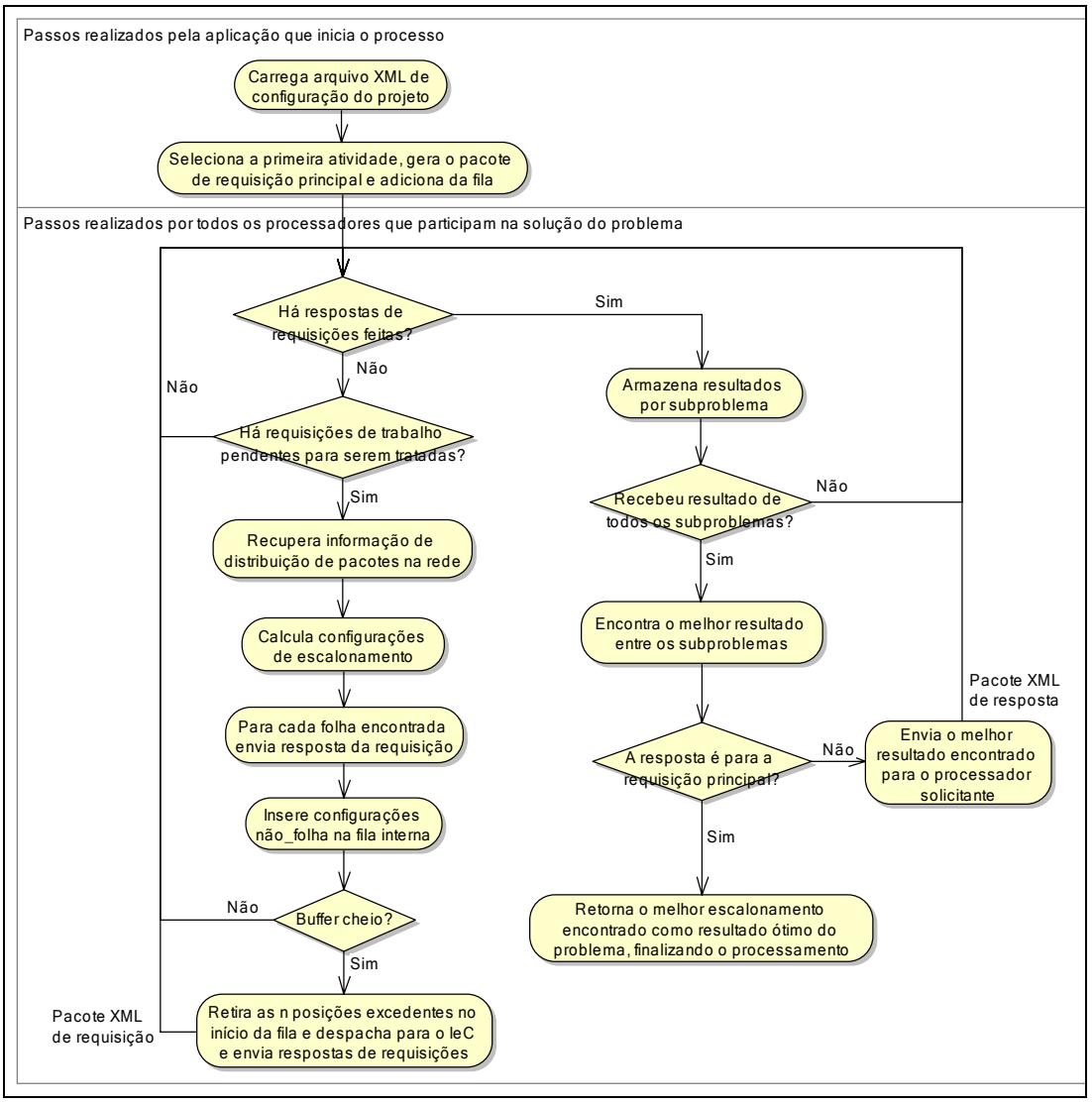

Figura 10 - Representação dos passos do algoritmo colaborativo

O programa que inicia a resolução do problema lê os dados de configuração do projeto em formato XML[32] (Figura 11). O arquivo possui a informação de total de tarefas, disponibilidade total de recursos e as tarefas do projeto, indicando o consumo de recursos por tarefa, duração e lista de sucessores de cada tarefa.

```
<!ELEMENT RCPS_Otimo (total_tarefas, total_recursos, configuracao)>
<!--Quantidade de tarefas do projeto-->
<!ELEMENT total_tarefas (#PCDATA)>
<!--Total de recursos disponíveis-->
<!ELEMENT total_recursos (recurso+)>
<!--Relação de tarefas com lista de precedência-->
<!ELEMENT configuracao (tarefa+)>
<!--Descreve cada tarefa do projeto-->
<!ELEMENT tarefa (sucessores, duracao, recursos)>
<!--Identificador único da tarefa-->
<!ATTLIST tarefa
  id_tarefa CDATA #REQUIRED>
<!--IDs das tarefas sucessoras separados por vírgula-->
<!ELEMENT sucessores (#PCDATA)>
<!--Duração de cada tarefa em unidades de tempo-->
<!ELEMENT duracao (#PCDATA)>
<!--Recursos necessários para a execução de cada tarefa-->
<!ELEMENT recursos (recurso+)>
<!--Quantidade do tipo de recurso-->
<!ELEMENT recurso (#PCDATA)>
<!--Tipo do recurso-->
<!ATTLIST recurso
  id_recurso CDATA #REQUIRED>
```

Figura 11 - DTD do arquivo de configuração do diagrama de rede do projeto

Da lista de atividades sucessoras, o programa deriva a lista de atividades antecessoras e encontra a primeira atividade do projeto, como sendo a atividade que possui o conjunto vazio de atividades antecessoras. O primeiro pacote de trabalho é criado tendo como objetivo a minimização do *makespan* do projeto a partir da primeira atividade. A resposta a este pacote determina a solução do problema.

Cada pacote de trabalho carrega, além da configuração do projeto, a indicação do escalonamento que deverá ser considerado na iteração seguinte do processador que receber o pacote, o limite superior para o problema e os conjuntos de atividades que estão em execução e que ainda não foram executadas (Figura 12). O pacote então é encaminhado ao *buffer* de pacotes da aplicação, cujo funcionamento é detalhado na seção 4.4.1 .

```
<!ELEMENT RCPS_Otimo (threshold,total_tarefas, total_recursos, configuracao)>
<!—Limite superior para o makespan o projeto-->
<!ELEMENT threshold (#PCDATA)>
<!--Quantidade de tarefas do projeto-->
<!ELEMENT total_tarefas (#PCDATA)>
<!--Total de recursos disponíveis-->
<!ELEMENT total_recursos (recurso+)>
<!--Relação de tarefas com lista de precedência-->
<!ELEMENT configuracao (tarefa+, estado_anterior)>
<!--Descreve cada tarefa do projeto-->
<!ELEMENT tarefa (sucessores, duracao, recursos)>
<!--Identificador único da tarefa-->
<!ATTLIST tarefa
  id_tarefa CDATA #REQUIRED>
<!--IDs das tarefas sucessoras separados por vírgula-->
<!ELEMENT sucessores (#PCDATA)>
<!--Duração de cada tarefa em unidades de tempo-->
<!ELEMENT duracao (#PCDATA)>
<!--Recursos necessários para a execução de cada tarefa-->
<!ELEMENT recursos (recurso+)>
<!--Quantidade do tipo de recurso-->
<!ELEMENT recurso (#PCDATA)>
<!--Tipo do recurso-->
<!ATTLIST recurso
  id_recurso CDATA #REQUIRED>
<!--Descrição do estado do processamento em um determinado momento-->
<!ELEMENT estado_anterior (tempo_atual, act, elg)>
<!--Tempo atual no cronograma calculado-->
<!ELEMENT tempo_atual (#PCDATA)>
<!--Tarefas em execução-->
<!ELEMENT act (idtarefa)>
<!--Tarefas a executar-->
<!ELEMENT elg (idtarefa)>
<!--Identificação da tarefa-->
<!ELEMENT idtarefa (#PCDATA)>
<!--Tempo de quando a tarefa iniciou o processamento-->
<!ATTLIST idtarefa
  tempo_inicio CDATA #IMPLIED>
```

Figura 12 - DTD para o XML do pacote de trabalho

Deste ponto em diante, todos os processadores executam os mesmos passos, conforme representado da Figura 10, alternando entre a busca de respostas a requisições feitas e novas solicitações de trabalho pendentes. A busca por respostas de requisições é feita por consulta direta ao IeC, através da API fornecida. Se existir nova resposta, o IeC entrega para o algoritmo o pacote XML de resposta (Figura 13). Para a busca por requisições, o

algoritmo busca novas requisições primeiramente no *buffer* interno de trabalho, que será explicada posteriormente, buscando, em seguida, novas requisições diretamente do IeC caso o *buffer* interno esteja vazio.

```
<!ELEMENT RCPS_Otimo_Result (tempo_projeto,tarefas,tempo_processamento)>
<!--Unidades de duração do projeto(makespan)-->
<!ELEMENT tempo_projeto (#PCDATA)>
<!--Escalonamento de tarefas-->
<!ELEMENT tarefas (tarefa)>
<!ELEMENT tarefa EMPTY>
<!--ID da tarefa e tempo de início-->
<!ATTLIST tarefa
  id CDATA #REQUIRED
  inicio CDATA #REQUIRED>
<!--Total, em seg, do tempo de processamento gasto por cada processador-->
<!ELEMENT tempo_processamento (#PCDATA)>
```

Figura 13 - DTD para o XML do pacote de trabalho

A seguir, na Figura 14, temos a descrição dos procedimentos realizados quando há requisições de trabalho pendentes.

procedimento Trata Requisição

Se buffer_interno vazio então

PCK ← pacote de trabalho do IeC

Senão

PCK ← pacote de trabalho de FilaInterna

Inicializa tempo_atual, limite_superior, ACT, END, RecursosDisponiveis e GrafoPrecedência com dados de PCK

EnviaResposta_IeC(PCK, MenorMakespan (tempo_atual,END,ACT))

Figura 14 - Algoritmo de tratamento requisições

A unidade de trabalho que cada processador deve executar é muito simples: dado um conjunto de atividades que podem ser colocadas em execução simultaneamente, gerar todas as combinações possíveis de execução (configurações), sempre respeitando a restrição de recursos. Além disso, a cada configuração é aplicada uma função de estimativa de limite mínimo, ocasionando o descarte da configuração caso este limite seja maior que o limite superior do projeto.

Todos esses passos são executados muito rapidamente nos processadores atuais, fazendo com que o tempo gasto para preparar os pacotes XML de comunicação e o envio destes pacotes pela rede seja maior que o tempo gasto com o processamento, causando perda de desempenho. Para melhorar o desempenho, utilizamos uma fila interna de requisições, funcionando como um *buffer* de pacotes, para reduzir a comunicação pela rede. Com essa abordagem, junto com a utilização de uma estratégia para diminuir o consumo de memória e a utilização de uma função de poda para descartar configurações inviáveis, conseguimos obter bons resultados com a implementação do algoritmo utilizando um ambiente colaborativo. A seguir, descrevemos as características principais das estratégias implementadas.

4.4.1 Fila de requisição interna – *Buffer* de pacotes

Uma preocupação constante no desenvolvimento de aplicações paralelas está no volume de comunicação entre os processos, uma vez que a velocidade da rede de intercomunicação sempre é inferior a velocidade dos processadores atuais. Quando falamos de um ambiente colaborativo P2P, o problema se agrava, uma vez que a rede de intercomunicação é muito heterogênea, podendo até contar com processadores interligados por conexão discada.

É importante então fazer com que cada pacote de trabalho recebido gere o máximo de processamento sem a necessidade de consulta ao meio de interconexão, além de maximizar o paralelismo da busca pelo resultado.

Para reduzir a utilização da rede de interconexão, inserimos no algoritmo uma fila de requisições, que funciona como um *buffer* de pacotes. Com isso, cada aplicação possui uma quantidade de *n* posições para armazenar pacotes de trabalho, ao invés de repassar para a IeC. Com a utilização deste *buffer* conseguimos simular que um pacote de trabalho recebido gere uma quantidade de processamento maior do que seria gerado sem o *buffer*, reduzindo assim a necessidade de recorrer ao meio de intercomunicação. O funcionamento

do *buffer* é explicado na Figura 15. Sem a utilização do *buffer* (Figura 15a) o tempo decorrido entre a chegada de um pacote e a geração de novos pacotes para o IeC é *T* (tempo de processamento de um pacote). Com a utilização do *buffer*, este tempo aumenta para *nT*, sendo *n* o número de pacotes armazenados no *buffer*.

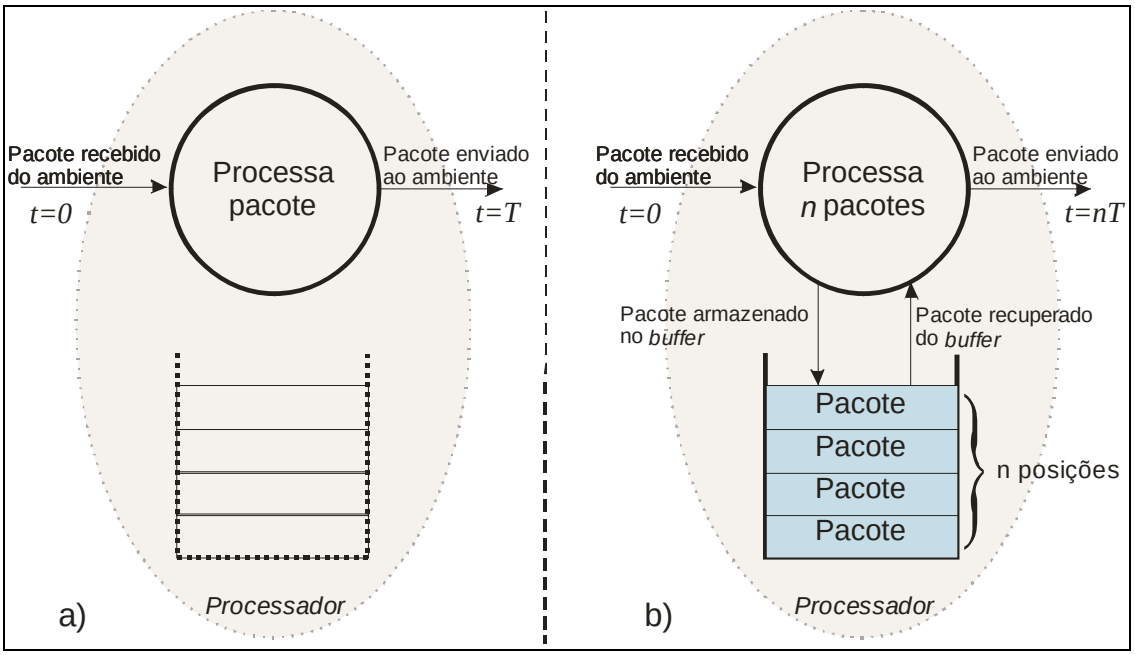

Figura 15 - Representação gráfica do efeito do *buffer* no processamento do pacote

Toda nova requisição gerada é inserida no final do *buffer* mesmo que este já esteja com o máximo de ocupação. Neste caso, as *m* primeiras posições excedentes são retiradas do início do *buffer* e encaminhadas ao IeC. A busca por novos pacotes de trabalho pendentes é feita primeira no *buffer* interno. Somente se o *buffer* estiver vazio o processador busca novas requisições no IeC.

Os pacotes de trabalho que se encontram no *buffer* são recuperados para processamento a partir do fim do mesmo. Dessa forma, os pacotes mais recentes são recuperados primeiro. A utilização desta estratégia contribui para um menor consumo de memória, como veremos no item seguinte.

Dimensionamento do *buffer* interno

O tamanho do *buffer* interno deve ser dimensionado corretamente. Um valor muito pequeno aumenta a intercomunicação e o número de pacotes de trabalho na rede, reduzindo a eficiência. Um valor muito grande para o tamanho do *buffer* diminui o paralelismo, uma vez que os processadores começam a reter mais trabalho do que conseguem realizar, deixando outros processadores ociosos. Novamente, temos uma redução na eficiência.

Resultados práticos (seção 5.1 –) também mostraram que o tamanho ideal do *buffer* varia de acordo com o tamanho do problema e da quantidade de processadores disponíveis. Um problema que possua uma árvore de solução menor deverá ter um *buffer* menor que o problema com maior árvore de solução. Da mesma forma, de nada adianta reduzir o tamanho do *buffer* para aumentar o paralelismo se só tivermos dois processadores disponíveis participando do processamento. Resultados experimentais também mostraram que, independente do tamanho do *buffer* escolhido, quando a busca pelo resultado se aproximava do fim, ocorre uma perda de paralelismo, uma vez que o tamanho fixo do *buffer* e a escassez de pacotes de trabalho faziam com que apenas alguns processadores permanecessem trabalhando.

Para resolver os problemas encontrados, optamos por utilizar um tamanho variável para o *buffer* que fosse auto-ajustável em função do andamento do processamento. O tamanho inicial do *buffer* é zero e é incrementado ou decrementado de acordo com informações obtidas da arquitetura. A IeC possui três níveis de indicação da dispersão dos pacotes na rede – Baixa, Média ou Alta. Sempre que inicia um novo processamento, o programa verifica o estado da dispersão e toma as ações necessárias.

Se a dispersão estiver baixa, significa que os processadores estão com uma carga equivalente de trabalho, indicando que o paralelismo alcançado está satisfatório. Com isso, o algoritmo incrementa o tamanho do *buffer* interno, tomando para si uma maior quantidade de pacotes de trabalho.

Se a dispersão for média, temos a indicação que alguns processadores estão com menos carga de trabalho que outros e que é necessário enviar pacotes de trabalho para o IeC distribuir. Por isso, o tamanho do *buffer* é decrementado, fazendo com que pacotes comecem a ser retirados do *buffer* e entregues ao IeC. Os pacotes retirados do *buffer* e entregues ao IeC são retirados do início do *buffer*, ou seja, pacotes mais antigos, porque este pacotes tem maior probabilidade de gerar uma maior sub-árvore de solução. O IeC é responsável por entregar os pacotes de trabalho primeiramente para os processadores menos carregados.

A dispersão alta indica que uma quantidade proporcionalmente grande de processadores está se tornando ociosa. A ação é similar ao que ocorre com a dispersão média. A diferença é que o decremento do tamanho do *buffer* é feito em passos maiores.

Com esta estratégia, conseguimos melhorar a distribuição de pacotes, fazendo com que os processadores finalizassem o processamento juntos.

```
procedimento EnviarParaWorker (trabalho)
    Dispersão ← informação de dispersão de trabalho do IEC
    Se Dispersão =Alta
        ReduçãoAlta (tamanho_buffer)
    Senão
        Se Dispersão =Média
            Redução (tamanho_buffer)
        Senão
            Aumento (tamanho_buffer)
    Se buffer_interno cheio
        Enviar execução de trabalho para IEC
    Senão
        Executar trabalho no mesmo processador
```

Figura 16 - Algoritmo de controle dinâmico do buffer interno

4.4.2 Estratégia para redução do consumo de memória

Devido a natureza exponencial do problema, tanto o consumo de memória, junto com o tempo de processamento, sempre foram fatores limitantes. Apesar de cada processador contar com sua memória própria e de cada um resolver apenas alguns sub-problemas, ainda assim a quantidade de informação que precisa ser armazenada é muito grande. Todas as soluções dos sub-problemas precisam ser armazenadas até que seja encontrada uma solução ótima local. Somente após ser encontrada pelo menos uma solução ótima local as demais soluções do sub-problema podem ser descartadas, liberando a memória utilizada.

Para favorecer o paralelismo, a função de ramificação (*branch*) gera todos os nós subseqüentes ao nó atual, funcionando como uma busca em largura. Cada nó gerado poderá então ser encaminhado a um outro processador e prosseguir dali em diante, independente dos demais. O problema com a busca em largura está exatamente no consumo de memória. A árvore tem todos os seus níveis explorados simultaneamente e a ocorrência de um nó folha é tardia. Como só há liberação de memória quando os nós folhas começam a ser encontrados, teríamos que possuir memória para armazenar toda a árvore de solução nos processadores, o que é inviável, visto que, facilmente, estas árvores ultrapassam a casa dos milhões de nós.

Para agilizar a busca por nós folha sem prejudicar o paralelismo utilizamos simultaneamente a busca em largura e busca em profundidade, esta última para encontrar mais rapidamente nós folha e assim liberar a memória mais rapidamente.

A escolha do tipo de busca é feita sobre o *buffer* interno. Os nós gerados pela ramificação são inseridos no *buffer*. Os últimos nós inseridos são recuperados pelo algoritmo para dar continuidade ao processamento, fazendo com que o *buffer* se comporte como uma pilha. Assim, enquanto existe trabalho no *buffer*, é priorizada a busca em profundidade, objetivando encontrar um nó folha. Quando a quantidade de pacotes é maior que o tamanho do *buffer*, os nós no início são enviados para o IeC, fazendo com que o

buffer funcione como uma fila fazendo com que a busca em largura seja utilizada para gerar o paralelismo no processamento.

Na Figura 17 podemos ver a representação gráfica do *buffer* e as buscas realizadas. O problema tem início na Figura 17a quando o processador A recebe a primeira tarefa e ramifica as configurações (2), (3) e (4), que são inseridas no *buffer*. No passo seguinte (Figura 17b), a última configuração inserida no *buffer* – configuração (4) – é recuperada e novamente ramificada (busca em profundidade). As configurações (5), (6) e (7) resultantes são inseridas no *buffer*. Supondo um *buffer* com capacidade três, a inclusão destas configurações ultrapassa o limite do *buffer* em duas unidades. Dessa forma, as duas primeiras configurações do *buffer* são enviadas a outros processadores, iniciando uma busca em largura (Figura 17c).

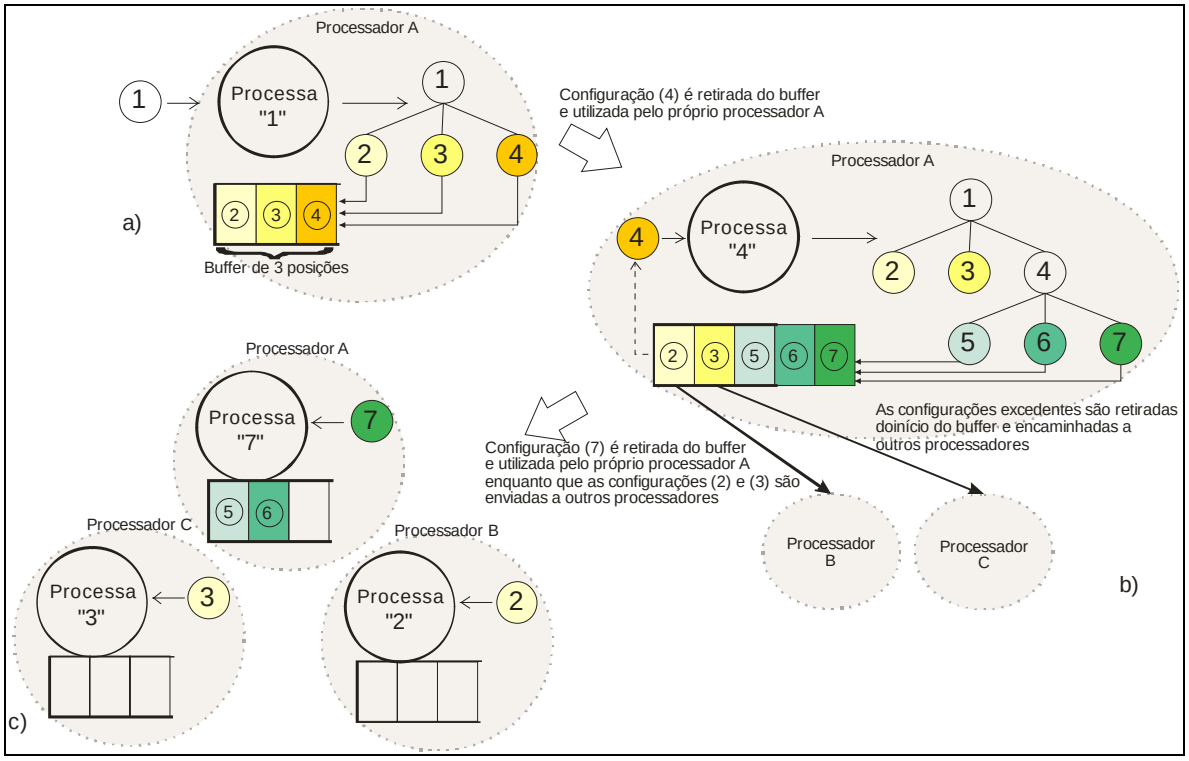

Figura 17 - Representação gráfica do efeito do *buffer* sobre o tipo de busca

4.4.3 Descarte de configurações

As funções que estimam o limite inferior da solução do problema devem ser as mais simples possíveis, para não impactar o processamento total e retornar o resultado mais próximo possível do valor real.

Para reduzir o espaço de busca de soluções, implementamos o seguinte mecanismo para identificar configurações inviáveis, que, apesar de bastante simples, apresentou resultados satisfatórios para o objetivo deste trabalho:

– O processador que inicia o problema aplica uma heurística no projeto para encontrar o limite superior para o *makespan*. Utilizamos a heurística GRPW (*Greatest Rank Positional Weight*) [22], onde as atividades são priorizadas para entrar em execução em função da quantidade de atividades sucessoras. Quanto maior o número de sucessoras, maior a prioridade da atividade:

– O valor encontrado no procedimento acima é repassado para todos os processadores junto com o pacote de trabalho (tag *threshold* do XML);

– Para cada ramificação gerada é aplicada uma função de estimativa de limite inferior, partindo do escalonamento já encontrado até o ponto da ramificação (ver algoritmo Figura 6). Para estimar o limite inferior para uma configuração, encontramos o escalonamento das tarefas restantes descartando a restrição de recursos. Dessa forma garantimos que o *makespan* obtido nunca será maior que o *makespan* com a restrição de recursos.

Existem várias outras funções para cálculo do limite inferior para o RCPSP[14]. Muitas pesquisas são feitas na tentativa de melhorar a estimativa do limite inferior, uma vez que quanto mais próximo estiver a estimativa do valor real maior o descarte de configurações inviáveis e menor o espaço de busca.

4.5 – Considerações finais

Neste capítulo apresentamos o trabalho desenvolvido. Descrevemos o funcionamento do algoritmo e as estruturas criadas para o suporte ao processamento. Destacamos os principais problemas encontrados e as soluções adotadas para resolvê-los. Mais ainda, explicamos como é feita a geração, distribuição e descarte de configurações de escalonamento, sempre com a preocupação com o tempo de comunicação entre os processadores e o consumo de memória.

Capítulo 5

AVALIAÇÃO

Para avaliar a performance do algoritmo, foi realizado um experimento que contou com 10 computadores, CPU Athlon 1.1GHz com 256Mb de memória RAM, conectados por uma rede local de 10Mbps. Todos os computadores utilizavam Windows XP como sistema operacional. Utilizamos o gerador de *benchmark* ProGen [21] para gerar as configurações de projeto que foram utilizadas no experimento. O arquivo de configuração do ProGen pode ser visualizado na Figura 32 (anexo).

O projeto, gerado pelo ProGen, utilizado no experimento possuia 15 atividades, considerando as atividades fictícias inicial e final. O problema escolhido gerou, no total, 766.038 requisições de trabalho, que ocasionou 5.099.470 nós na árvore de solução. Foram realizados 1.857.823 descartes. Através da exploração parcial da árvore e utilizando heurísticas para obtenção do resultado ótimo, estimamos que o problema sem descartes geraria aproximadamente 400.000.000 de nós na árvore de busca.

O experimento consistiu na execução do algortimo RCPS proposto sobre o problema gerado pelo ProGen, variando o número de computadores participantes da computação.

No experimento, avaliamos a distribuição de carga entre os processadores ao longo do tempo de processamento, tempo total para encontrar a solução ótima do problema e o *speed-up* do algoritmo. *Speed-up*, ou aceleração, é uma medida de eficiência do algoritmo paralelo e é dado pela seguinte fórmula:

$$S = Ts / Tp$$

onde S é o speed-up, Ts é o tempo de execução serial e Tp o tempo paralelo.

Exemplificando, se ao utilizarmos uma máquina paralela de quatro processadores o tempo de processamento for reduzido a ¼ do valor do processamento serial, temos que o *speed-up* é igual a quatro. Quanto mais próximo o *speed-up* do número de processadores, melhor a eficiência do algoritmo.

5.1 – Resultados

A seguir, da Figura 18 à Figura 22, temos os gráficos das distribuições de pacotes para os cenários de 2, 4, 6, 8 e 10 processadores, respectivamente. Em todos eles utilizamos o valor de 200 pacotes para o *buffer* interno e 100 pacotes para o fator de espalhamento. O fator de espalhamento indica a quantidade de pacotes que serão encaminhadas ao IeC antes do *buffer* interno começar a ser utilizado. No experimento, cada processador participante irá enviar 100 pacotes para o IeC antes de inserir o primeiro pacote no *buffer*. O tamanho utilizado para o *buffer* fez com que não houvesse redistribuição durante o processamento do problema, uma vez que o *buffer* nunca ficou totalmente ocupado. Neste cenário, a distribuição dos pacotes sempre ocorre no início do processamento.

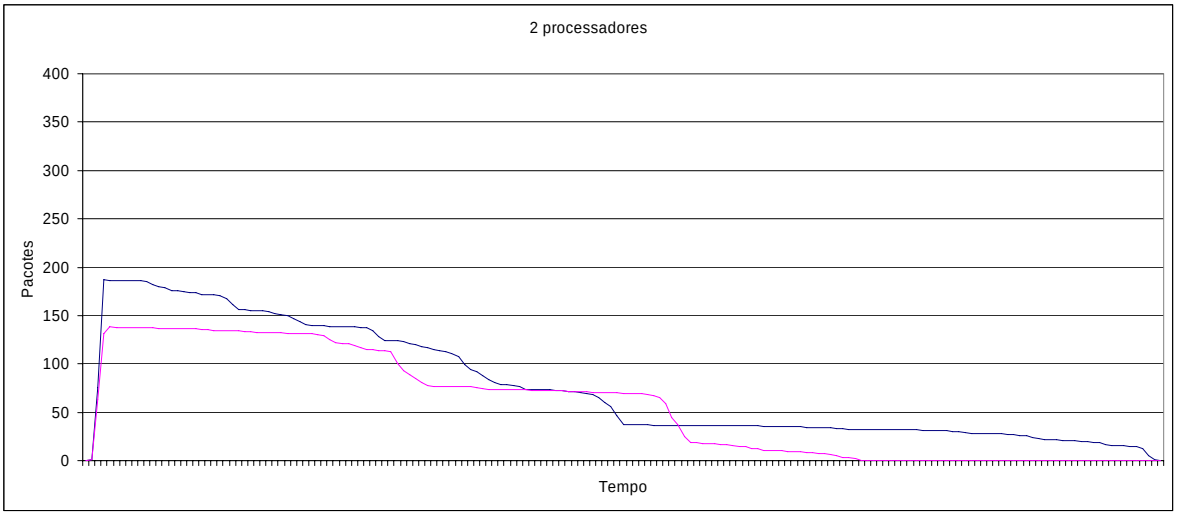

Figura 18 - Distribuição de carga com 2 processadores, controle estático do *buffer* e 100 de espalhamento.

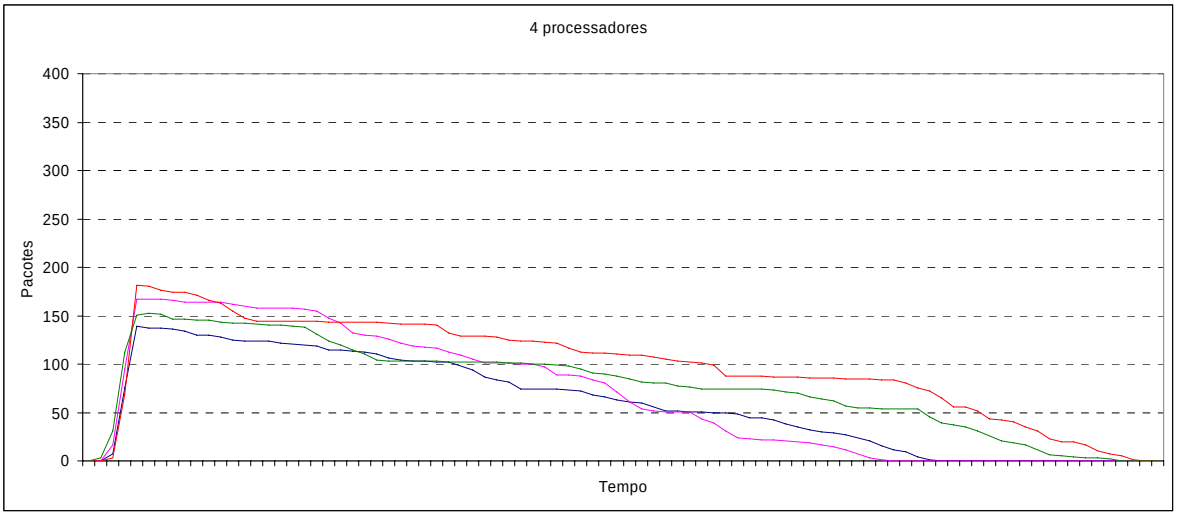

Figura 19 - Distribuição de carga com 4 processadores, controle estático do *buffer* e 100 de espalhamento.

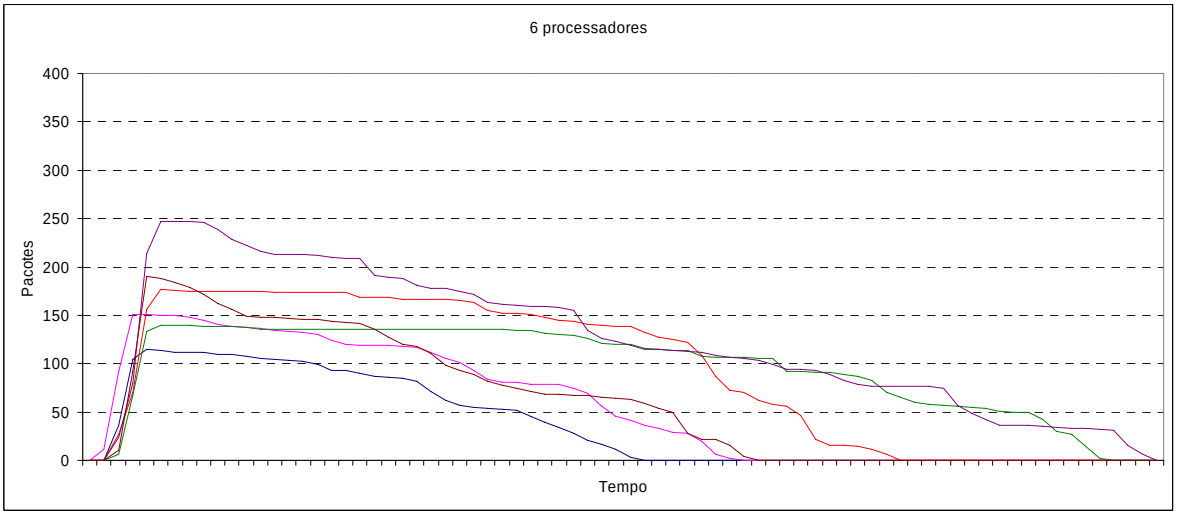

Figura 20 - Distribuição de carga com 6 processadores, controle estático do *buffer* e 100 de espalhamento.

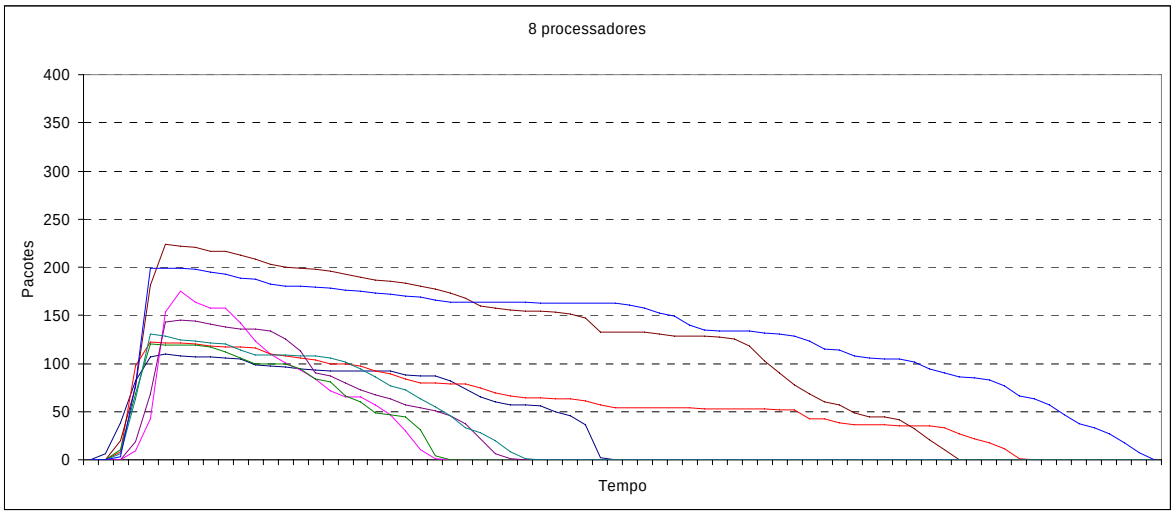

Figura 21 - Distribuição de carga com 8 processadores, controle estático do *buffer* e 100 de espalhamento.

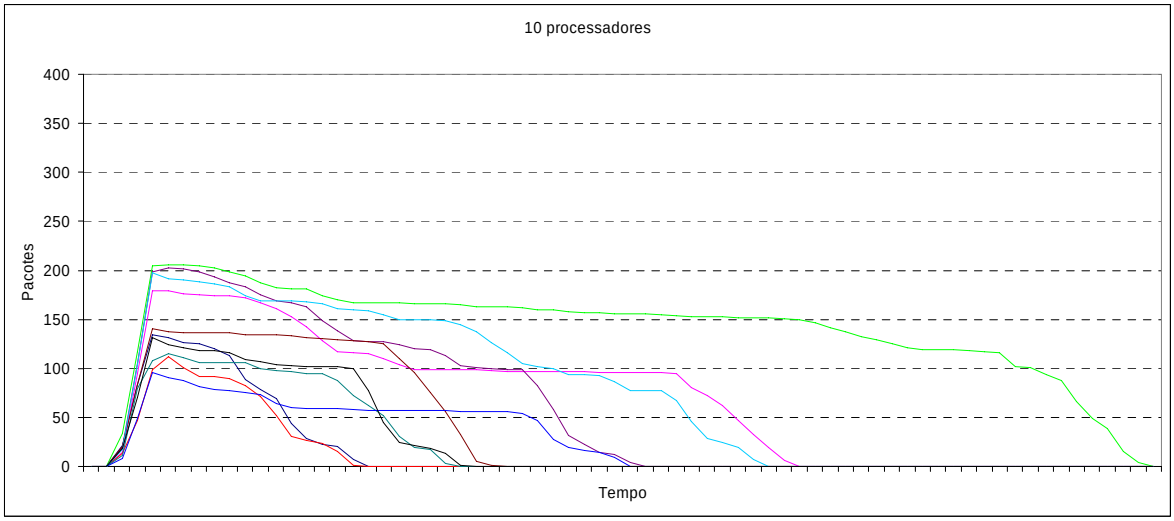

Figura 22 - Distribuição de carga com 10 processadores, controle estático do *buffer* e 100 de espalhamento.

No cenário seguinte, da Figura 23 à Figura 27, temos os gráficos das distribuições de pacotes os testes com 2, 4, 6, 8 e 10 processadores. Neste cenário, utilizamos o controle dinâmico para o tamanho do *buffer*. O fator de espalhamento não é utilizado, já que a distribuição de pacotes ocorre durante todo o processamento e é determinada automaticamente pelo algoritmo.

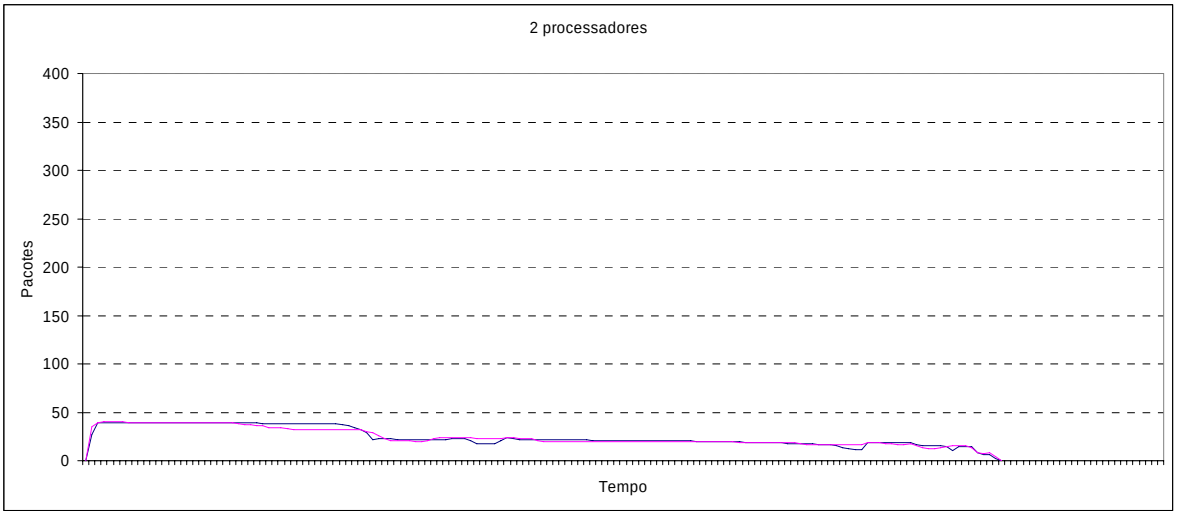

Figura 23 - Distribuição de carga com 2 processadores e controle dinâmico do *buffer*

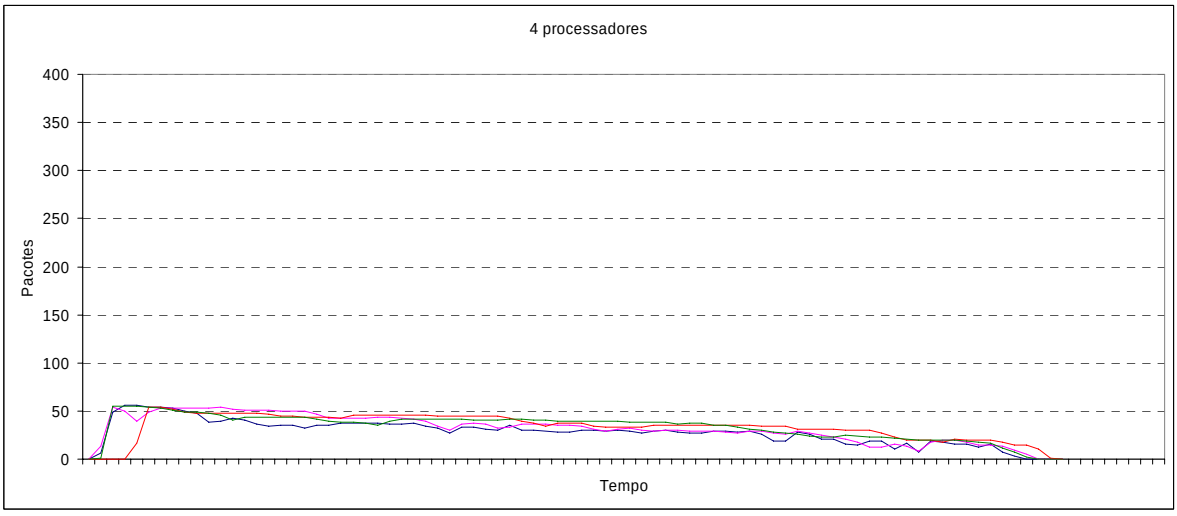

Figura 24 - Distribuição de carga com 4 processadores e controle dinâmico do *buffer*

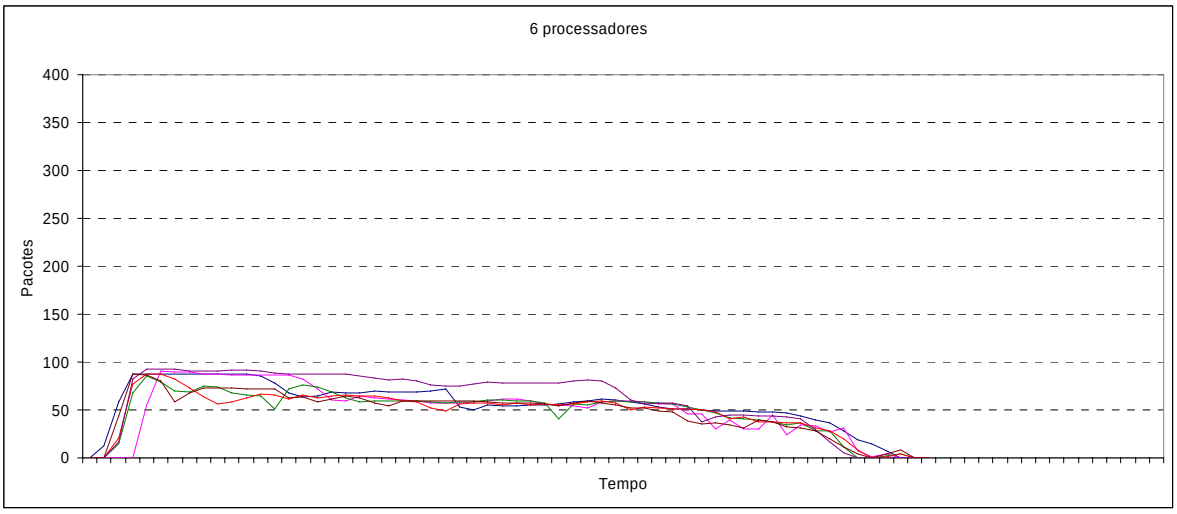

Figura 25 - Distribuição de carga com 6 processadores e controle dinâmico do *buffer*

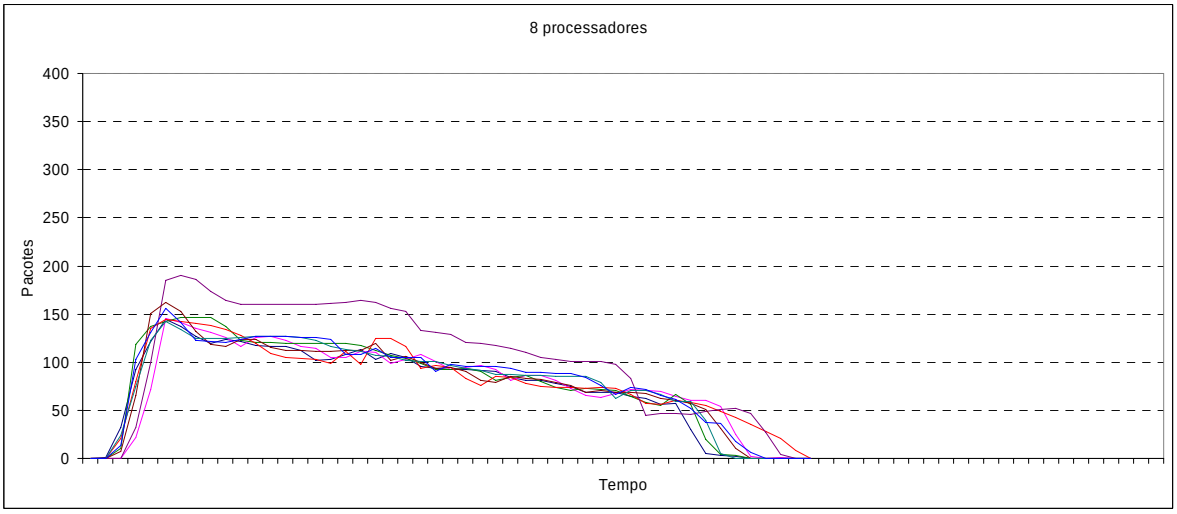

Figura 26 - Distribuição de carga com 8 processadores e controle dinâmico do *buffer*

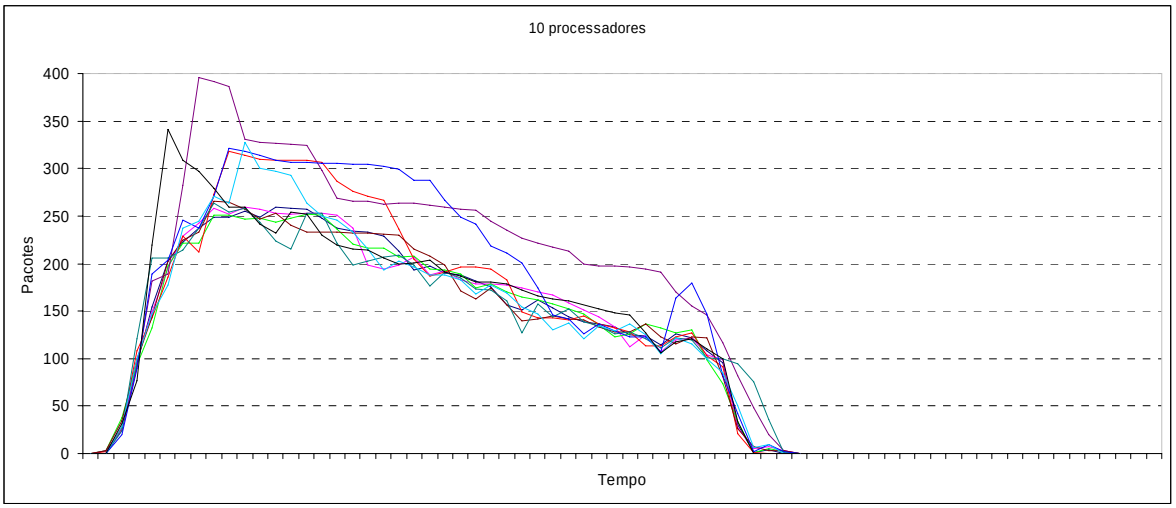

Figura 27 - Distribuição de carga com 10 processadores e controle dinâmico do *buffer*

Na Figura 28 seguir, temos o gráfico comparativo entre o tempo de resolução do problema com o controle dinâmico e estático. A finalização do problema é dada quando todo o espaço de busca é percorrido pelo algoritmo, considerando os descartes.

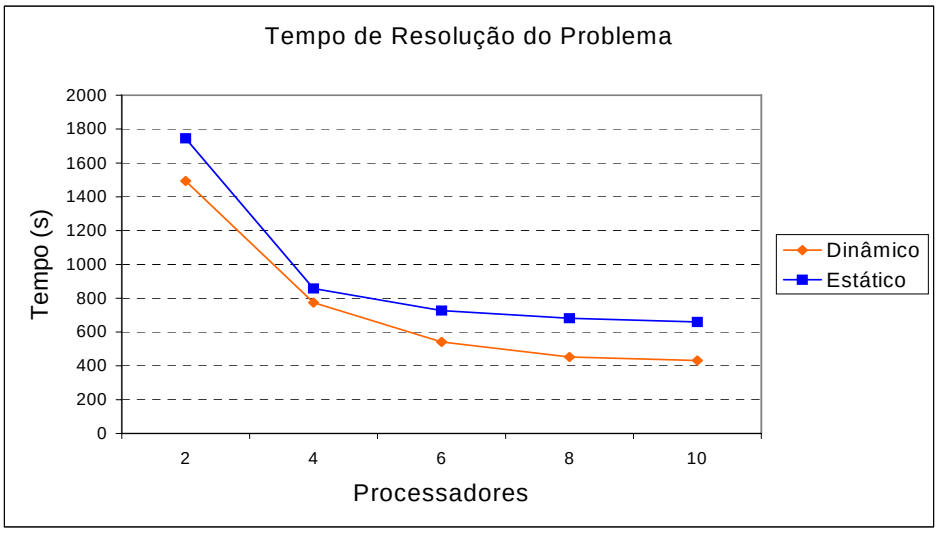

Figura 28 - Comparação do tempo de resolução do problema

Nas Figura 29 e Figura 30 temos o comparativo entre os tempos de processamento real, serial e ideal, com controle dinâmico e estático, respectivamente. Utilizamos as seguintes definições para a classificação dos tempos:

– Real – tempo efetivo medido desde o início do processamento até o retorno da resposta;

– Serial – somatório do tempo de processamento gasto por cada processador;

– Ideal – razão entre o tempo serial e o total de processadores.

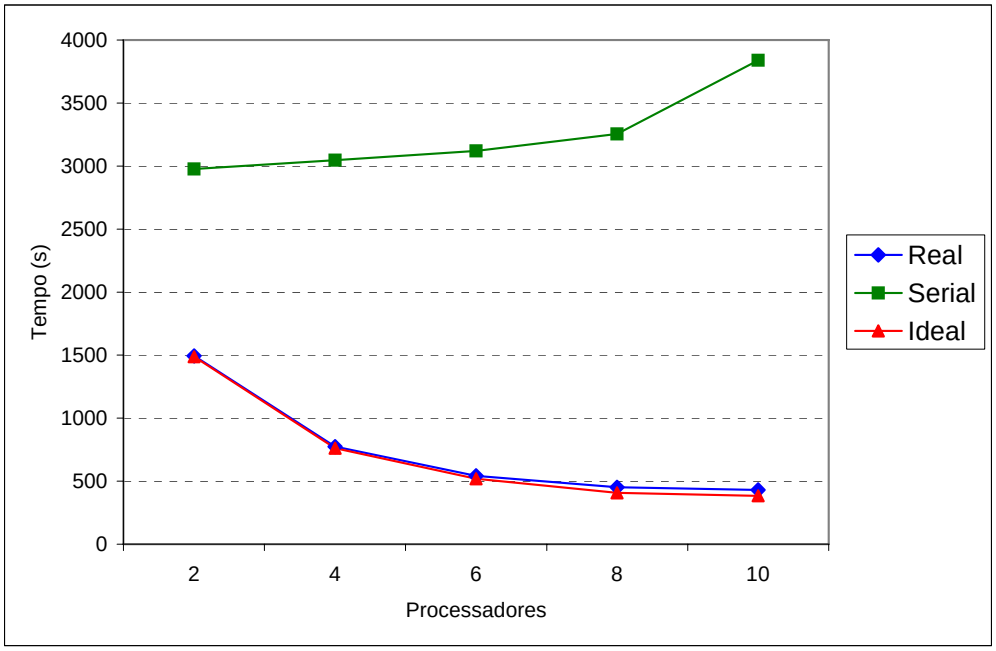

Figura 29 - Variação dos tempos de processamento real, serial e ideal com controle dinâmico do *buffer*

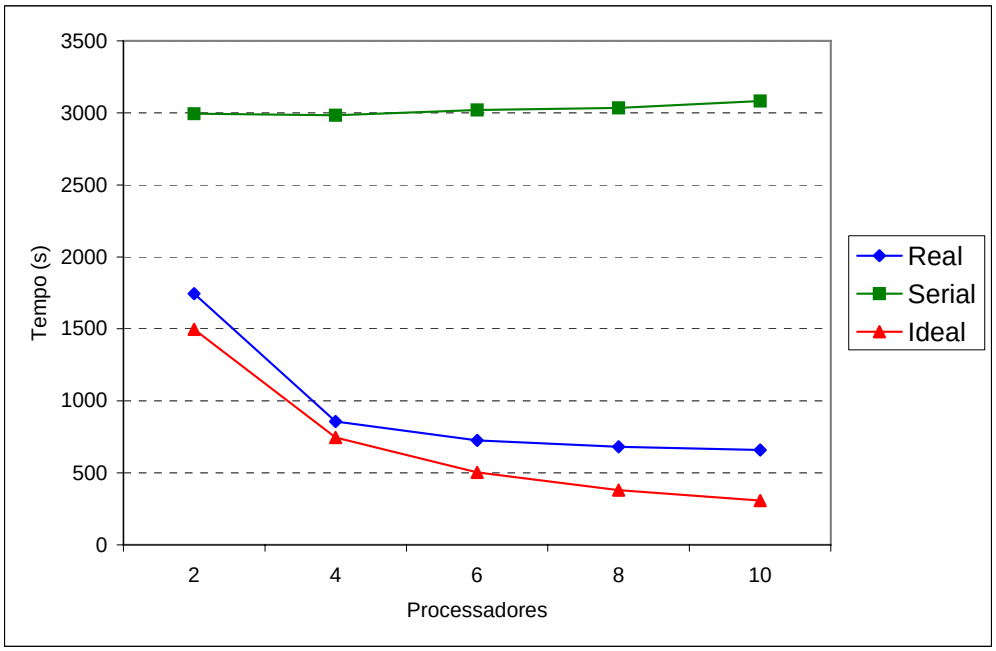

Figura 30 - Variação dos tempos de processamento real, serial e ideal com controle estático do *buffer*

No gráfico da Figura 31 temos o comparativo entre o *speed-up* obtido com o controle estático e dinâmico do *buffer*. O gráfico mostra ainda a linha do valor ideal para o *speed-up*.

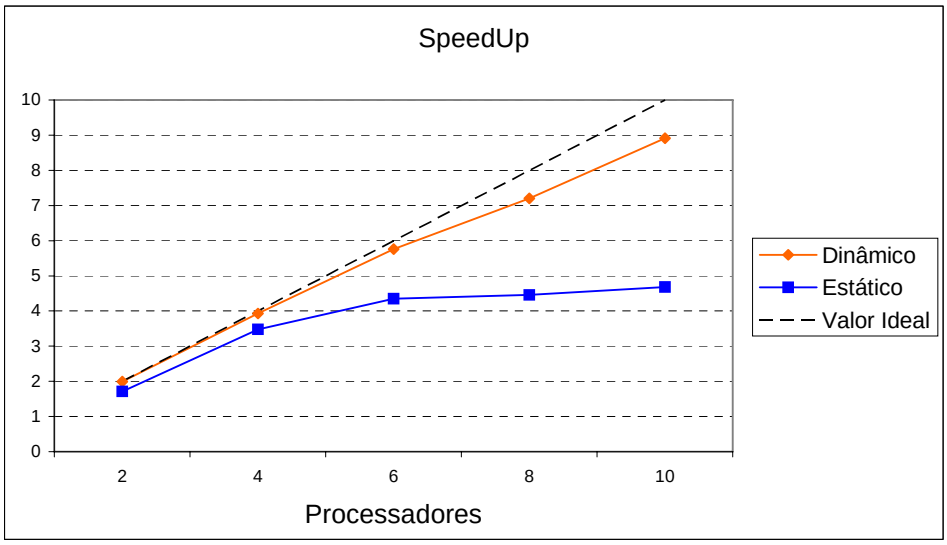

Figura 31 - *Speed-up* do algoritmo

A Tabela 2 mostra o cálculo do *speed-up* do algoritmo.

Dinâmico			
Máquinas	Real (s)	Serial (s)	Speed-Up
2	1493,35	2976,58	1,99
4	774,57	3046,37	3,93
6	541,56	3120,02	5,76
8	451,88	3255,18	7,20
10	430,74	3840,09	8,91
Estático			
2	1745,31	2993,37	1,71
4	857,19	2983,63	3,48
6	726,04	3020,81	4,35
8	681,52	3035,58	4,45
10	658,96	3083,65	4,68

Tabela 2 - Cálculo do *speed-up* do algoritmo

Para validar o resultado do algoritmo, utilizamos um *benchmark* conhecido. Optamos pela utilização do 13º *benchmark* de Patterson (Patterson13) [31], devido a sua grande utilização para avaliar a eficiência de heurísticas de escalonamento de projetos. A rede de atividades do *benchmark* é composta por 20 atividades, mais duas atividades fictícias no início e fim do projeto. Na Tabela 3 temos o comparativo entre os resultados obtidos para o Patterson13, e o projeto gerado pelo ProGen e utilizado no experimento. Os valores referentes ao Patterson13 foram obtidos utilizando 10 processadores e o controle automático do *buffer*.

	Patterson 13	Benchmark gerado pelo ProGen
Número de atividades	22	15
Resultado ótimo	20 unidades de tempo	59 unidades de tempo
Tempo para solução ótima	59min	7min
Tempo serial	9h 42min	1h 4min
Speed-up	9,8	8,9
Total de requisições de trabalho	6.859.604	766.038
Total de descartes	81.895.089	1.857.823

Tabela 3 – Comparação entre os resultados do problema gerado pelo ProGen e o Patterson13

Capítulo 6

DISCUSSÃO DO EXPERIMENTO

Nesta seção, analisaremos os resultados práticos obtidos, explicando os gráficos e tabelas do capítulo anterior.

6.1 – Controle estático vs controle dinâmico

O objetivo da análise da distribuição de pacotes de trabalho nos gráficos da Figura 18 até a Figura 27 é comparar a eficiência da estratégia de controle dinâmico do *buffer* versus o controle estático e o comportamento da distribuição a medida que aumentamos o número de processadores participantes.

Podemos perceber que nos gráficos de distribuição estática (da Figura 18 à Figura 22), só há distribuição no início do problema. Esta distribuição é proporcionada pelo fator de espalhamento, que determina a quantidade de pacotes de trabalho que cada processador deverá repassar ao IeC antes de utilizar o *buffer* interno. Uma vez alcançado no número de pacotes determinado pelo fator de espalhamento, o *buffer* interno começa a ser utilizado. Como o controle do *buffer* é estático seu tamanho permanece o mesmo durante todo o decorrer do processamento. Dessa forma, o fato de um ou mais processadores ficarem ociosos, os demais processadores continuam armazenando pacotes em seus *buffers* internos, da mesma forma que faziam no início do problema quando o número de pacotes era abundante. O resultado disso é que, após o espalhamento inicial, não há redistribuição de pacotes. Os pacotes distribuídos inicialmente são consumidos e alguns processadores finalizam seu processamento bem antes que outros, dependendo da complexidade dos pacotes que cada um recebeu. Podemos notar que, independente do número de

processadores, sempre temos uma grande variação entre o tempo de término de processamento do primeiro e último processadores. Como esse intervalo de tempo significa desperdício de tempo, temos a redução da eficiência do algoritmo, já que só podemos afirmar que a solução ótima foi encontrada quando todos os processadores finalizarem seus trabalhos.

Por outro lado, da Figura 23 à Figura 27, temos os gráficos de distribuição dinâmica do algoritmo. Nesta situação não prefixamos o tamanho do *buffer* nem há a necessidade de utilizar o fator de espalhamento. No início do problema, o *buffer* de cada processador possui tamanho zero, o que significa que os trabalhos que a primeira requisição de cada processador gerar não serão inseridos no *buffer* e sim enviados ao IeC. À medida que o processamento continua o *buffer* tem seu tamanho constantemente aumentado, ou reduzido, em função da distribuição de pacotes por toda a rede. Dessa forma, quando todos os processadores possuem uma carga equivalente de trabalho, o tamanho *buffer* é aumentado, de modo a não gerar novas requisições ao IeC, já que não haverá nenhum processador disponível para respondê-la. Esta situação equivale aos períodos do gráfico onde temos uma curva descendente. À medida que os pacotes são consumidos com mais velocidade por alguns processadores, a distribuição de pacotes começa a ficar desigual. Neste momento, o *buffer* passa a ser decrementado até ocorrer a situação onde existem mais pacotes no *buffer* do que o permitido. É neste momento que os pacotes excedentes mais antigos são retirados do *buffer* e novamente repassados ao IeC, que se encarrega de entrega-los ao processador com menor quantidade de trabalho pendente. Esta situação é retratada nos gráficos pela curvas ascendentes. Pode-se notar nos gráficos que essas curvas ascendentes não apresentam grandes picos, o que é muito bom pois indica que o re-equilíbrio de distribuição é rapidamente alcançado. Manter todos os processadores ocupados a maior parte do tempo garante uma boa eficiência do paralelismo da busca pela solução.

Comparando os resultados da distribuição estática com a dinâmica, podemos perceber que, mesmo com a variação do número de processadores utilizados, o algoritmo conseguiu manter a distribuição de carga equivalente entre os processadores com o

controle dinâmico da fila, fazendo com que todos os processadores terminassem aproximadamente juntos, independente do número de processadores utilizados. O mesmo não acontece com a distribuição estática pois o tamanho do *buffer* permanece constante. Desta forma, mesmo que existam máquinas ociosas, o *buffer* retém trabalho para si ao invés de retransmiti-los.

6.2 – Ganhos e Perdas

Na Figura 28 temos o gráfico de resolução do problema em função do número de processadores. Em todos os casos, o controle dinâmico da distribuição supera os resultados obtidos com o controle estático. Este resultado está diretamente ligado a melhor distribuição de trabalho obtida com o controle dinâmico, que possibilitou que o tempo de ociosidade dos processadores fosse minimizado. Mesmo assim, podemos perceber que o tempo de solução do problema tende a um valor mínimo. Os resultados mostrados nas Figura 29 e Figura 30 ajudam a explicar este comportamento.

Chamamos de tempo real o tempo efetivo medido desde o início do processamento até o retorno da resposta, tempo serial como o somatório do tempo de processamento gasto por cada processador e tempo ideal como sendo a razão entre o tempo serial e o total de processadores. O tempo ideal indica qual seria o melhor valor para o tempo real, ou seja, melhor ganho de paralelismo. Na Figura 29 podemos verificar que o tempo serial aumenta a medida que o número de processadores aumenta. O tempo serial não é constante ao variarmos o número de processadores, pois, além de considerar o tempo gasto na resolução do problema propriamente dito, leva ainda em consideração o tempo gasto com a geração de pacotes de trabalho e o tratamento das respostas destes pacotes. Todo este gerenciamento de pacotes consome tempo e este tempo aumenta em função do número de pacotes trafegando na rede. Como cada processador participante pode gerar pacotes de trabalho, quanto mais processadores na rede, maior a quantidade de pacotes e maior o tempo gasto com o gerenciamento deles. Além disso, pelos gráficos de distribuição

podemos verificar que o controle dinâmico gera muito mais pacotes de trabalho que o controle estático, gerando a diferença de comportamento em relação ao tempo serial do controle estático (Figura 30).

Enquanto que, para o controle dinâmico, o fator limitante para o tempo do problema é a relação entre quantidade de processadores e quantidade de requisições geradas (a geração de requisições de trabalho é grande), para o controle estático, o limite do tempo de execução é dado pela fraca distribuição de trabalho entre os processadores (maquinas ficam ociosas no decorrer do processamento). Dessa forma, para um mesmo problema, irá existir um limite para o número de processadores de tal forma que, aumentando este número, o ganho de tempo será desprezível. Este fato está ligado a capacidade de paralelismo da árvore de solução do problema. Uma árvore com menos ramos que outra irá oferecer uma menor capacidade de paralelismo.

Com os valores de tempo real e serial, podemos comparar o ganho do processamento paralelo através do cálculo da aceleração obtida (*speed-up*). O *speed-up* é dado pela razão entre o tempo serial e o tempo real. É importante destacar que o tempo serial que utilizamos **não** é o tempo gasto considerando um algoritmo puramente serial executando o problema e sim o somatório do tempo de CPU utilizado por cada processador. Com isso, obtemos o gráfico da Figura 31. No gráfico, indicamos pela linha tracejada o valor ideal para o *speed-up*. Podemos perceber que a partir de 6 processadores o *speed-up* do controle dinâmico começa a se distanciar da linha ideal. Se verificarmos no gráfico da Figura 29, este ponto coincide com o disparo do tempo serial. Mesmo assim, o *speed-up* obtido com 10 processadores ficou bem próximo do valor ideal. O controle estático apresenta um valor bem inferior ao controle dinâmico, reforçando a baixa escalabilidade desta abordagem. Na Tabela 2 temos os valores medidos de tempo serial e real utilizados para o cálculo do *speed-up*.

6.3 – Patterson 13

Para finalizar, apresentamos, na Tabela 3, os resultados obtidos ao executar um *benchmark* conhecido – Patterson13. Este problema apresenta uma complexidade muito maior do que o problema utilizado nos testes anteriores, gerando quase 9 vezes mais requisições. Isso significa uma árvore de busca com um potencial de paralelismo muito maior. Enquanto que o *speed-up* do problema de teste foi de 8,9 com 10 processadores, o *speed-up* obtido com o Patterson13 chegou a 9,8 – apenas dois décimos abaixo do valor ideal. Isto indica que, se fossem utilizados mais processadores, o tempo de resolução do problema poderia cair muito mais.

Capítulo 7

CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo traz as conclusões finais e o fechamento do trabalho.

7.1 – Conclusões

Este trabalho apresentou a utilização de um algoritmo para aproveitar o processamento distribuído em uma rede heterogênea e os mecanismos necessários para tornar este ambiente eficiente.

O experimento mostrou a escalabilidade do algoritmo, que com a adição de novos processadores levam a redução do tempo total de processamento. A partir desses resultados, podemos esperar que a inclusão de novos processadores vá reduzir o tempo total de processamento requerido para solucionar problemas ainda maiores.

Pelos resultados obtidos podemos concluir que a árvore de solução do problema tem influência direta na performance do algoritmo. O grau de paralelismo obtido está diretamente ligado à largura da árvore de solução. A árvore que possui poucos nós irmãos irá ter um menor fator de paralelismo, tendendo a uma execução serial. A altura da árvore de solução também influencia a performance. Uma árvore com poucos níveis é resolvida muito rapidamente, não compensando o tempo gasto na gerência e envio dos pacotes de trabalho.

O algoritmo irá mostrar melhor performance com problemas maiores, com árvores de solução largas – favorecendo o paralelismo – e profundas – gerando uma quantidade de trabalho que não possa ser trivialmente resolvida por processadores independentes.

Finalmente, o método proposto não está restrito ao RCPSP, já que ambas as técnicas utilizadas neste trabalho, o *buffer* interno e o modelo *master-worker* hierárquico podem ser aplicadas a qualquer problema que utilize uma abordagem *branch and bound.* Mais ainda, as técnicas apresentadas permitem utilizar uma arquitetura P2P de modo eficiente a ponto de emular uma máquina paralela.

7.2 – Trabalhos Futuros

Uma grande possibilidade de melhoria do algoritmo é a utilização de outras funções de estimativa de limite inferior para o RCPSP. Utilizamos, no trabalho, a função mais simples existente que, apesar de ser facilmente calculada, não apresenta estimativas muito próximas do valor real. A utilização de funções de estimativas mais eficientes iria melhorar muito o descarte de soluções inviáveis, aumento a performance do algoritmo.

Outra modificação possível, ainda para melhorar o descarte de soluções inviáveis, seria realimentar o limite superior do problema assim que um valor menor for encontrado. Na implementação atual do algoritmo, o limite superior é calculado no início do problema e não é alterado mesmo que algum processador encontre um valor menor.

Finalmente, a abordagem proposta poderia ser utilizada para calcular soluções aproximadas para problemas ainda maiores, utilizando heurísticas que sejam complexas o suficiente para necessitarem de processamento paralelo para reduzir o tempo de execução como, por exemplo, a utilização de algoritmos genéticos para a busca de soluções aproximadas para o RCPSP com grandes quantidades de atividades. Neste caso, a geração das populações poderia ser feita em paralelo nos diversos nós de processamento.

ANEXO

```
SAMPLEFILE BASEDATA
PROJEKTS
  NrOfPro            :  1      & number of projects
  MinJob             :  13     & minimal number of jobs per project
  MaxJob             :  13     & maximal number of jobs per project
  MaxRelDate         :  0      & maximal release date
  DueDateFactor      : 0.0     & maximal due date
MODES
  MinMode            :  1      & minimal number of modes
  MaxMode            :  1      & maximal number of modes
  MinDur             :  1      & minimal duration
  MaxDur             :  15     & maximal duration
NETWORK
  MinOutSource       :  1      & minimal number of start activities per project
  MaxOutSource       :  1      & maximal number of start activities per project
  MaxOut             :  3      & maximal number of successor per activity
  MinInSink          :  1      & minimal number of end activities per project
  MaxInSink          :  1      & maximal number of end activities per project
  MaxIn              :  3      & maximal number of predecessors per activity
  Complexity         :  1.5    & complexity of network
RESSOURCEREQUEST/AVAILABILITY
  Rmin               :  4      & minimal number of renewable resources
  Rmax               :  4      & maximal number of renewable resources
  RminDemand         :  1      & minimal (per period) demand
  RmaxDemand         : 10      & maximal (per period) demand
  RRMin              :  1      & minimal number of resources requested
  RRMax              :  4      & maximal number of resources requested
  RRF                :  0.5    & resource factor
  RRS                :  0.2    & resource strength
  Number R-Func.     :  2
    p1               :  0.0    & probability to choose (duration) constant function
    p2               :  1.0    & probability to choose monotonically decreaseing function
  Nmin               :  0
  Nmax               :  0
  NminDemand         :  0
  NmaxDemand         :  0
  NRMin              :  0
  NRMax              :  0
  NRF                :  0.0
  NRS                :  0.0
  Number N-Func.     :  2
    p1               :  0.0
    p2               :  1.0
  Dmin               :  0
  Dmax               :  0
  DminDemand         :  0
  DmaxDemand         :  0
  DRMin              :  0
  DRMax              :  0
  DRF                :  0.0
  DRST               :  0.0
  DRSP               :  0.0
  Number D-Func.     :  2
    p1               :  1.0
    p2               :  0.0
LIMIT OF ITERATIONS
  Tolerance Network  :  0.05  & tolerated network deviation
  Tolerance RF       :  0.05  & tolerated resource factor deviation
  MaxTrials          :  200   & maximal number of trials randomly gen.
```

Figura 32 - Arquivo de configuração do ProGen utilizado para o projeto do experimento

```
******************************************************************************
file with basedata            : EXPL_15.BAS
initial value random generator: 90
******************************************************************************
projects                      :  1
jobs (incl. supersource/sink ):  15
horizon                       :  75
RESOURCES
  - renewable                 :  5   R
  - nonrenewable              :  0   N
  - doubly constrained        :  0   D
******************************************************************************
PROJECT INFORMATION:
pronr.  #jobs rel.date duedate tardcost  MPM-Time
    1     13      0       37        0       37
******************************************************************************
PRECEDENCE RELATIONS:
jobnr.    #modes  #successors   successors
   1        1          1           2
   2        1          3           3   4   5
   3        1          3           7   9  12
   4        1          2          10  12
   5        1          3           6   9  12
   6        1          2           7  10
   7        1          2           8  13
   8        1          1          14
   9        1          1          10
  10        1          1          11
  11        1          1          13
  12        1          1          13
  13        1          1          14
  14        1          1          15
  15        1          0
******************************************************************************
REQUESTS/DURATIONS:
jobnr. mode  duration: min  expected  max       R 1  R 2  R 3  R 4  R 5
------------------------------------------------------------------------------
  1     1               0      0      0          0    0    0    0    0
  2     1               1      5      7          0   12    7    2    2
  3     1               1      2      3          0    7    9    6    4
  4     1               3      8     12          8    4    9    0    4
  5     1               1      2      3          9    1    0    7    5
  6     1               2      7      9          0   10   10    7    0
  7     1               2      7     11          6    8    0    5   10
  8     1               3      5      9          0   11    7    5    3
  9     1               5      9     15          7    0    5    4    1
 10     1               4      8      9          0    6    7    7    6
 11     1               4      5      9          5    6    0    7    7
 12     1               2      9     14          0    7    7    8    4
 13     1               1      2      3          3    7    6    3    0
 14     1               5      6     11          8    5    0    7    3
 15     1               0      0      0          0    0    0    0    0
******************************************************************************
RESOURCEAVAILABILITIES:
  R 1  R 2  R 3  R 4  R 5
   11   14   13   10   12
******************************************************************************
```

Figura 33 - Arquivo de saída do Progen do problema utilizado no experimento

REFERÊNCIAS

[1] **BOTTCHER J., DREXL A., KOLISCH R., SALEWSKI F. – "Project scheduling under partially renewable resource constraints.", relatório técnico, Universitat Kiel, 1996.**

[2] **MARTINS S. – "Gerência de projeto: meta-heurísticas para otimização do escalonamento de atividades na exploração e produção de petróleo", tese de doutorado, Universidade Federal Fluminense, 2000.**

[3] **KOLISCH R., PADMAN R., – "An integrated Survey of Project Scheduling", relatório técnico, Universitat Kiel, novembro 1997.**

[4] **LENSTRA J., RINNOOY K. – "Computational complexity of discrete optimization problems", Annals of Discrete Mathematics, 1979.**

[5] **GRAHAM R., LAWLER E., LENSTRA J., RINNOOY K. – "Optimization and approximation in deterministic sequencing and scheduling theory: a survey", Annals of Discrete Mathematics, 1979.**

[6] **PINEDO M. – "Scheduling: theory, algorithms and systems", editora Prentice-Hall, 1995.**

[7] **ICMELI O., ROM W. – "Analysis of the characteristics of projects in diverse industries", Journal of Operations Management, 1998.**

[8] **GAREY M. R., JOHNSON D. S. – "Computer and intractability – A guide to the theory of NP-completeness", W.H. Freeman, San Francisco, 1979.**

[9] **NEUMANN K., SCHWINDT C. – "Projects with minimal and maximal time lags: construction of activity-on-node networks and applications", Universitat Karlsruhe, Relatório Técnico, wior-447, 36p, 1995.**

[10] **KOLISCH R., HEMPEL K. – "Finite scheduling capabilities of commercial project management systems", manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel, Alemanha, no. 397, 15p, 1996.**

[11] **AIDA K., NATSUME W., FUTAKATA Y. – "Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm", 3rd International Symposium on Cluster Computing and the Grid, Japão, 2003.**

[12] **BLACK P. – National Institute of Standards and Technology (NIST), www.nist.gov.**

[13] SIMPSON W., PATTERSON J. – "A multiple-tree search procedure for the resource-constrained project scheduling problem", European Journal of Operational Research, 1996.

[14] MINGOZZI A., MANIEZZO V., RICCIARDELLI S., BIANCO L.– "An Exact Algorithm for the Resource Constrained Project Scheduling Problem Based on a New Mathematical Formulation", Management Science 44, 714-729, 1998.

[15] KRAUTER K. – "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing", Software - Practice and Experience, 2001.

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

[17] PEREIRA, P. – "IEC – Uma Arquitetura XML para Computação Colaborativa P2P", dissertação de mestrado, NCE/IM, Universidade Federal do Rio de Janeiro, Janeiro 2005.

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

[19] MILOJICIC D., KALOGERAKI V., LUKOSE R., NAGARAJA K., PRUYNE J., RICHARD B., ROLLINS S., XU, Z. – "Peer-to-peer computing.", Technical Report HPL-2002-57, HP Lab, 2002. http://citeseer.ist.psu.edu/milojicic02peertopeer.html.

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

[21] KOLISCH R., SPRECHER A., DREXL A. – "Characterization and generation of a general class of resource-constrained project scheduling problems", Management Science 41, 1693±1703, 1995.

[22] KOLISH R., HARTMANN S. – "Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis", Advances in Project Scheduling, Elsevier, pp 113-134, 1996.

[23] BAAR T., BRUCKER P., KNUST S. – "Tabu-search algorithms for the resource-constrained project scheduling problem", Technical Report, Osnabrücker Schriften zue Mathematik, Fachbereich Mathematik/Informatik, Universität Osnabrücker, 1997.

[24] BOULEIMEN K., LECOCQ H. – "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem", Technical Report, Service de Robotique et Automatisation, Université de Liège, 1998.

[25] EGLESE R. – "Simulated annealing: A tool for operational research", European Journal of Operational Research, 46: 271-281, 1990.

[26] HARTMANN S. – "A competitive genetic algorithm for resource-constrained project scheduling", Naval Research Logistics, 45: 733-750, 1998.

[27] MERKLE D., MIDDENDORF M., SCHMECK H. – "Ant colony optimization for resource-constrained project scheduling", IEEE Transactions on Evolutionary Computation, 6:333-346, 2002.

[28] CARRUTHERS J., BATTERSBY A. – "Advances in critical path methods", Operations Research Quarterly, 17(4): 359-380, 1966.

[29] PATTERSON J., HUBER W. – "A horizon-varying, zero-one approach to project scheduling", Management Science, 20: 990-998, 1974.

[30] DEMEULEMEESTER E., HERROELEN W., ELMAGHRABY S. – "Optimal procedures for the discrete time/cost trade-off problem in project networks", European Journal of Operational Research, 88: 50-68, 1996.

[31] PATTERSON J. – "A comparison of exact approaches for solving the multiple constrained resource project scheduling problem", Management Science, 30, 854 – 867, 1984.

[32] Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, Fevereiro 2004.