PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

OTIMIZANDO O FLUXO DE TAREFAS EM SISTEMAS DISTRIBUÍDOS DE IMPRESSÃO: UM ALGORITMO DE ESCALONAMENTO DINÂMICO NÃO PREEMPTIVO BASEADO EM MECANISMO DE PREVISÃO

Rafael Nemetz

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Luiz Gustavo Leão Fernandes

Porto Alegre
2011

Dados Internacionais de Catalogação na Publicação (CIP)

N433o Nemetz, Rafael
Otimizando o fluxo de tarefas em sistemas distribuídos de impressão : um algoritmo de escalonamento dinâmico não preemptivo baseado em mecanismo de previsão / Rafael Nemetz. – Porto Alegre, 2011.
79 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Luiz Gustavo Leão Fernandes.

1. Informática. 2. Documentos – Personalização. 3. Impressão Digital. 4. Algoritmos. I. Fernandes, Luiz Gustavo Leão. II. Título.

CDD 005.1

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "**Otimizando o Fluxo de Tarefas em Sistemas Distribuídos de Impressão: Um Algoritmo de Escalonamento Dinâmico Não Preemptivo Baseado em Mecanismo de Previsão**", apresentada por Rafael Nemetz, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 31/03/2011 pela Comissão Examinadora:

Prof. Dr. Luiz Gustavo Leão Fernandes– Orientador — PPGCC/PUCRS

Prof. Dr. César Augusto Missio Marcon - — PPGCC/PUCRS

Prof. Dr. Eduardo Augusto Bezerra – — UFSC

Homologada em 28./06../..11....., conforme Ata No. 11/11..... pela Comissão Coordenadora.

Prof. Dr. Fernando Luís Dotti
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

OTIMIZANDO O FLUXO DE TAREFAS EM SISTEMAS DISTRIBUÍDOS DE IMPRESSÃO: UM ALGORITMO DE ESCALONAMENTO DINÂMICO NÃO PREEMPTIVO BASEADO EM MECANISMO DE PREVISÃO

RESUMO

Nos últimos anos, com a modernização e informatização das casas de impressão, documentos digitais como o PDF tornaram-se formatos padrão para a descrição das tarefas a serem impressas nestes ambientes. Estes documentos, descritos em uma linguagem de alto nível de abstração, não são diretamente reconhecidos pelas impressoras e necessitam passar por etapas de conversão antecedentes à impressão, chamadas de pré-processamento de documentos. Estas etapas, no entanto, exigem demasiado poder computacional e tornaram-se processos limitantes da produção das casas de impressão. O processamento paralelo passou, então, a ser empregado a fim de aumentar a velocidade deste processo. Este trabalho propõe estratégias baseadas em busca de recursos e previsão de tempo dedicada com a finalidade de automatizar o escalonamento de tarefas neste ambiente de processamento distribuído, atribuindo tarefas de forma otimizada aos diferentes *clusters* de computadores responsáveis pelo processamento. Assim, um aumento do fluxo global de tarefas e do desempenho e confiabilidade no atendimento das tarefas de todo o processo de impressão pode ser atingido. Após sugerir uma metodologia para mecanismos dedicados de previsão de tempo de execução, são propostos e testados novos algoritmos de escalonamento dinâmico não-preemptivos para sistemas heterogêneos.

Palavras-chave: rasterização, PDF, impressão, escalonamento, busca de recursos, previsão dedicada, sistemas heterogêneos, processamento de documentos, casas de impressão.

OPTIMIZING THE TASKS FLOW IN DISTRIBUTED PRINT SHOPS SYSTEMS: A DYNAMIC NON-PREEMPTIVE SCHEDULING ALGORITHM BASED ON PREDICTION MECHANISMS

ABSTRACT

In recent years, with the modernization and automation of print shops, digital documents as PDF have become the standard format for describing printing tasks in these environments. These digital documents are described in a high-level abstraction language and cannot be directly recognized by printers. Thus, digital documents need to go through conversion steps prior to printing. However, these steps require high computing power and became bottleneck processes to the print shops production. Parallel processing has then started to be employed in order to speed up this process. This work proposes strategies based on a resource discovery system and a dedicated forecasting mechanism in order to automate the scheduling of tasks in distributed printing environments. After suggesting a dedicated methodology for time executions forecasting mechanisms, new scheduling algorithms for dynamic non-preemptive heterogeneous systems are proposed and tested.

Keywords: ripping, PDF, printing, scheduling, resource discovery, dedicated prediction, heterogeneous systems, document processing, print shops.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BMP	*Bitmap Image File*
DHT	*Distributed Hash Table*
DPI	*Dots Per Inch*
EDF	*Earliest Deadline First*
FFD	*First Fit Description*
FIFO	*First In First Out*
GAR	*Grid Archive*
HTTP	*HyperText Transfer Protocol*
IP	*Internet Protocol*
JDF	*Job Definition Format*
JMF	*Job Messaging Format*
JPG	*Joint Photographic Experts Group*
MIMD	*Multiple Instruction Multiple Data*
MIPS	*Million Instructions per Second*
MIS	*Management Information System*
PDL	*Page Description Langauge*
PDF	*Portable Document Format*
PNG	*Portable Network Graphics*
PPML	*Personalized Print Markup Language*
PS	*PostScript*
RAM	*Random Access Memory*
RIP	*Raster Image Processor*
SIMD	*Single Instruction Multiple Data*
TAFT	*Time-Aware Fault-Tolerant*
TCP	*Transmission Control Protocol*
TIFF	*Tagged Image File Format*
VDP	*Variable Data Printing*
WSDD	*Web Service Data Definition*
XML	*Extensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

A tecnologia das impressoras industriais, desde as primeiras impressoras matriciais nascidas na década de 80, que imprimiam menos de 30 caracteres por segundo [1], cresceu com grande velocidade. Hoje, à *laser*, as impressoras são capazes de imprimir grande quantidade de páginas com alta resolução em curtos espaços de tempo (menos que uma página por segundo) [2]. Acompanhando este processo, cresceu também a demanda pela publicação impressa. Folhetos de propaganda, *outdoors*, álbuns fotográficos, revistas e livros podem ser impressos em grande escala utilizando recursos avançados de formatação, cor e transparência. Tirando proveito desta nova tecnologia e do avanço da computação, diversos formatos digitais passaram a ser criados com o intuito de facilitar a descrição do conteúdo destes documentos. Os formatos possibilitam a descrição, em alto nível, dos objetos de texto e imagem presentes nas páginas dos documentos.

Os formatos digitais abrem possibilidades para a utilização de técnicas avançadas, como as que envolvem os conceitos de VDP (*Variable Data Printing*), por exemplo. Estas técnicas permitem que um documento possua porções fixas e porções variáveis, dando a ele um certo grau de personalização [2]. Formatos amplamente difundidos, como o PS (*PostScript*) [3] e o PDF (*Portable Document Format*) [4], são exemplos de documentos nestes moldes. O alto nível de abstração destes formatos, que permite que os objetos dos documentos sejam expressos e modificados através de parâmetros, como em uma linguagem de programação, impede que as impressoras reconheçam estes documentos diretamente para a impressão. Então, algumas etapas de pré-processamento são necessárias aos documentos até que eles sejam convertidos em imagens no formato *bitmap* (mapa de bits) ou outros formatos de imagem que sejam diretamente reconhecidos pelas impressoras.

A conversão de documentos digitais descritos em linguagens de alto nível para arquivos de imagem recebe o nome de rasterização ou *ripping* (*Raster Image Processing*) e este processo demanda alto poder computacional. Tal fator acaba tornando-se um redutor na produção das casas de impressão, por ser um processo mais lento que os demais.

1.1 Motivação e Objetivos

O avanço da computação permitiu a aceleração de aplicações, como processamento de imagens, processamento de vídeo e resolução de sistemas lineares. Entre diversas áreas, principalmente nas áreas de Engenharia, Meteorologia e Biologia, estas aplicações passaram a se tornar essenciais na pesquisa e no desenvolvimento. Como são aplicações que demandam alto poder computacional, a partir da década de 80, o conceito de *cluster* [5] começou a se popularizar e os *clusters* passaram a ser utilizados como forma de acelerá-las. A ideia do *cluster* é de que se tenha uma rede local na qual estão conectadas máquinas de baixo custo (máquinas disponíveis no mercado para o público em geral). Dentre estas máquinas, o processamento é distribuído, permitindo um alcance de alto nível de paralelismo e considerável ganho de desempenho.

A utilização de *clusters* vem sendo empregada pelas casas de impressão nas etapas que envolvem alto custo computacional [6], como as etapas de renderização e de rasterização. Geralmente, existe mais de um destes aglomerados de computadores responsáveis pelo processamento e o escalonamento das tarefas ainda é feito de forma manual. Através da análise das características das tarefas, a automatização e a otimização do escalonamento e da gerência das filas de documentos que esperam pela rasterização podem trazer um ganho considerável de desempenho. Desta forma, um aumento de velocidade de todo o sistema pode ser alcançado, trazendo benefícios para todo o fluxo de tarefas em um sistema de impressão.

O objetivo deste trabalho é buscar um escalonador global responsável por distribuir as tarefas de documentos PDF (formato digital mais difundido e utilizado pelas casas de impressão da atualidade) para serem rasterizados pelos diferentes *clusters* que possam estar presentes em um ambiente de impressão distribuído. Após apresentados os trabalhos anteriores e uma investigação a respeito do cenário atual das casas de impressão, serão apresentadas a solução proposta e as etapas nela envolvidas. Serão apresentadas, então, as aplicações desenvolvidas, os resultados e conclusões obtidos e sugestões para trabalhos futuros.

1.2 Contribuições

O presente volume descreve um sistema construído a partir de pesquisa, testes e análise de resultados. De cada um dos módulos propostos, diferentes contribuições científicas podem ser extraídas. Contudo, três contribuições maiores podem claramente ser apontadas:

1. Sugestão de uma metodologia de utilização geral para realização de estimativas de tempo de processamento para tarefas iguais operando sobre diferentes conjuntos de dados que contenham um valor de custo previamente atribuído;

2. Um sistema de previsão de tempo de processamento específico para rasterização de documentos PDF;

3. Criação e testes de novos algoritmos de escalonamento dinâmico não-preemptivos orientados a *Deadlines* para sistemas heterogêneos.

1.3 Estrutura do Texto

Este documento organiza-se da seguinte forma: o Capítulo 1 apresentou a introdução, a motivação e os objetivos deste documento. Uma compilação das principais contribuições científicas trazidas por este trabalho também foi apresentada neste capítulo. O Capítulo 2 apresenta o cenário atual das casas de impressão, informações gerais a respeito de documentos digitais e os processos de renderização e rasterização, e, por fim, apresenta os trabalhos anteriores que servem como ponto de partida para as soluções que serão descritas aqui. O Capítulo 3 relaciona cada uma das diferentes etapas deste trabalho com outras pesquisas e produções científicas realizadas na mesma área. O

Capítulo 4 descreve o problema e mostra a solução proposta através de uma perspectiva geral. Em seguida, são apresentadas as etapas envolvidas no sistema, juntamente com os módulos envolvidos em cada uma delas. Todos os módulos são, então, definidos e descritos individualmente. O Capítulo 5 apresenta os testes realizados, os resultados obtidos e a validação das fórmulas e métodos encontrados. Finalmente, no Capítulo 6, estão as conclusões e as propostas para trabalhos futuros.

2. CONTEXTUALIZAÇÃO

Neste capítulo, serão discutidos aspectos importantes para que seja alcançada completa compreensão do contexto deste volume. Será descrito o cenário atual das casas de impressão de grande porte. Uma visão geral de formatos de documentos digitais com foco no formato PDF também será transmitida, seguida por uma noção dos conceitos de renderização e rasterização de documentos. Finalmente, serão descritas pesquisas anteriores relativas à aplicação de técnicas de distribuição e paralelismo à ambientes de impressão. Os resultados destas pesquisas são de fundamental relevância para o desenvolvimento do presente trabalho.

2.1 Ambientes de Impressão

O contexto global deste trabalho restringe os sistemas de impressão considerados a ambientes de impressão de grande porte e alta tecnologia. Embora gráficas e editoras de pequeno e médio porte possam, por exemplo, se beneficiar de um mecanismo de previsão para rasterização de documentos como uma ferramenta útil para outros propósitos, apenas as maiores dispõem de *clusters* e sistemas informatizados para que a solução de escalonamento sugerida possa ser completamente aproveitada.

Além das etapas de rasterização e renderização que se encontram dentro da fase de pré-processamento dos documentos, existem outras etapas no processo de impressão. As etapas vão desde a solicitação de um serviço por um cliente ou uma editora até a distribuição do produto final. A Figura 2.1 fornece uma visualização geral do fluxo de produção em uma casa de impressão. De um cliente até o distribuidor, uma tarefa passa pelas fases de pré-impressão, pós-impressão e geração do produto final a partir das folhas impressas. No decorrer das etapas, estão presentes alguns MIS (*Management Information System*), que controlam e administram processos.

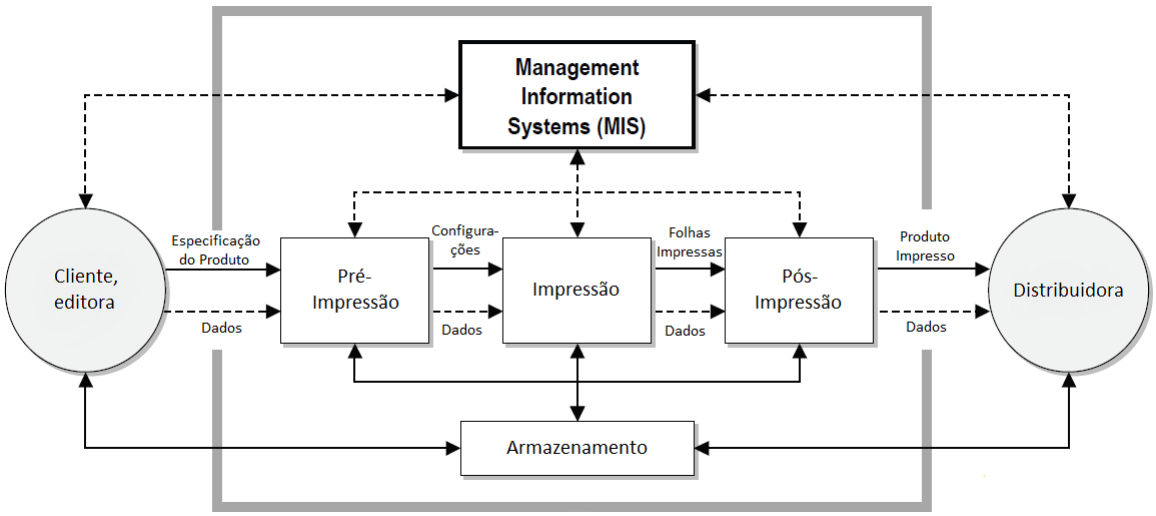

Figura 2.1: Fluxo de Produção em uma Casa de Impressão [7]

A fase de pré-impressão prepara o material solicitado pelo cliente para que esteja pronto para a

impressão. Como entrada desta etapa, estão as especificações do cliente a respeito dos produtos a serem impressos, juntamente com a matéria prima, que são, hoje em dia, quase exclusivamente documentos digitais do tipo PDF [7]. A saída desta etapa são imagens e parâmetros de configuração específicos para as impressoras. A fase de impressão gera folhas impressas que são recebidas na fase de pós-impressão, onde são cortadas, agrupadas, encadernadas e encapadas. Como saída desta última fase já estão os produtos finais prontos para distribuição.

Em 2001, com o surgimento do protocolo JDF (*Job Definition Format*) (tratado com detalhes na Seção 3.1), a integração entre as etapas ganhou mais automação, mas muitos controles continuam sendo manuais ou bastante primitivos. Em casas de impressão de grande porte, múltiplos *clusters* de computadores são encarregados das etapas de pré-processamento dos documentos. A cada um destes *clusters*, impressoras industriais estão associadas, imprimindo as imagens geradas após a rasterização dos documentos. Estes *clusters* podem estar fisicamente separados e podem possuir diferentes capacidades de processamento. As impressoras podem também ter velocidades diferentes, e o tamanho de cada *cluster* pode ser ajustado conforme a velocidade das impressoras, para evitar que elas fiquem ociosas e que seja perdida eficiência.

Tarefas que chegam para impressão são endereçadas para uma fila, onde esperam sua vez de serem processadas. Esta fila recebe tarefas novas de forma dinâmica, que podem chegar a qualquer momento, possuindo diferentes propriedades, complexidades e prioridades.

A distribuição de tarefas entre os *clusters* é feita de forma manual. De acordo com as necessidades de tempo do cliente e características de suas tarefas, um controlador humano seleciona o *cluster* que julga otimizado para tal tarefa, adequado para as necessidades do cliente, ou apenas utiliza o único disponível naquele momento.

Já no escalonamento local, ou seja, na distribuição de tarefas que chegam para cada *cluster* entre seus núcleos de processamento, há uma forma automática de escalonamento sendo utilizada. As estratégias de escalonamento implantadas são primitivas, pois não utilizam inteligência, análise das características das tarefas, nem se preocupam com ajustes finos de balanceamento de carga, deixando máquinas ociosas e, consequentemente, reduzindo eficiência. As estratégias utilizadas são as seguintes:

- *Alocacação de uma máquina para cada tarefa*: pode ser interessante quando as tarefas forem todas de baixo custo computacional, mas quando o tempo de processamento apresenta variação e tendências de crescimento, o resultado pode oferecer baixo desempenho, gerando sub-carga e sobrecarga das unidades de processamento;

- *Alocação de todas as máquinas para cada tarefa*: esta estratégia quebra uma tarefa em unidades iguais, ou seja, uma quantidade de páginas iguais para cada unidade de processamento. Aqui, se abdica das vantagens trazidas pela reutilização de objetos nos documentos e pelo armazenamento em memória dos mesmos, uma vez que objetos iguais podem acabar sendo distribuídos para unidades diferentes. Além disto, as cargas distribuídas para cada unidade podem estar altamente desbalanceadas, uma vez que não são conhecidas as carac-

terísticas das páginas contidas em cada grupo;

- *Alocação de um número fixo de máquinas para cada tarefa*: ajustar a alocação de um número fixo de máquinas para cada tarefa pode trazer algum desempenho quando as tarefas apresentarem características similares, ou estiverem agrupadas na fila com algum padrão específico, porém, de forma geral, traz as mesmas desvantagens das duas estratégias anteriores.

É notável o avanço tecnológico e o aumento da produção nas casas de impressão que se utilizam de sistemas informatizados e técnicas paralelas de processamento para aceleração das etapas de pré-processamento. São observáveis, no entanto, diversas deficiências relacionadas ao balancemanto de carga e processos que ainda não são automatizados. Estas deficiências são passíveis de pesquisas e melhorias capazes de aumentar a eficiência do escalonamento de tarefas.

2.2 Documentos Digitais

A tecnologia de impressão atual dispõe de impressoras digitais industriais de alta capacidade que imprimem grandes quantidades de páginas em pouco tempo e atingem altíssimas resoluções, podendo imprimir até *outdoors* inteiros. As impressoras permitem ajustes finos de tonalidades de cor, transparência, brilho e contraste, entre outras propriedades. Com a possibilidade de imprimir documentos com cada vez mais recursos, diversas formas de simplificar a criação digital destes documentos passaram a ser investigadas. Impressoras reconhecem imagens *bitmap*, que são imagens nas quais cada *pixel* contém uma descrição a respeito de suas propriedades. Gerar manualmente cada uma das imagens a serem impressas não é viável e, em razão disto, tanto se utilizam hoje as PDLs (*Page Description Languages*) [8].

A primeira tendência que alavancou o desenvolvimento das PDLs atuais foi o VDP, um conceito relativo à construção de documentos flexíveis. Com VDP, é possível imprimir diversos documentos com porções fixas e porções variáveis [9]. Empresas publicitárias enviam, por exemplo, folhetos com formatação idêntica, porém com o nome do destinatário específico e cores diferentes para homens e mulheres. DOCs de pagamento enviados por uma empresa (possuem sempre o mesmo *layout*) são outros exemplos de utilização dos conceitos de VDP.

As PDLs são, então, linguagens de alto nível de abstração que descrevem o conteúdo de um documento. Nas PDLs modernas é possível descrever documentos utilizando-se métodos e objetos como em uma linguagem de programação orientada a objetos. Justamente em função deste alto nível de abstração, impressoras e dispositivos de vídeo não reconhecem estas linguagens. As imagens precisam, então, passar por uma fase de pré-processamento que as convertam em um formato reconhecível, para que possam ser impressas ou exibidas. Entre as PDLs mais conhecidas estão o PS e a mais utilizada PDF, que se trata de uma nova PDL baseada no próprio PS.

O formato PDF é uma PDL proprietária desenvolvida pela empresa Adobe. Trata-se de uma linguagem de descrição de documentos independente de *software*, *hardware* e sistema operacional utilizado para criá-lo e de sistemas de saída utilizados para impressão ou exibição. Um documento

PDF consiste de um agrupamento de objetos que descrevem a aparência e o conteúdo (textos, gráficos e imagens) de uma ou mais páginas. O arquivo PDF é constituído por objetos organizados segundo uma estrutura fixa, todos representados através de sequencias de *bytes*.

Os principais objetos gráficos de um PDF são:

- *Path Objects*: sequencia de pontos conectados ou desconectados, linhas e curvas que juntos descrevem uma forma e seu posicionamento na página;
- *Text Objects*: um ou mais glifos que representam caracteres de texto. As formas dos glifos são descritas em uma estrutura separada chamada de *font*;
- *Image Objects*: matriz retangular representando uma cor em determinada posição. São objetos tipicamente utilizados para representar fotografias.

Outras características gerais relativas ao formato PDF são:

- *Portabilidade*: pode ser criado, transportado e lido em sistemas com diferentes arquiteturas e sistemas operacionais;
- *Compressão*: a fim de reduzir o tamanho do arquivo, são suportados diversos tipos de compressão e descompressão para imagens e gráficos;
- *Gerência de Fontes*: suporta vários tipos de fonte e permite a instalação de novas, porém o leitor do documento deve conter as mesmas fontes usadas na sua criação, caso contrário, fontes diferentes podem substituí-las;
- *Geração de Arquivos com única passagem*: possibilidade de criação de um PDF mais leve, sem todos os recursos, para que possa ser gerado e lido em sistemas com pouca memória;
- *Acesso Aleatório*: No final do arquivo, uma tabela de endereçamento chamada *cross-reference table* que guarda as localizações das páginas e dos outros objetos PDF. Desta forma, a ordem da descrição dos objetos e das páginas no arquivo não é importante;
- *Segurança*: o formato provê criptografia e assinatura digital, tornando os arquivos seguros contra cópias e modificações;
- *Atualização Incremental*: permite que arquivos sejam modificados de tal forma que novos conteúdos possam ser adicionados sem a necessidade de modificação ou reescrita do arquivo original;
- *Extensibilidade*: novas funcionalidades podem ser adicionadas e aplicações antigas conseguem lidar com elas com tranquilidade.

O formato PDF completo é complexo e seu total entendimento depende do estudo de detalhes sobre as propriedades de cada um de seus objetos, funcionalidades e estruturas do arquivo e do documento que já foram tratadas nos trabalhos anteriores discutidos na Seção 2.3.

2.2.1 Renderização e Rasterização

Renderização e rasterização são os nomes dados aos dois processos integrantes da etapa de pré-processamento de documentos. São os processos responsáveis pela interpretação do documento e conversão até que possam ser exibidos ou impressos. Embora sejam comumente confundidos como sinônimos, eles apresentam conceitos bastante dissemelhantes. O processo de renderização, muito utilizado na apresentação de resultados em aplicações de descrição de modelos 2D e 3D, se trata do processo de interpretação de uma linguagem não inteligível e não usual a um formato que possa ser compreendido por um ser humano ou alguma outra aplicação de exibição. O resultado de uma renderização é um documento descrito em uma linguagem alto nível mais usual, que pode ser lida com facilidade por outras aplicações (uma imagem vetorial, por exemplo). Já a rasterização ou RIPping (Raster Image Processing), é o processo que, a partir de uma imagem vetorial ou linguagem alto nível como uma PDL, converte o arquivo em uma imagem *bitmap*. As unidades de processamento responsáveis pela rasterização são chamadas de rasterizadores ou RIPs (*Ripping Image Processors*).

Especificamente em ambientes de impressão, um documento PDF pode ser renderizado a partir da extração de conteúdos PDF contidos dentro de uma descrição de documentos PPML(*Personalized Print Markup Language*) [10]. Como saída da renderização de um arquivo PPML, por exemplo, podem ser obtidos documentos PDF. Na próxima etapa, o PDF está pronto para ser rasterizado e enviado para a impressora em um formato de imagem baixo nível, que pode ser reconhecido por ela.

A forma mais usual de trabalho das casas de impressão é já utilizarem documentos PDFs, sem a necessidade de extraí-los de outras linguagens de descrição de documentos. Assim, o processo de renderização está fora do contexto desta pesquisa.

2.3 Trabalhos Anteriores

O sistema proposto aqui envolve um escalonador global que distribui tarefas entre dois ou mais *clusters*. Em um âmbito local, tratando do escalonamento de documentos PDF dentro de cada *cluster*, um estudo anterior, com resultados bastante satisfatórios, já foi realizado focando as etapas de renderização [11] e de rasterização [12]. É importante que sejam conhecidos os trabalhos envolvendo rasterização, uma vez que as ferramentas desenvolvidas, os algoritmos propostos e os resultados obtidos formam o alicerce do escalonador global.

Uma ferramenta chamada *PDF Profiler* foi desenvolvida [13]. A ferramenta extrai as informações mais relevantes de cada página de um documento PDF. Conhecendo as características dos objetos contidos em cada página, através de testes que coletaram o tempo de rasterização, foram encontrados os fatores que mais influenciam no tempo de processamento: resolução das imagens, transparência e objetos reutilizados. Estes testes possibilitaram que fossem encontradas métricas que expressam o custo computacional de cada tarefa. Conhecendo o custo das tarefas, puderam ser testadas adaptações para diferentes algoritmos de escalonamento, que objetivam encontrar um balanceamento de carga ideal para determinado *cluster* de máquinas [14].

A Figura 2.2 apresenta uma simplificação do sistema e mostra o *Profiler* gerando um arquivo XML contendo as informações extraídas dos documentos que esperam na fila de tarefas. O XML contém informações sobre todos os objetos presentes em cada página, a área opaca e a área transparente ocupada por cada um, área de sobreposição, quantidade de vezes e número das páginas nas quais cada objeto foi reutilizado. Informações sobre presença de cor e espaço de cor também estão presentes no arquivo. O roteador adaptativo calcula o custo das tarefas baseado em métricas pré-definidas e faz a distribuição para os rasterizadores do *cluster*. Os algoritmos e as estratégias utilizadas pelo escalonador são de extrema relevância e alguns aspectos já pesquisados a este respeito serão abordados na subseção seguinte.

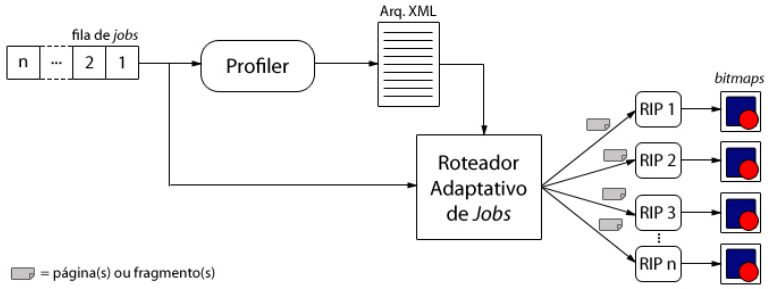

Figura 2.2: Sistema de *Profiling*

O custo das tarefas calculado através das métricas (detalhadas e validadas em [12]) é diretamente proporcional ao tempo de processamento das mesmas. Em uma máquina de alto desempenho e de uso exclusivo, foram executados testes nos quais arquivos PDFs com custos variados foram rasterizados. O tempo de rasterização foi coletado e, a partir dos pontos apresentados no gráfico da Figura 2.3, pode-se observar uma tendência linear do crescimento do tempo de rasterização em relação ao valor de custo calculado para os diferentes documentos.

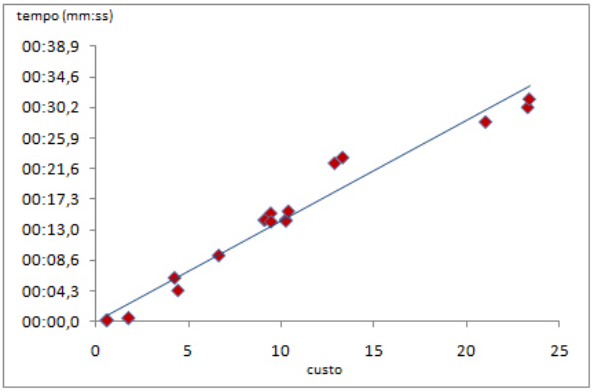

Figura 2.3: Gráfico Tempo x Custo

É importante para o entendimento do comportamento de alguns módulos do sistema tratado aqui, que sejam levantados alguns pontos que possivelmente são responsáveis pelas pequenas flutuações observáveis no gráfico da Figura 2.3. Além de processos concorrentes que comumente executam em sistemas operacionais que não são de tempo real e podem, esporadicamente, influenciar no tempo de processamento de um teste específico, existem ainda alguns aspectos do documento

PDF considerados menos relevantes (pouco custo computacional) que ainda não são levados em conta pelo *Profiler* e, por conseguinte, também não estão incluídos nas métricas. Estes aspectos são objetos PDF como os *Shadding Patterns*, *Path Objects* e objetos que utilizam máscaras *Soft Mask* para transparência com degradês. Fontes de texto de alfabetos com diferentes grafias ou fontes especiais muito desenhadas também podem influenciar. Embora sejam fatores de peso computacional leve em comparação aos analisados pelas ferramentas, quando em quantidade, podem sim gerar flutuações e afastar alguns pontos da linha de tendência.

2.3.1 Escalonamento Local

Em [13] foram estudados e implementados três algoritmos de escalonamento local, focando um sistema de computação paralela para rasterização de documentos PDF. Os seguintes algoritmos foram investigados:

- *List Scheduling*: neste algoritmo, uma tarefa é associada a uma máquina somente quando tal máquina estiver ociosa. Quando uma máquina acaba uma tarefa, recebe mais trabalho para processar. O *List Scheduling* não é a melhor solução e apresenta considerável queda de desempenho principalmente quando a última tarefa é a de maior custo computacional e as outras máquinas ficam ociosas esperando até que a última máquina termine o trabalho;

- *Largest Processing Time First*: este algoritmo visa eliminar o problema do algoritmo anterior ordenando a fila de espera de tarefas em ordem decrescente. Isto evita que o trabalho mais longo seja o último a ser processado. O *Largest Processing Time First* obteve melhor desempenho em relação ao *List Scheduling*, mas continua se tornando lento à medida que a quantidade de máquinas servidas pelo escalonador aumenta;

- *Multifit*: primeiro algoritmo a apresentar um ganho relativamente significativo em relação ao *Largest Processing Time First*. Neste algoritmo, ordena-se a fila de tarefas em ordem decrescente e, utilizando um algoritmo com a heurística FFD (*First-Fit Decreasing*), em poucas iterações, o algoritmo consegue agrupar as tarefas em pacotes, de modo que cada pacote possua tempo de execução similar. A quantidade de pacotes deve ser igual ao número de máquinas disponíveis.

Também em [13], como otimização para o escalonamento, é proposta a adição de uma *thread* que atribuiria os valores de custo computacional concorrentemente com a organização da fila. Caso não houvesse tempo de atribuir um valor e colocá-lo na posição correta da fila, a tarefa seria escalonada mesmo assim. No caso do *Multifit*, ela seria colocada no pacote com menor custo até o momento.

Outra abordagem para o escalonamento é a utilização de uma ferramenta que, baseada nas informações providas pelo *Profiler*, divide a tarefa em segmentos de custos equivalentes, alcançando um nível de balanceamento de carga bastante equilibrado. Tal abordagem, porém, impossibilita a estratégia que utiliza o *Profiler* como uma *thread*, uma vez que a ferramenta responsável pela divisão

é inteligente e precisa custo da tarefa e do arquivo XML gerado pelo *Profiler* para fazer a quebra otimizada das tarefas. Em [15] foi desenvolvido um estudo a respeito da vantagem da utilização deste método.

3. TRABALHOS RELACIONADOS

O sistema de otimização e automação de processos em ambientes distribuídos de impressão proposto por este trabalho é composto por diversas etapas específicas e trabalhos com propostas similares ainda não estão presentes na literatura. Este capítulo aborda os principais trabalhos científicos relacionados especificamente com os diferentes assuntos envolvidos aqui, objetivando justificar as decisões tomadas e possibilitar a realização de comparações entre o trabalho e as últimas pesquisas relacionadas com ele. Serão tratadas pesquisas nas áreas de automação de ambientes de impressão, sistemas de busca de recursos, sistemas de predição de tempo de processamento e algoritmos de escalonamento para sistemas heterogêneos.

3.1 Automatização de Ambientes de Impressão

Conforme visto anteriormente, mesmo nas grandes casas de impressão, o escalonamento de tarefas ainda é feito manualmente e não existem trabalhos a respeito de escalonamento automático. Tratando-se de automação para sistemas de impressão, podem ser apontados os protocolos criados especificamente para este fim.

Embora na maioria das casas de impressão grande parte das etapas envolvidas já fossem informatizadas, a operação dos MIS (máquinas responsáveis pelo controle e administração dos processos), conexão ente eles e a conexão entre os dispositivos era feita de forma manual até o surgimento do protocolo JDF(*Job Definition Format*) [16], em 2001.

O protocolo JDF foi desenvolvido especificamente para automatizar o processo de impressão. Baseado em XML, o JDF é independente de plataforma e facilmente pode ser suportado pelos diferentes dispositivos envolvidos. A função do JDF é lidar com o fluxo de tarefas do início ao fim do processo e fazer a conexão dos MIS com os outros dispositivos. O JDF carrega, juntamente com a tarefa, informações sobre o estado e sobre as próximas etapas às quais cada parte da tarefa deve ser submetida, indicando os locais por onde elas passaram e ainda devem passar. Em uma tarefa JDF, cada etapa é representada por um nodo de uma árvore, a qual representa o produto final desejado. Na Figura 3.1, pode ser observada a estrutura da árvore. A hierarquia dos nodos da árvore organiza-se por conexões baseadas em consumo de entradas e produção de saídas. Para a maioria das etapas, a saída de cada uma delas é entrada das etapas futuras. Cada sub-nodo da árvore define o componente de um produto que contém características em comum, como tamanho físico ou restrições de cor. Os nodos pertencentes aos níveis intermediários da estrutura representam um grupo de processos necessários para produzir um componente do produto. Finalmente, os nodos folhas provêm informação detalhada específica para que um dispositivo realize determinada operação [17].

Um ambiente de impressão compatível com JDF é composto por quatro componentes principais [7]:

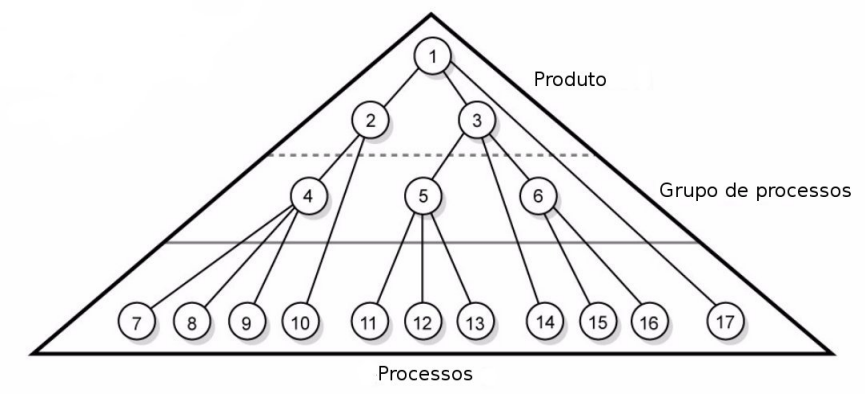

Figura 3.1: Estrutura da Árvore JDF [16]

- *Agente*: sistema que escreve instâncias JDF. Pode atualizar e adicionar informações às instâncias JDF que processam;
- *Controlador*: interpreta instâncias JDF, podendo, ocasionalmente, adicionar informações a elas. O Controlador é responsável pelo roteamento das instâncias JDF para futuras etapas. Ele pode, também, juntar pedaços ou dividir uma instância em diferentes partes a fim de roteá-las para diferentes processos. Controladores estão localizados no topo da hierarquia e, aquele localizado na raiz da estrutura, é considerado um MIS;
- *Dispositivo*: sistema que recebe instâncias JDF e executa os procedimentos nelas especificados;
- *Máquina*: sistema que recebe comandos de um dispositivo, mas não conhece nada sobre o protocolo JDF. Um exemplo deste componente pode ser uma impressora.

Durante o processo de impressão controlado por mecanismos que implementam o JDF, a comunicação entre os controladores e dispositivos é indispensável e ocorre com grande frequência. Muitas vezes, um sistema deseja apenas enviar um comando ou trocar pequenas informações sem que uma tarefa JDF inteira precise ser enviada. Para servir como um protocolo de comunicação entre os componentes JDF, a própria especificação do JDF [16] define o JMF (*Job Messaging Format*) [18].

As mensagens JMF utilizam o protocolo de transporte HTTP [19]. As mensagens são pequenos documentos XML que consistem de um elemento raiz chamado JMF, o qual pode conter uma ou mais mensagens. Seis famílias de mensagens são suportadas pelo protocolo:

- *Mensagens de comando*: modificam o estado do sistema;
- *Mensagens de solicitação*: solicitam o estado de algum sistema;
- *Mensagens de registro*: instruem um sistema a enviar uma mensagem de comando a outro sistema;
- *Mensagens de resposta*: respostas síncronas enviadas após o recebimento de mensagens de outras famílias;

- *Mensagens de reconhecimento*: respostas assíncronas enviadas após o recebimento de mensagens das famílias de comando, solicitação ou registro;
- *Mensagens de sinal*: eventos enviados por um sistema toda vez que ele atinge determinado estado. Para especificar quais sinais um sistema deve gerar, são utilizadas mensagens de solicitação.

As mensagens de comando, solicitação e registro são as mensagens que iniciam uma comunicação JMF e são sempre enviadas em um HTTP *Request*. Embora as mensagens de reconhecimento e de sinal sejam mensagens enviadas em resposta a alguma outra mensagem, elas também são transmitidas em um HTTP *Request*. Apenas as mensagens de resposta são enviadas em uma mensagem de HTTP *Response*.

Estes protocolos representaram um grande avanço na área de automatização de processos de impressão, mas ainda são relativamente novos. Nos próximos anos, com a maturação destes protocolos, novas aplicações e facilidades para se trabalhar com eles devem emergir.

3.2 Busca de Recursos

A busca de recursos corresponde a um mecanismo que coleta informações sobre recursos disponíveis em um computador ou um aglomerado deles. A partir dos resultados obtidos com a busca de recursos, um escalonador pode, por exemplo, tomar decisoes mais acertadas sobre como balancear melhor as cargas em um *cluster*, ou estimar o tempo de execução de uma aplicação. Existem muitos sistemas de busca de recursos e uma vasta gama de pesquisas em torno de seus desempenhos e funcionalidades. Os modelos de sistemas mais clássicos foram estudados como forma de servir de inspiração para construir o sistema de busca de recursos utilizado neste trabalho. Sistemas de busca de recursos podem ser centralizados ou distribuídos, dependendo do cenário no qual será utilizado.

Para busca de recursos distribuída, existem sistemas como o SWORD [20], baseados na tecnologia DHT (*Distributed Hash Table*) [21]. DHTs são tabelas *hash*, com pares chave e valor, que podem estar distribuídas em diversos nós. Nesta tecnologia, ao ser buscada uma chave, um nó é capaz de recuperar seu valor independentemente do local em que ela esteja. Tabelas de mapeamento são mantidas de forma distribuída entre todas as máquinas. DHTs são conhecidas por sua alta escalabilidade e confiabilidade: entrada e saída de novos nós, mesmo ocorrendo com frequência, não abalam seu funcionamento. O SWORD é uma ferramenta para grandes grades computacionais, dando demasiada importância às diferentes latências das redes. Outra opção para a busca de recursos distribuída seria a utilização de agentes móveis, porém, por ainda não serem uma tecnologia consolidada, foram pouco explorados. Agentes móveis são agentes de *software* capazes de transportar-se autonomamente pela rede, coletando informações. Há apenas a necessidade de que cada máquina rode um servidor que gerencie os agentes. Em [22] e [23], são propostas soluções para busca de recursos em ambientes de computação distribuída utilizando agentes móveis.

A solução mais simples para um sistema centralizado é um sistema no qual cada *cluster* ou rede do sistema tenha o seu próprio monitor de recursos e uma única máquina é responsável por

coletar os dados de todas as outras e fazer o processamento necessário. Sabe-se que, em modelos centralizados, quando a dimensão do sistema cresce, um grande *overhead* é gerado na máquina, que pode demorar a tratar e gerar as solicitações. A demora no tratamento das informações pode gerar problemas do tipo: ao terminar de coletar e processar todos os dados, o sistema já se modificou e as informações estão desatualizadas. Mesmo assim, por serem de simples implementação, estes modelos são muito usados na prática.

A fim de realizar um estudo sobre busca de recursos centralizada, foi analisada uma solução publicada em [24], a qual foi escolhida por tratar-se de um sistema simples e que possui bem definidas as características de um sistema centralizado. Além disto, o sistema é baseado em XML, fator que torna mais fácil uma adaptação para que ele possa vir a funcionar em conjunto com os outros módulos tratados neste trabalho. Para troca de informações, o sistema estudado utiliza *Web Services* [25].

O sistema proposto em [24], segundo os autores, possui como grande vantagem a possibilidade de adição de novos recursos, não limitando-se apenas a recursos de *hardware* e disponibilidade de processamento. A existência ou não de um *software* instalado na máquina, por exemplo, também pode ser tratada como um recurso.

O sistema é baseado em uma arquitetura mestre/escravo, na qual o mestre é responsável por atualizar o banco de dados com os recursos disponíveis e os escravos são responsáveis por obter as informações de cada máquina do sistema. Os serviços são baseados em *Web Services* e tanto o serviço mestre quanto o escravo, são descritos em WSDD (*Web Service Data Definition*). A máquina que roda o serviço mestre possui um arquivo XML contendo os endereços IPs e nomes de todas as máquinas que estão sob sua responsabilidade. No serviço escravo, é mantido um arquivo XML sempre atualizado com as informações que podem ser solicitadas pelo mestre, como mostra a Figura 3.2.

```
<?xml version="1.0" enconding="UTF-8" standalone="yes"?>
<System>
    <Host>
        <Name>machine-00</Name>
        <IP>192.168.0.1</IP>
        <Memory>
            <Total>1098880.0</Total>
            <Available>1496.0</Available>
        </Memory>
        <CPU>
            <Model>AMD Athlon(tm) Processor</Model>
            <Total>896.0</Total>
            <AvailablePercent>1.36148<AvailablePercent>
        </CPU>
        <OS>
            <Name>Linux</Name>
            <Version>2.4.20-0</Version>
        </OS>
        <Process>
            <Number>78</Number>
        </Process>
    </Host>
</System>
```

Figura 3.2: Exemplo de um XML Mantido no Serviço Escravo [24]

Novos módulos (com novos tipos de informação) podem ser incorporados ao sistema em tempo

de execução. Para incorporar um novo módulo, o usuário precisa apenas desenvolver, em Java, um método que busque as informações desejadas. Feito isto, ele deve criar um arquivo de configuração contendo o nome da classe, a quantidade e o nome dos métodos que desenvolveu e o caminho para o arquivo de extensão *.jar* (*Java Archive*), que deve ser gerado contendo as classes desenvolvidas. Ao ser detectado um novo módulo, ele é automaticamente incorporado ao sistema. A detecção de módulos personalizados é feita periodicamente através da verificação da existência do arquivo de configuração. Quando o arquivo é encontrado, o arquivo *PersonalizedServices.xml*, responsável pela definição de módulos adicionais, é atualizado. Este arquivo contém a definição e o nome das classes e métodos de todos os módulos adicionais a serem invocados. Em seguida, os arquivos *.jar* definidos na configuração são adicionados a um arquivo GAR (*Grid Archive*). O Arquivo GAR é então enviado, juntamente com o *PersonalizedServices.xml*, para todos os serviços escravos. No serviço escravo, toda vez que requisições de recursos são solicitadas, o serviço verifica se há módulos adicionais definidos. Se houver, o *PersonalizedServices.xml* é lido e as classes responsáveis por procurar aqueles recursos são instanciadas, seus métodos são invocados e os resultados são inseridos no XML de saída, na *tag PersonalizedServiceInfo*.

Os testes realizados neste sistema mostraram que, quando diminui a frequência de verificações por novos módulos, o sistema torna-se mais lento, pois leva mais tempo até que ele perceba que um novo método foi definido. Este tempo aumenta ainda mais quando o número de máquinas escravas cresce. Outro fator que causou impacto no tempo do mecanismo testado em [24] foi o tamanho do arquivo GAR e este precisa ser controlado.

A possibilidade de incorporação de novos métodos agrega valor ao mecanismo e, num ambiente de processamento de documentos, pode tornar-se muito útil, uma vez que dependendo da aplicação, diferentes *softwares* devem estar instalados nas máquinas para que possam ser utilizadas. Métodos que coletam informações sobre latência de rede podem também ser incorporados e aumentar a eficiência das decisões tomadas pelo escalonador. Como desvantagens, os autores citaram o custo do envio do arquivo *GAR* e o baixo desempenho quando a rede aumenta muito e necessidade de que todas as máquinas envolvidas possuam um IP válido na Internet.

3.3 Sistemas de Previsão

Mecanismos capazes de fornecer uma estimativa de tempo de execução de tarefas apresentam variadas formas de aplicação e são úteis em diversas ocasiões. Estimativas de tempo de transferência de dados, instalação de aplicativos, operações de disco, respostas de sistemas remotos são algumas das formas pelas quais estas predições podem auxiliar não só usuários humanos como outros sistemas a fazer planejamento de tarefas futuras, capacidade, rendimento e desempenho.

Grande parte das aplicações que contam com algum tipo de mecanismo de previsão, possuem um mecanismo baseado em uma metodologia simples, mas com boa acurácia. São mecanismos que, a partir de uma noção da totalidade de uma tarefa (tamanho de arquivo, quantidade de etapas a serem processadas), em tempo de execução, determinam quanto da tarefa já foi realizado até determinado

momento e, desta forma, estimam o tempo restante até a conclusão da tarefa. Este método é simples e eficaz, embora possam ocorrer grandes flutuações da previsão, quando o sistema se torna mais ou menos carregado, quando a velocidade da rede se altera, quando a memória disponível diminui ou quando o leitor de um disco se afasta ou se aproxima do centro.

Em certas ocasiões, não é viável ou desejável fazer uma predição em tempo de execução. Pode ser desejável obter uma estimativa sem a necessidade de executar a tarefa. Quando se necessita de uma estimativa de tempo atribuída de forma rápida e eficiente à determinada tarefa, sem a possibildade de fazer grande processamento, os mecanismos passam a se tornar algoritmos complexos, com maior dificuldade na obtenção de acurácia e de desempenho. Como exemplo, no caso de uma empresa desejar apenas fornecer ao cliente uma estimativa para o término de sua computação, mas não estiver apta a iniciá-la no exato momento, estas soluções mais complexas passam a se tornar inevitáveis. Outras aplicações para as quais algoritmos complexos de previsão se tornam necessários são aplicações que utilizam predições de tempo para escalonamento de tarefas em multiprocessadores, *clusters* ou grades computacionais [26].

Um escalonador, quando possui uma estimativa do tempo de execução das tarefas presentes em sua fila, pode tomar decisões inteligentes a fim de balancear as cargas, reduzir o tempo total de processamento da fila e lidar melhor com possíveis restrições de tempo impostas às tarefas. São a este fim que se destinam a maioria dos raros trabalhos científicos relacionados com predição de tempo.

Os mecanismos abordados na literatura são mecanismos genéricos, para serem usados em tarefas gerais. Por este motivo, eles implementam heurísticas e métodos de inteligência artificial baseados em aprendizagens anteriores para realizar suas estimativas. As aplicações são classificadas através de análise de código, quantidade de instruções ou comportamentos passados.

Em [27] é proposto um algoritmo de escalonamento para gerenciamento de balanço de carga em ambientes de grades computacionais. O algoritmo é baseado em um sistema de predição de desempenho que provê estimativas de tempo de execução para diferentes tarefas de acordo com as diferentes configurações de recursos disponíveis nas máquinas. A predição leva em conta a aplicação e recursos de *hardware*. A aplicação é avaliada através da análise de código C. Os componentes sequenciais do código são analisados através de grafos de controle de fluxo e, a partir dele, o custo computacional da aplicação é estimado. Os recursos de *hardware* são caracterizados através de *micro-benchmarking* e técnicas de modelagem para comunicação e hierarquia de memória.

Um sistema baseado em aprendizagens anteriores descrito em [28] propõe a categorização de tarefas similares por grupos. Ao chegar uma tarefa qua ainda não havia sido executada no passado, ela é colocada em grupo ao qual suas características mais se assemelham. O algoritmo parte do pressuposto de que tarefas similares possuem um tempo de execução mais próximos entre si, do que tarefas sem nada em comum. Dois algoritmos de inteligência artificial (guloso e genético) foram utilizados para gerar *templates* contendo os atributos de cada tarefa e as similaridades entre os casos. Os erros das predições ficaram entre 40% e 59% das execuções reais.

Em [29], [30] e [31], são propostos e estudados modelos matemáticos lineares de predição como

o ARIMA [32]. A partir de experiências anteriores com aplicações similares, os coeficientes dos modelos vão sendo reajustados e os resultados se tornam mais precisos.

Em [33] é proposto um mecanismo de predição que também utiliza aprendizagem de experiências passadas, porém através de redes neurais. Este algoritmo utiliza a carga média de utilização do processador para as previsões, com erros médios de até 37% e variância média de 22%.

Os trabalhos presentes na literatura são mecanismos de predição para escalonamento em ambientes distribuídos, pois são as aplicações que mais se beneficiam destes mecanismos. Os modelos são todos com um propósito de estimar tempos para tarefas gerais, muito baseados na dependência das tarefas entre si. Não foram encontrados trabalhos focados em predição dedicada para alguma aplicação específica, e trabalhos que focam tarefas sem dependências entre si são muito raros.

3.4 Escalonamento

Antes de abordar escalonamento heterogêneo, é necessário entender quais são os tipos de escalonamento existentes e quando cada um deles é aplicado. Após apresentar a taxonomia dos algoritmos de escalonamento e dos tipos de tarefas, serão apresentados trabalhos relacionados que tratam especificamente do tipo de escalonamento necessário para ambientes distribuídos de impressão.

Escalonamento ou agendamento de tarefas corresponde a um algoritmo que organiza uma fila de tarefas para serem processadas por um ou mais núcleos de processamento. O escalonador é a aplicação que decide, através de diversas políticas, qual o melhor processador para executar cada tarefa, qual a faixa de tempo que cada tarefa executará em cada processador e, a partir das decisões tomadas, aloca os recursos e dispara a execução das tarefas. O objetivo final do escalonador é conseguir com que as tarefas executem aproveitando o máximo dos recursos possíveis, terminem no menor tempo possível, e respeitem restrições de tempo ou outras políticas específicas aplicadas às tarefas.

3.4.1 Tipos de Escalonamento e Tarefas

Algoritmos de escalonamento podem ser classificados em:

- *Estáticos ou Dinâmicos*: escalonadores estáticos são aqueles que recebem como entrada uma fila fixa pré-definida de tarefas para serem escalonadas. As decisões podem ser tomadas todas no início do processo, quando o escalonador atribui qual recurso será utilizado por cada uma das tarefas de acordo com suas características. Depois, as tarefas são executadas conforme foi decidido. Escalonadores dinâmicos são aqueles que podem receber novas tarefas na fila em tempo de execução. A fila pode sofrer novas alterações com o passar do tempo e o escalonador está sempre ativo tomando as melhores decisões para o conjunto atual de tarefas presentes na fila;

- *Preemptivos ou Não-Preemptivos*: preemptivos são aqueles escalonadores que permitem que uma tarefa em execução seja temporariamente interrompida para que seja retomada mais

tarde. Tal fenômeno pode ocorrer em decorrência da chegada de uma tarefa com maior prioridade ou do término de uma fatia de tempo de processamento cedida àquela tarefa. Já em escalonadores não-preemptivos, uma vez iniciada a tarefa, ela será executada até sua conclusão.

- *Homogêneos ou Heterogêneos*: escalonadores para sistemas homogêneos são aqueles nos quais as unidades de processamento para as quais as tarefas são distribuídas possuem todas a mesma capacidade de processamento, memória e disponibilidade de recursos. Já escalonadores para sistemas heterogêneos, tratam-se daqueles capazes de lidar com sistemas nos quais as unidades têm diferentes capacidades computacionais.

As tarefas podem ser classificadas em:

- *Periódicas*: também conhecidas como tarefas cíclicas. São aquelas tarefas das quais o tempo de chegada são previsíveis, pois seguem um ciclo de repetição determinado por políticas fixas;
- *Esporádicas*: tarefas das quais o escalonador não possui conhecimento de quando podem chegar. Tarefas esporádicas podem chegar a qualquer momento, ou nunca chegar.

Algumas propriedades importantes que tarefas podem apresentar são:

- *Dependência*: tarefas podem ou não apresentar dependências. Dependências entre tarefas impõem uma limitação ao escalonador, uma vez que tarefas dependentes devem esperar que outras tarefas atinjam algum estado específico para prosseguir. Quando há dependência entre tarefas, pode haver também a necessidade de troca de mensagens entre elas. Logo, quanto maior a dependência entre as tarefas, maior a complexidade do escalonador e menor a eficiência do mesmo, pois muitas etapas não poderão ser executadas em paralelo ou na ordem natural que seria escolhida pelo escalonador.
- *Prioridade*: tarefas com prioridade são aquelas que possuem um valor de prioridade atribuídos à elas. Quando encontram tarefas de menor prioridade, passam na frente na fila e ganham os recursos antes das demais. Existem escalonadores que trabalham com prioridades fixas ou variáveis. Prioridades fixas permanecem inalteradas durante o processo de escalonamento. Prioridades variáveis podem ser alteradas pelo escalonador, como forma, por exemplo, de dar chances de uma tarefa de baixa prioridade ganhar os recursos através do aumento de sua prioridade ao longo do tempo.
- *Deadline*: tarefas com *deadlines* são tarefas que apresentam restrições temporais. Elas possuem uma data máxima para terminar ou devem ser concluídas dentro de um espaço de tempo pré-determinado. Alguns sistemas mais complexos utilizam também o conceito de *deadlines* nominais e *deadlines* críticos, cuja ideia foi inicialmente introduzida em [34]. O *deadline* nominal é a restrição de tempo da tarefa e o *deadline* crítico expressa o tempo máximo aceitável

para que a tarefa ultrapasse seu *deadline* nominal, sem causar prejuízos reais ao sistema, que possam representar alguma catástrofe, como riscos de morte em sistemas hospitalares ou aeronáuticos.

Desta forma, em um ambiente de impressão, pode-se caracterizar o escalonamento de tarefas para diferentes *clusters* como um escalonamento dinâmico não-preemptivo para sistemas heterogêneos. O escalonamento é dinâmico pois o escalonador funciona ininterruptamente e novas tarefas são solicitadas por clientes, entrando no sistema em intervalos aleatórios. O escalonamento é não-preemptivo, pois não é desejável que parte de um documento seja rasterizado em um *cluster* e impresso em determinada impressora, enquanto outra parte migre para ser rasterizada em outro local e seja impressa em uma impressora diferente. Também não é desejável que se tenha tarefas diferentes sendo impressas de forma misturada. Quanto ao tipo de sistema, trata-se de um sistema heterogêneo, pois os *clusters* podem tanto ter dimensionamento diferentes, quanto serem compostos por máquinas com diferentes configurações de *software* e *hardware*. Em relação à natureza das tarefas, elas não têm dependência, uma vez que não dependem uma das outras para executar. São tarefas esporádicas, pois não obedecem nenhum tipo de ciclo de tempo para chegarem na fila. As tarefas têm *deadlines* impostos por clientes ou por usuários e as prioridades podem ser atribuídas conforme diferentes políticas que sejam eventualmente aplicadas.

3.4.2 Escalonamento Heterogêneo

O escalonamento heterogêneo somente passou a ser de interesse da comunidade científica após a popularização dos *grids* ou grades computacionais, que são *clusters* de computadores, podendo estar geograficamente distantes (diferentes universidades, cidades, países ou continentes) interconectados através de uma rede, cooperando entre si e realizando atividades em paralelo. Estes são sistemas claramente heterogêneos e extremamente complexos, que abriram um novo ramo de pesquisa para otimizar o escalonamento nestes ambientes. Os estudos encontrados são direcionados para escalonamento de tarefas com dependências, que utilizam comunicação entre si e, em sua maioria, sistemas preemptivos e sem *deadlines*. Na literatura não são abordados sistemas similares ao necessário neste trabalho (dinâmicos, não-preemptivos, baseados em *deadlines* e específicos para tarefas sem dependências), mas uma quantidade razoável de estudos foi realizada com escalonamentos baseados em mecanismos de predição, e conhecê-los de forma geral, elucida as vantagens e desvantagens do escalonamento prosposto neste trabalho.

Fórmulas matemáticas para determinar qual o escalonamento ótimo para um conjunto específico de tarefas existem somente para sistemas com diferentes arquiteturas de máquinas paralelas. Em [35], é apresentada uma fórmula ótima para escalonamento de tarefas quando máquinas com diferentes arquiteturas estão presentes. Através de análise de código C, um programa é quebrado em segmentos de modo que o segmento ideal para ser processado por uma máquina MIMD (*Multiple Instruction Multiple Data*) será alocado para ela e o segmento para uma máquina SIMD (*Single Instruction Multiple Data*) será atribuído à máquina correspondente.

Um algoritmo ótimo para a escolha da melhor máquina para uma determinada tarefa entre um conjunto de máquinas com diferentes capacidades ainda é um problema sem solução [36].

Em [37] e [38] é tratada uma abordagem que visa dividir as tarefas em grãos mínimos para que a dosagem de trabalhos para cada máquina seja controlada em tempo de execução, enviando mais grãos para as máquinas com maior poder de processamento e menos grãos para as máquinas mais limitadas.

No trabalho desenvolvido em [27], que apresenta uma abordagem para escalonamento baseado em predições, são propostas três opções de algoritmos para escalonameto em sistemas heterogêneos. Estes algoritmos funcionam com a distribuição de uma tarefa para as máquinas presentes através de particionamento. As três opções sugeridas pelos autores são: *Deadline Sort* (ordenação por *deadlines*), *Deadline Sort with Node Limitation* (ordenação por *deadlines* com limitação de nós) e *Genetic Algorithm* (algoritmo genético). O *Deadline Sort* ordena a fila de tarefas por ordem de *deadlines* (quanto mais cedo for o *deadline*, maior a prioridade da tarefa) e divide a tarefa pela quantidade de nós. A vantagem deste algoritmo está na ordenação por prioridades, pois a divisão das tarefas pelo número de nós não garante balanceamento e a possibilidade de máquinas com menos carga necessitarem esperar até que as outras terminem o trabalho é grande. O *Deadline Sort with Node Limitation* apresenta uma sútil melhora limitando a quantidade de nós alocadas para cada tarefa, mas embora reduzido, o desbalanceamento continua apresentando probabilidade de ser bastante acentuado. No *Genetic Algorithm*, o escalonador escolhe entre diversas opções de escalonamentos possíveis, qual o melhor para aquele grupo de tarefas, baseado em aprendizagens dos escalonamentos anteriores. Existe uma função de *fitness* que utiliza como métrica o tempo entre o início da primeira tarefa até o término da última (*makespan*). O escalonamento que possuir o melhor *fitness* será aplicado para aquele grupo de tarefas. Para a realização de testes foram criadas 32 tarefas com *deadlines* cíclicos de 5 a 11 minutos. Utilizando o algoritmno genético o melhor resultado foi obtido, tendo o algoritmo levado em torno de 400 segundos para terminar as tarefas contra 590 segundos do algoritmo com limitação de nós e 1200 segundos do algoritmo que aloca todos nós para cada tarefa.

No trabalho descrito por este volume, foi seguida uma estratégia para melhorar o escalonamento das tarefas nos ambientes de impressão que apresenta conceitos semelhantes com a técnica discutida em [39], relativa a um algoritmo de escalonamento dinâmico não-preemptivo para um único processador. Estimando o tempo de execução de todas tarefas, quando houver tempo disponível, o processador é mantido em estado *idle*, permitindo que uma tarefa com mais prioridade que chega na fila durante este período, possa ganhar o processador na frente das outras. O estudo mostrou que esta técnica pode aumentar bastante a qualidade de um escalonamento.

Estes trabalhos representam o material científico encontrado na literatura que tratam de alguns aspectos de escalonamento que relacionam-se com os conceitos aplicados nesta pesquisa. Na Seção 4.3, que discorre especificamente a respeito do escalonador contruído, mais informações sobre algoritmos de escalonamento podem ser encontradas.

4. SISTEMA PROPOSTO

Após o trabalho direcionado ao ganho de desempenho em um âmbito local, dentro de cada *cluster*, o cenário apresentado pelas casas de impressão atuais permite também a realização de uma análise mais global do fluxo de tarefas. Esta análise pode evidenciar novos aspectos a serem otimizados. Normalmente, existe mais de um *cluster* responsável pela etapa de rasterização de documentos, podendo, o ambiente, ser comparado a uma pequena grade computacional. A otimização do gerenciamento da fila das tarefas que esperam pelo processamento, assim como do escalonamento e balanceamento de carga entre os *clusters* presentes (podem ser heterogêneos), pode aumentar ainda mais a velocidade do sistema como um todo. Neste capítulo, além de uma visão global do funcionamento de todo o sistema, cada um dos módulos envolvidos é descrito em detalhes juntamente com sua construção e estudos envolvidos.

Apesar do surgimento, em 2001, dos protocolos JDF e JMF (tratados detalhadamente na Seção 3.1), específicos para auxiliar na automatização do processo de impressão, encapsulando tarefas e trocando mensagens baseadas em XML, estes protocolos são relativamente novos e apenas são utilizados nas redes de impressão mais sofisticadas. Além disto, suas especificações são complexas e as bibliotecas para Java e C++ que facilitariam o trabalho com eles ainda estão em fase de desenvolvimento. Por este motivo, optou-se por construir o sistema de escalonamento baseado em comunicação TCP com *sockets*, aumentando a viabilidade de sua implantação em qualquer casa de impressão. No futuro, uma adaptação para um sistema completamente baseado em JDF e JMF deve ser passível de realização sem maiores alterações no sistema.

O escalonamento de tarefas para os diferentes *clusters* de uma casa de impressão é uma das tarefas que ainda são realizadas manualmente. De acordo com as necessidades e urgências no processamento das tarefas, as tarefas são repassadas, por usuários, aos diferentes *clusters*. Diferentemente das abordagens para o escalonamento local, no contexto global não é uma boa opção fazer divisões da mesma tarefa para diferentes aglomerados de computadores, pois suas localizações físicas podem ser distantes e as folhas impressas após o processamento ficariam espalhadas em locais distintos. Assim, o escalonador distribui tarefas inteiras e é capaz de lidar com as prioridades, restrições de tempo e outras necessidades que possam ser impostas pelos diferentes clientes às suas tarefas.

Com o intuito de prover informações necessárias para que o escalonador possa tomar suas decisões com maior eficiência, um mecanismo de previsão dedicado estima o tempo que cada nova tarefa ingressante na fila leva para ser rasterizada em cada um dos *clusters* presentes. Como entrada, este mecanismo obtém a custo da tarefa (obtido com as ferramentas citadas na Seção 2.3) e informações sobre recursos disponíveis nos *clusters*. A Figura 4.1 ilustra de forma simplificada o funcionamento do sistema, que contém três módulos: o escalonador propriamente dito, o sistema de busca de recursos e o mecanismo de previsão.

O escalonador mantém uma fila de tarefas ordenada por *deadlines*, na qual tarefas com maior urgência estão na frente e têm prioridade no escalonamento. Uma vez escalonada e alocado um

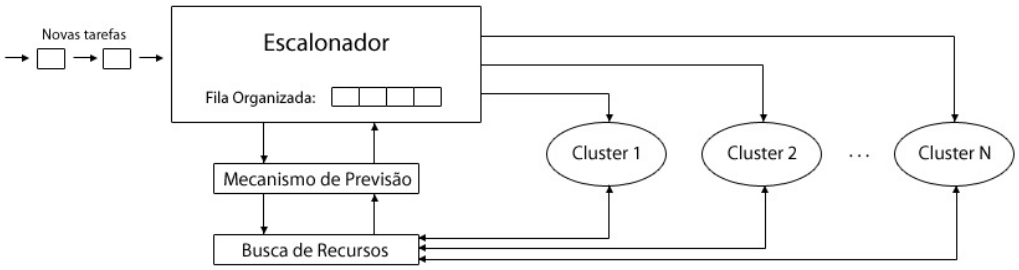

Figura 4.1: Arquitetura do Sistema Proposto

cluster para cada tarefa, ela será executada até o final no mesmo *cluster* onde iniciou. Novas tarefas já chegam com um valor de custo atribuído nas etapas anteriores pela ferramenta de *profiling* e podem, a qualquer momento, chegar ao escalonador a fim de serem rasterizadas. Assim, a fila de tarefas ordenadas armazenada pelo escalonador possui tamanho variável, crescendo quando aumenta a quantidade, velocidade ou a complexidade das tarefas que chegam e diminuindo quando estes atributos são reduzidos. O mecanismo de previsão, através da disponibilidade de recursos em cada *cluster* e do custo da tarefa, empregando uma metodologia baseada em análises estocásticas previamente realizadas e pequenos *benchmarks*, estima o tempo que a tarefa levará para ser processada em cada um dos *clusters*. Logo que uma tarefa chega no início da fila (próxima tarefa a ser escalonada), ela tem suas estimativas calculadas e, assim que recursos estiverem disponíveis, ela é imediatamente alocada, liberando uma posição de fila de processos em espera e entrando em execução. A disponibilidade de recursos é adquirida a partir de um sistema mestre/escravo de busca de recursos, que faz uma varredura em todas as máquinas sempre que solicitado pelo mecanismo de predição.

Nas seções seguintes, são tratados, então, a busca de recursos, o mecanismo de previsão e o escalonador, relembrando que os testes e resultados somente aparecem no próximo capítulo (Capítulo 5), onde todos estão compilados.

4.1 Busca de Recursos

A busca de recursos, também conhecida como descoberta de recursos, corresponde a um mecanismo que coleta informações sobre recursos disponíveis em um computador ou um aglomerado deles. A partir dela, o mecanismo de previsão calcula o tempo necessário para fazer a rasterização de determinada tarefa em cada *cluster*. Na Seção 3.2 foram estudadas as principais arquiteturas destes mecanismos de busca de recursos, e descritas as vantagens e desvantagens de cada uma delas.

O SWORD oferece uma quantidade rica de funcionalidades e oferece a vantagem da alta tolerância a falhas, no entanto, os sistemas centralizados, se mostraram mais velozes para diversos casos, por não precisarem buscar informações em lugares distintos [20]. A questão da tolerância a falhas pode ser melhorada aplicando-se replicação de servidores ou sistemas de eleição. Como o *ripping* de PDFs é uma aplicação pouco amarrada, sem comunicação entre processos, não é tão relevante a

importância dada pelo SWORD às restrições de latência de rede entre nós ou entre redes de longa distância, bastando uma métrica simples para avaliar o custo de envio de tarefas.

As conclusões extraídas do estudo dos outros sistemas de descoberta e busca de recursos motivaram que o sistema desenvolvido fosse inspirado na arquitetura mestre/escravo de [24].

Ambas as aplicações mestre e escrava foram desenvolvidas em Java (versão 1.6) a fim de manter maior compatibilidade com os trabalhos anteriores. O ambiente de desenvolvimento adotado foi o *NetBeans* devido a sua interface gráfica intuitiva e sua vasta quantidade de ferramentas que tornam a programação mais veloz. As aplicações foram construídas no sistema operacional Linux Ubuntu, mas funcionam em outros ambientes (ver Seção 5.1). As aplicações utilizam *sockets* TCP para comunicar-se entre si. Logo, é necessário que uma porta específica (configurável nas duas aplicações) esteja aberta em todas as unidades de rasterização.

A aplicação escrava executa em todas as máquinas rasterizadoras de todos os aglomerados aos quais se intenciona monitorar. A aplicação é projetada para consumir o mínimo de processamento possível, uma vez que fica apenas aguardando uma solicitação por informações sobre seu estado. Assim que uma solicitação do mestre é recebida, ela coleta os dados, calcula os valores necessários e envia os resultados, sempre com garantia de entrega e possibilidade de reenvio.

A aplicação mestre é responsável por solicitar informações a respeito do estado das outras máquinas. Esta solicitação pode ser manual (através de um usuário), ativada por outra aplicação, ou continua em intervalos de tempo configuráveis. A aplicação mantém um arquivo chamado *machines.rdm* com informação a respeito dos *clusters* presentes, contendo o nome de cada *cluster* e o nome ou endereço IP de cada uma de suas máquinas. A Figura 4.2 mostra a aparência do arquivo.

```
--cluster1
maquina1
maquina2
maquina3
--cluster2
192.168.0.4
192.168.0.5
--clusterPantanal
RIP01
RIP02
RIP03
RIP04
```

Figura 4.2: Arquivo *machines.rdm*

Uma vez solicitada, a aplicação mestre abre uma *thread* para cada máquina descrita no arquivo. Cada *thread* solicita informação de um escravo. Após receber todas as respostas, as *threads* são finalizadas. A aplicação, então, gera estatísticas com somatórios dos recursos de cada *cluster*, e deixa as informações coletivas e individuais disponíveis ao usuário. Além disto, o mestre grava um arquivo XML para cada máquina consultada, contendo as últimas informações obtidas. A Figura 4.3 mostra a estrutura do arquivo XML.

Em adição às características físicas de cada unidade analisada, como arquitetura, processador, memória e disco, o XML carrega informações de versões de *software* (versão do *ImageMagick* na TAG *imagemagick*). Também com grande importância, o XML contém o tempo de rasterização

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<System>
   <GENERAL>
      <host_name>desktop4</host_name>
      <ip>192.168.0.4</ip>
      <date>Tue Apr 29 23:09:08 BRT 2010</date>
   </GENERAL>
   <OPERATIONAL_SYSTEM>
      <os_name>Linux</os_name>
      <os_kernel>2.6.31-14-generic</os_kernel>
      <os_arch>i386</os_arch>
   </OPERATIONAL_SYSTEM>
   <JAVA>
      <java_name>Java Platform API Specification</java_name>
      <java_vendor>Sun Microsystems Inc.</java_vendor>
      <java_version>1.6</java_version>
   </JAVA>
   <PHYSICAL_PROPERTIES>
      <MEMORY>
         <total_ram>2004</total_ram>
         <used_ram>770</used_ram>
         <used_by_apps_ram>286</used_by_apps_ram>
         <free_ram>750</free_ram>
         <free_for_apps_ram>1234</free_for_apps_ram>
      </MEMORY>
      <DISK>
         <disk_used>5228368</disk_used>
         <disk_available>2418872</disk_available>
         <disk_size>7647240</disk_size>
      </DISK>
      <PROCESSOR>
         <processor_model> Intel(R) Core(TM)2 CPU         T5300  @ 1.73GHz</processor_model>
         <processor_freq>1730</processor_freq>
         <number_of_cores>2</number_of_cores>
         <processor_idle_time>95.8</processor_idle_time>
         <total_number_of_processes>160</total_number_of_processes>
         <running_processes>2</running_processes>
      </PROCESSOR>
   </PHYSICAL_PROPERTIES>
   <SOFTWARE>
      <imagemagick>ImageMagick 6.5.1-0 2009-08-27</imagemagick>
      <pdf_rip_time>00:01:24,1</pdf_rip_time>
      <out_resolution>100</out_resolution>
   </SOFTWARE>
</System>
```

Figura 4.3: Arquivo XML

de um PDF de baixo custo (PDF pivô), que o escravo rasteriza para estimar a velocidade da máquina naquele instante, como forma de realizar um *micro-benchmarking*. Encontrar este valor é fundamental, pois será utilizado como base nos cálculos do mecanismo de previsão de tempo de rasterização. Este valor é encontrado na TAG *pdf_rip_time*. Na TAG *out_resolution*, está a resolução desejada para imagem de saída da rasterização, que deve ser informada ao escravo, para que ele a utilize como parâmetro de entrada do processamento do PDF pivô. No decorrer do trabalho, mais detalhes sobre estes valores serão discutidos.

4.2 Mecanismo de Previsão

O mecanismo de previsão, baseado nas informações sobre disponibilidade de recursos fornecidas pelo mecanismo de busca e nas informações sobre o custo das tarefas, é responsável por estimar o tempo que cada tarefa requerirá para executar nos *clusters* presentes. O escalonador utiliza esta informação nas suas tomadas de decisão.

Além de auxiliar no escalonamento, a previsão é uma funcionalidade diretamente acessível ao usuário. Quando for desejada alguma operação manual, na qual um usuário necessite conhecer

o tempo de processamento estimado para determinada tarefa, selecionando o *cluster* desejado e informando o custo da tarefa, o usuário obterá a estimativa solicitada.

Fazer previsão de tempo de execução de tarefas baseada na análise de código, como é realizado pelos trabalhos relacionados na Seção 3.3, não é uma opção possível na rasterização de documento, pois o tempo depende não só da aplicação, mas dos arquivos de entrada. O tamanho do arquivo de entrada também não tem relação com o tempo de execução. Os dados a serem utilizados para uma estimativa dependem de uma análise do perfil do documento e das técnicas e funcionalidades do PDF utilizadas em cada um. Além disto, trabalhos que investigam previsão de tempo dedicada à tarefas específicas não são encontrados facilmente na literatura, que dá maior ênfase a estratégias para estimar o tempo de tarefas de uso genérico.

Sendo assim, algumas variáveis importantes devem ser levadas em conta pelo mecanismo de previsão no cálculo de suas estimativas no contexto de documentos PDF. A resolução, o espaço de cores e o formato da imagem de saída são variáveis que devem ser informadas ao *software* rasterizador e que influenciam no tempo de processamento. O comportamento destas variáveis no tempo de processamento de documentos PDF também é estudado nesta seção.

Como mostrado na Seção 2.3, a relação custo x tempo de processamento é diretamente proporcional, tendendo a apresentar uma função linear. Esta função crescente deve aumentar seu ângulo em máquinas mais lentas e diminuir em máquinas mais velozes. A opção mais precisa para se obter uma estimativa de tempo seria através da aplicação de uma fórmula que considerasse o custo do documento e a configuração da máquina obtida com a busca de recursos. Sabe-se, no entanto, que as informações providas pela busca de recursos, como frequência do processador, quantidade de memória RAM, espaço disponível em disco e arquitetura do processador (32 ou 64 bits) não são suficientes para determinar o desempenho de uma máquina. Muitas outras variáveis influenciam com alta significância: memória cache e sua velocidade, frequência da memória RAM, velocidade de acesso à memória RAM, utilização de *dual channel*, arquitetura do processador, arquitetura da placa-mãe, *chipset*, barramentos, etc. A solução encontrada foi, antes de fazer a previsão, processar um PDF pivô na máquina que fará a rasterização, coletar o tempo de processamento naquele momento e utilizá-lo para estimar o tempo de processamento de um documento com custo maior. As informações de memória RAM, frequência e porcentagem de uso do processador são usadas para aumentar a precisão das estimativas.

Sabe-se, no entanto, que características de *hardware* como velocidade de memórias e discos, arquitetura e outros são especificações fixas, permanentes para cada máquina, enquanto ela não tiver seus componentes alterados. Em [29] é provado que, quando não há limitação de memória e disco, a execução de um processo em uma máquina é diretamente proporcional ao percentual de utilização do processador naquele momento (valor facilmente obtido através de qualquer ferramenta de monitoria de recursos computacionais). Sendo assim, fica a proposta para que, no futuro, esta constatação possa ser utilizada pelo mecanismo, a fim de reduzir a quantidade de vezes que o PDF pivô é rasterizado e aumentar a velocidade média das predições realizadas.

Quando se deseja encontrar a equação de uma reta, encontrar dois pontos pertencentes a ela é

a primeira opção. Alguns gráficos, no entanto, na relação custo x tempo de processamento de um documento, formam uma nuvem de pontos quando estão muito próximos. Além disto, se os pontos escolhidos fossem muito distantes, o sistema perderia muito tempo rasterizando os PDFs de teste e realizar a estimativa pode deixar de ser vantajoso. Torna-se, então, inviável conseguir escolher os pontos corretos para definir a reta. O gráfico da Figura 4.4 evidencia tais fatores, apresentando o tempo de rasterização de PDFs com diferentes custos e a sua linha de tendência.

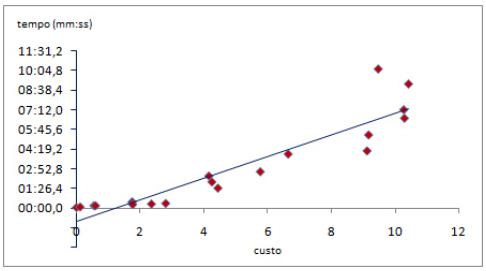

Figura 4.4: Custo x Tempo

A forma encontrada para gerar a equação da reta foi, através de testes de PDFs com custos variados executados em máquinas com diferentes recursos computacionais, estimar uma curva que relacione o coeficiente angular da reta em função do tempo de processamento de um PDF pivô de baixo custo de complexidade. Como os ambientes de virtualização de máquinas não permitem que seja modificada a frequência da máquina virtual [40] [41], os testes foram realizados em máquinas reais. A precisão das estimativas pode ser aumentada se novas máquinas, com diferentes configurações, forem adicionadas aos testes. Para a execução dos testes, foram selecionados 25 documentos retirados de clientes reais de grandes redes de impressão. O critério utilizado nesta seleção foi selecionar uma amostragem de documentos que apresentasse custos computacionais variados. A Tabela 4.1 mostra os PDFs selecionados e seus respectivos custos computacionais (fornecidos pelo *PDF Profiler* e pela aplicação das métricas).

Os PDFs apresentados na Tabela 4.1 foram rasterizados pelo RIP *Open Source ImageMagick-6.6.3-1-Q16 Converter* [42]. Em vista do longo tempo de execução dos processos mais complexos, cada execução foi realizada 5 vezes e todos os gráficos apresentados representam a média aritmética simples destes valores (variâncias abaixo de 0,5%). Os testes foram rodados com 100dpi e 300dpi (*dots per inch*) de resolução. Algumas das máquinas utilizadas nos testes são limitadas (pouca memória e disco) e não conseguiram rasterizar os documentos mais custosos. Quando utilizados 100dpi de resolução, mais documentos puderam ser testados e, por isto, somente estes testes foram selecionados para este trabalho. Embora as inclinações das retas sejam diferentes quando alterada a resolução da imagem de saída, as relações entre a inclinação das retas e o tempo de rasterização dos PDFs são as mesmas, não oferecendo impedimento algum para que as fórmulas sejam geradas a partir dos testes com 100dpi. A Figura 4.5 apresenta os gráficos gerados.

A determinação do melhor PDF pivô é fundamental. O PDF pivô deve ter baixo custo para que reduza o tempo da busca de recursos o máximo possível. O pivô, entretanto, também não pode ser muito leve, pois conforme observável nos gráficos da Figura 4.5, os PDFs mais leves não representam

Tabela 4.1: *Documentos Selecionados*

Nome do Documento	Custo
Transparent1Image1Page.pdf	0,024261
Alphatransp5images5overlaps.pdf	0,13883
Card.2.40Documents.40Pages.pdf	0,568068
Postcard.8.20Documents.40Pages.pdf	0,618342
Brochure.2.9Documents.108Pages.pdf	1,758585
Poster.2.70Documents.70Pages.pdf	1,770716
Poesy.Book.139p.pdf	1,780619
PhotoBook.notransp.60p.pdf	4,161904
Flyer1.80Documents160Pages.pdf	4,258732
Flyer2.100Documents200Pages.pdf	4,444371
Economy.Book.359p.pdf	5,77062
Flyer4.75Documents150Pages.pdf	6,643543
Letter8.90Documents450Pages.pdf	9,105436
Newsletter4.170Documents340Pages.pdf	9,155543
Card.1.400Documents.200Pages.pdf	9,454723
Letter6.130Documents390Pages.pdf	9,474335
Poster1.500Documents500Pages.pdf	10,2555
Flyer3.250Documents500Pages.pdf	10,27748
Postcard.6.500Documents.250Pages.pdf	10,40458
Newspaper.1.400Documents.400Pages.pdf	12,89868
Brochure6.300Documents600Pages.pdf	13,33834
Letter7.400Documents800Pages.pdf	21,05484
Newspaper2.150Documents.900Pages.pdf	23,33215
Brochure5.200Documents1000Pages.pdf	23,41858

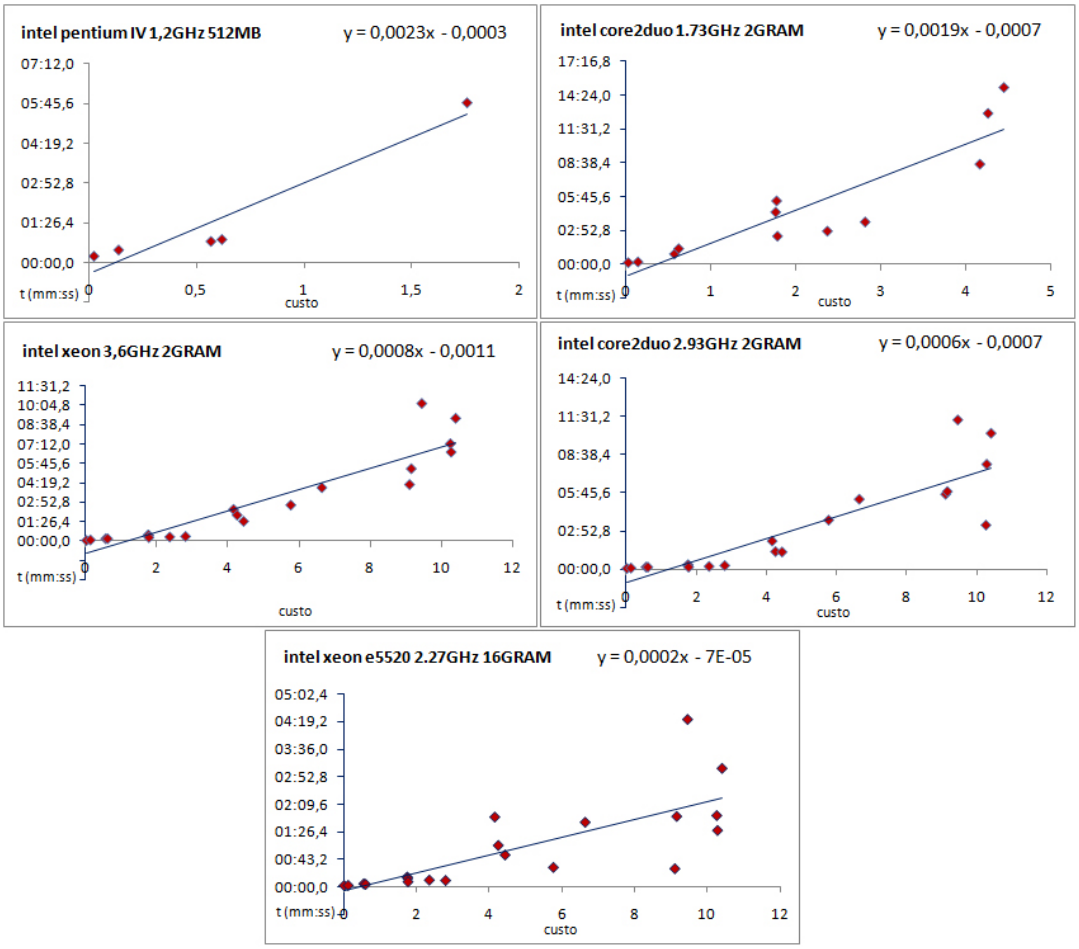

Figura 4.5: Curvas Custo x Tempo em Diferentes Máquinas

bem o comportamento da reta. O pivô escolhido foi o documento *Brochure.2.9Documents.108Pages*, por ter mantido um comportamento parecido em todos os testes e ter executado em todas as máquinas testadas.

Nos gráficos da Figura 4.5, o coeficiente angular das linhas de tendência cresce para as máquinas mais lentas. Utilizando os tempos de processamento do PDF pivô nas máquinas testadas, foi traçado um gráfico que relaciona estes tempos com os coeficientes angulares das retas de tendência. A curva gerada resultou em um polinômio de segunda ordem mostrado na Equação 4.1 (Figura 4.6), onde t é o tempo, em segundos, da rasterização do PDF pivô. Com este polinômio, a partir do tempo de rasterização do PDF pivô, pode ser encontrado o coeficiente angular da reta para determinada máquina de acordo com seu estado atual de utilização de recursos. Em posse do coeficiente angular e do tempo de rasterização do PDF Pivô, a equação da reta pode ser encontrada.

$$t = 8 * 10^7 CoefAngular^2 - 24873 CoefAngular + 8,8164 \tag{4.1}$$

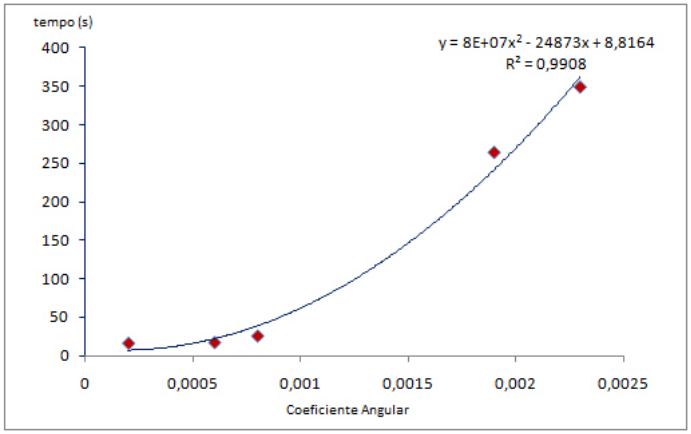

Figura 4.6: Polinômio

Uma vez estimada a equação da reta para determinada máquina, substitui-se na fórmula, a variável x da equação $ax + b$ da reta pelo custo do PDF a ser estimado e o valor de tempo será retornado. Esta reta, porém, ainda não engloba de forma correta os tempos associados aos PDFs de baixo custo. Conforme observável na Figura 4.4, principalmente nos pontos que representam PDFs de baixa complexidade, eles fogem da tendência dos demais e, se fosse aplicada a mesma equação dos demais, resultados até negativos poderiam ser retornados. Para corrigir estes valores distorcidos que possam aparecer neste grupo de documentos, uma nova reta é traçada pela aplicação com ponto inicial no tempo zero e custo zero (0,0) e final no ponto exato de localização do PDF pivô. Assim, conforme ilustrado na Figura 4.7, quando um PDF de custo menor que o pivô precisa ser calculado, a aplicação utiliza uma segunda reta para gerar o resultado.

Os valores obtidos como resultado até agora não levam em consideração os possíveis acréscimos de tempo provocados por variáveis como memória RAM disponível, espaço em disco e outras variáveis que se tornam mais influentes quando cresce o tamanho e a complexidade dos documentos. O *ImageMagick* utiliza bastante memória RAM e ela não pode deixar de entrar no cálculo da estimativa

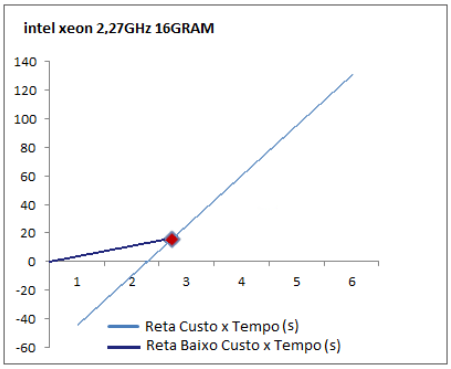

Figura 4.7: Reta Específica para PDFs de Baixo Custo

de tempo.

Na realização de testes, foi constatado que, quando a memória disponível para aplicações começa a se tornar limitada, o tempo de processamento da rasterização cresce exponencialmente. Conforme informações supracitadas, alguns documentos não chegam a concluir o processamento em máquinas com pouca memória. Entra, então, a necessidade da aplicação de uma fórmula de ajuste, visando aumentar ainda mais a precisão dos resultados. A fórmula considera o tempo de rasterização do PDF pivô, o custo do PDF para o qual se intenciona fazer a previsão e a quantidade de memória RAM disponível para aplicações (retornada pelo mecanismo de busca de recursos). A fórmula adiciona uma penalidade de tempo à estimativa, de acordo com as limitações de memória da máquina. No futuro, é interessante que informações sobre o disco sejam incluídas na fórmula, ressaltando que quanto menor a quantidade de memória RAM disponível, maior a necessidade do uso do disco rígido. Podem, ainda, ser agregadas informações de latência de rede e tempo gasto com operações de entrada e saída. A fórmula foi projetada de modo que, quanto menor for a memória RAM disponível, maior relevância ela dará ao acréscimo da penalidade de tempo. Quando houver memória disponível, a fórmula acrescenta valores desprezíveis ao resultado. O custo e o tempo de rasterização do pivô são diretamente proporcionais e a quantidade livre de memória RAM é inversamente proporcional à penalidade de tempo. Os coeficientes podem ser modificados se mais testes forem realizados (análise da relação entre custo e consumo de memória) e podem atingir valores ainda mais próximos dos reais. A fórmula pode ser visualizada na Equação 4.2, onde t é o tempo obtido como retorno da equação da reta estimada. Atualmente, esta fórmula representa apenas uma heurística a fim de indicar casos críticos que poderiam gerar falhas em sistemas com pouca memória. O espaço de possibilidades ainda não foi completamente explorado, porém, conforme os resultados apresentados na Seção 5.2, a aplicação desta fórmula não representou perda de qualidade aos resultados. Em trabalhos futuros, com uma avaliação mais precisa do comportamento do tempo de rasterização em função de limitação de memória, a fórmula poderá ser melhorada e conclusões mais concretas poderão ser extraídas da vantagem de sua aplicação.

$$TempoTotalEstimado = \frac{custoPivo * tempoPivo * 4 * 10^9}{memRAM^4} + t \tag{4.2}$$

Em relação à resolução da imagem de saída, foram também executados testes em diferentes máquinas, variando-se a resolução entre 10dpi e 4000dpi para diversos documentos. Todos os resultados apresentaram curvas exponenciais (exemplos na Figura 4.8). O resultado é muito impreciso quando se tenta interpolar e estimar uma curva exponencial com poucos pontos. A alternativa para contornar tal dificuldade é rasterizar o PDF pivô já na resolução de saída desejada.

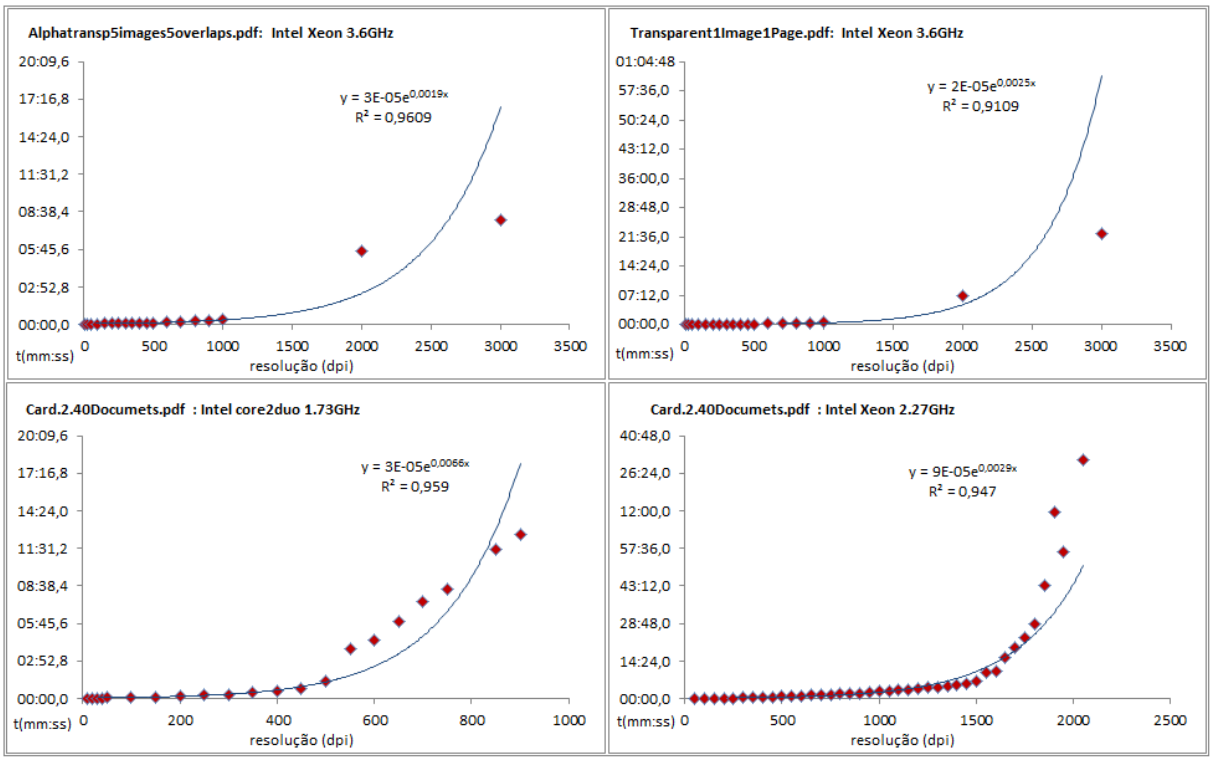

Figura 4.8: Curvas Resolução x Tempo

A extensão de saída da imagem rasterizada apresentou um comportamento padrão em todos os casos de teste. Na Figura 4.9, pode ser observada uma amostragem de 6 dos 34 documentos analisados, escolhidos por serem documentos com propriedades diversificadas. De cima para baixo, cada gráfico mostra os tempos de rasterização de um documento PDF, respectivamente, para os seguintes formatos de saída: FIG, BMP, TIFF, PNG e JPG. Na Figura 4.10 é apresentado um gráfico contendo a média de tempo, por formato de saída, de todos os documentos analisados. Desconsiderando o bmp, que é uma imagem de matriz de bits, as outras extensões possuem diferentes tipos de compressão, que utilizam diferentes algoritmos e geram arquivos com tamanhos diferentes para cada imagem. Diferentemente do esperado (tempos variados para cada formato), o ImageMagick levou o mesmo tempo na rasterização dos formatos, com exceção do png, que os superou.

É compreensível que os documentos no formato PNG (*Portable Network Graphics*) levem mais tempo que os demais, pois é um formato novo (1996), com algoritmos de compressão mais modernos e complexos, que mesmo com alta taxa de compactação, preservam mais as propriedades originais da imagem que os anteriores. Este formato oferece ainda uma vasta quantidade de opções de transparência que não estão presentes nos demais, permitindo, por exemplo, que seja retirado ou substituído os planos de fundo das imagens.

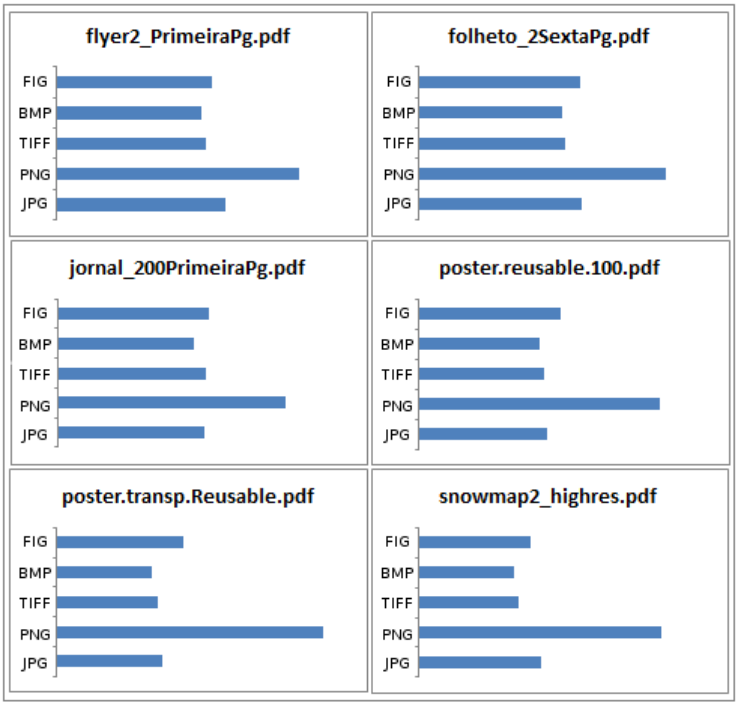

Figura 4.9: Tempo de Rasterização em Função do Formato de Saída

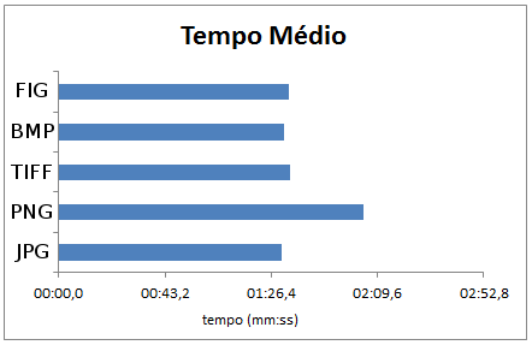

Figura 4.10: Tempo Médio de Rasterização em Função do Formato de Saída

Tomando como base o JPG(*Joint Photographic Experts Groups*), formato amplamente difundido e padrão do ImageMagick, o png ultrapassou, em média 36,7% do tempo de sua rasterização nos documentos analisados. Os valores oscilaram entre 20% e 70%, sendo que 21 dos 34 documentos ficaram entre 25% e 50%. Desta forma, pode-se considerar como um primeiro valor a ser utilizado na estimativa, o acréscimo de 35% do valor encontrado com os métodos descritos até agora, quando o formato de saída do arquivo desejado pelo cliente for PNG.

Para analisar a influência do espaço de cores de saída no processamento do documento, 52 PDFs foram rasterizados (10 vezes cada) em 100dpi e 300dpi com 9 espaços de cores diferentes: YUV, YPbPR, YCbCR, XYZ, sRGB, RGB, Gray, CMYK e CMY. Dos 52 PDFs, 33 apresentaram comportamento similar ao da Figura 4.11. 15 PDFs apresentaram comportamento similar ao da Figura 4.12 e outros 4 apresentaram o comportamento da Figura 4.13.

Nos PDFs representados pela Figura 4.11, os espaços de cores YUV, YPbPR, YCbCR e XYZ

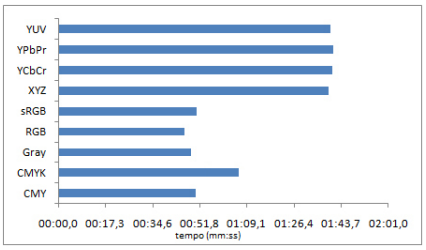

Figura 4.11: Comportamento de 64% dos Documentos

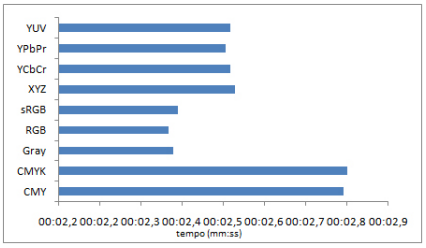

Figura 4.12: Comportamento de 28% dos Documentos

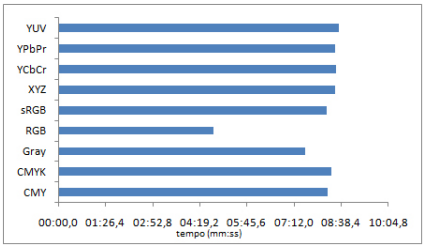

Figura 4.13: Comportamento de 8% dos Documentos

apresentaram sempre o tempo de processamento parecido e significativamente mais alto que os sRGB, RGB, Gray, CMYK e CMY, os quais também apresentaram tempos similares. O RGB foi sempre o espaço de cor mais rápido para rasterizar e, nos grupos mais rápidos, o CMYK foi sempre o mais lento. Os PDFs que apresentaram este comportamento são documentos com um grau de complexidade mais elevado, levando tempos na ordem de minutos para processar. Os PDFs representados pela Figura 4.12 são documentos com tempo de rasterização baixo (segundos), o que sugere que os espaços de cores CMYK e CMY levam mais tempo para inicializar. Já os PDFs representados pela Figura 4.13, são documentos complexos, com muitas fotografias e recursos de transparência, que levaram mais de 10 minutos para rasterizar. A fim de compreender o motivo deste comportamento diferenciado, foi adicionada ao *Profiler* a capacidade de identificar a presença de *SoftMasks* (objetos especiais de transparência que ainda não são levados em conta no custo dos documentos). Estes documentos, no entanto, não continham este objeto. Permanece ainda a possibilidade de existirem outros objetos que não estão nas métricas, ou alguma característica específica da criação destes documentos que esteja causando este comportamento.

Os documentos da Figura 4.12 possuem custo abaixo de 5, e 80% deles têm custo abaixo de 1,4. O aumento médio de tempo para os espaços de cor CMYK e CMY foi de 48.3% em relação ao espaço de cor padrão RGB. Em consequência deste resultado, foi considerado um aumento de

50% do tempo de rasterização para documentos com custo abaixo de 1,5, quando o esaço de cor for CMYK ou CMY. Para imagens geradas em outros espaços de cores, para custos menores que 1,5, o valor permance inalterado.

O aumento relativo ao RGB para os documentos da Figura 4.11 foi, em média, de 27,6%. Assim sendo, foi considerado um aumento de 30% em relação ao RGB para os espaços de cores YUV, YPbPR, YCbCR e XYZ, quando os custos forem maiores ou iguais a 1,5. Os espaços de cores restantes permanecem com seus valores inaltaredos.

Resumindo, ao executar o mecanismo de previsão, as seguintes etapas são executadas:

1. Uma solicitação de busca de recursos é enviada à aplicação mestre, contendo informações sobre a resolução de saída desejada para o PDF que deseja estimar;

2. O sistema de busca de recursos escravo rasteriza o PDF pivô (formato de saída JPG e espaço de cor RGB) na resolução recebida pela aplicação mestre, coleta o seu tempo de rasterização e o envia de volta juntamente com as outras variáveis presentes no XML do sistema de busca de recursos;

3. O mecanismo de previsão coleta os dados fornecidos pela busca de recursos;

4. O tempo (em segundos) da rasterização do PDF pivô recebido pelo sistema de busca é aplicado no polinômio e o coeficiente angular da reta é estimado;

5. Possuindo o coeficiente angular e o tempo de rasterização do pivô, a equação da reta $ax + b$ é montada;

6. O custo do documento ao qual se deseja estimar o tempo é aplicado na equação da reta;

7. Ao tempo estimado, é aplicada a fórmula de ajuste que utiliza a memória RAM disponível na máquina rasterizadora para adicionar uma penalidade de tempo ao valor encontrado;

8. Quando as configurações de saída da imagem fugirem do padrão, as correções relativas ao espaço de cor e formato de saída são, então, aplicadas;

9. A estimativa de tempo está pronta, e pode ser coletada por um usuário ou pela aplicação escalonadora.

4.3 Escalonador

Conforme retratado na Seção 3.4.2, não foram encontrados na literatura científica que trata especificamente de algoritmos de escalonamento, trabalhos que possuam otimizações de escalonamento nas mesmas configurações necessárias para o tipo de escalonamento que está sendo proposto: execução em sistemas heterogêneos, dinâmico, não-preemptivo, tarefas esporádicas, sem dependências, com valores de custo e prioridade previamente atribuídos, baseado em mecanismo de predição

dedicado e preocupado com o cumprimento de *deadlines*. Os algoritmos existentes utilizam técnicas de inteligência artificial baseadas em estudo de casos passados. Com inteligência artificial é necessário tempo para que o algoritmo aprenda qual o melhor escalonamento para aquele sistema. Esta aprendizagem pode ser realizada durante a execução, ou em um período anterior, antes de colocar o algoritmo para funcionar realmente. Além da complexidade de implementação, quaisquer eventuais modificações realizadas ao sistema implicam em um novo período de aprendizagem ou gasto de tempo até que o sistema atinja um nível adequado de eficiência novamente. Poderiam também, ser utilizadas abordagens de força bruta, tentando achar a melhor solução possível para a presente lista de tarefas em um determinado período de tempo. Estas abordagens, para filas grandes, ou não achariam uma resposta adequada no tempo solicitado, ou levariam muito tempo até achar uma solução que possa ser considerada boa. O algoritmo de escalonamento desenvolvido aqui pretende resolver o problema do escalonamento através de um algoritmo fixo, simples e que se adapta rapidamente a qualquer ambiente heterogêneo no qual as tarefas se enquadrem no perfil correto.

O desafio das casas de impressão está em aumentar a eficiência da produção com os recursos disponíveis e satisfazer os clientes, entregando o produto o mais rápido possível, sem estourar os prazos e respeitando as urgências impostas pelos clientes às suas tarefas. Assim, é importante que o algoritmo trabalhe com *deadlines*, sendo necessário herdar características dos algoritmos de escalonamento de tempo real [43].

Dos algoritmos de escalonamento de tempo real, o EDF (*Earliest Deadline First*) se mostrou o algoritmo ótimo para escalonamento dinâmico preemptivo em sistemas com um único processador [44]. Embora apresente bons resultados para escalonamento não-preemptivo, o escalonamento torna-se NP-difícil e pode apresentar algumas deficiências em casos particulares [45].

Dado o cenário apresentado, foram criados os algoritmos de escalonamento que se propõem a modificar o algoritmo de EDF para trabalhar em ambientes multiprocessados heterogêneos. Nas subseções seguintes serão descritos seis algoritmos dispostos em ordem didática para que possam ser compreendidos com mais facilidade.

A fim de testar os algoritmos e compará-los, foi desenvolvido, também em Java, um simulador de escalonamento. Os simuladores existentes como [46] e [47] permitem a personalização de tarefas, mas não permitem modificações aos algoritmos já implementados de forma simples, nem oferecem suporte a sistemas heterogêneos. O simulador desenvolvido implementa integralmente os algoritmos de escalonamento e, devido a sua capacidade de controlar o tempo de simulação, gerar estatísticas e outras funcionalidades, acaba se tornando mais complexo que o escalonador real. Apenas utilizando o núcleo do simulador, substituindo o nome das máquinas virtuais pelos nomes dos servidores de cada *cluster* e desenvolvendo um mecanismo de envio físico de tarefas pela rede, o escalonador está pronto para execução.

O simulador possui dois modos de funcionamento. Ele pode funcionar com tarefas especificadas manualmente, através de um arquivo, ou podem ser fornecidos alguns parâmetros como entrada, deixando o simulador se encarregar de gerar as tarefas. Para que o simulador gere as tarefas

automaticamente, deve receber como entrada os seguintes parâmetros:

- quantidade de tarefas;
- quantidade de unidades de processamento;
- limites mínimo e máximo, em segundos, dos *deadlines* (período máximo desejado para término de uma tarefa desde o momento da sua chegada);
- limites mínimo e máximo, em segundos, do momento de chegada das tarefas;
- limites mínimo e máximo, em segundos, de tempo de previsão das tarefas em cada unidade de processamento.

Com estes parâmetros, o simulador gera as tarefas com valores aleatórios uniformemente distribuídos entre os limites configurados. Este modo foi utilizado na realização dos testes apresentados no Capítulo 5. A entrada manual de tarefas tem o objetivo de permitir a visualização do funcionamento dos algoritmos a fim de exemplificar e comprovar seu funcionamento. Na Figura 4.14 pode ser observada a aparência do arquivo *tasks.ssim*, no qual estão descritas as tarefas.

```
start
0 10 5 5 6 7 8
1 20 4 25 22 12 13
2 20 1 15 12 20 21
3 18 1 26 11 12 10
4 15 2 20 20 20 20
5 15 2 10 9 10 4
6 15 3 10 10 10 10
7 30 1 14 31 17 18
8 14 5 30 16 19 18
```

Figura 4.14: Arquivo *tasks.ssim*

Após o "start", cada linha do arquivo descreve uma tarefa. Cada uma das 9 tarefas presentes neste exemplo são descritas, respectivamente, conforme os seguintes atributos: identificador único, *deadline* (s), tempo de chegada da tarefa (s), tempo previsto para executar na unidade P0 (s), tempo previsto para executar na unidade P1 (s), tempo previsto para executar na unidade P2 (s), e assim sucessivamente.

A tabela 4.2 contém a descrição das tarefas da Figura 4.14, as quais são as tarefas utilizadas na demonstração dos algoritmos. Claramente, apenas por observação da Tabela, algumas tarefas nunca cumprirão *deadline*. As tarefas foram geradas aleatoriamente, com objetivo único de demonstrar o funcionamento correto dos algoritmos.

Ao final da simulação, o simulador apresenta um gráfico com o resultado e gera um arquivo contendo estatísticas da simulação como, tempo de simulação, tempo real de execução do aplicativo, quantidade de *deadlines* atingidos, quantidade de processos executados por cada processador e porcentagem média de ultrapassagens de *deadlines* em relação ao tamanho total da tarefas. Ambos os recursos são utilizados nas subseções seguintes para exemplificação do funcionamento de cada um dos algoritmos.

Tabela 4.2: *Tarefas Geradas para Simulação*

Tarefa	Chegada	*Deadline*	Tempo Est. P0	Tempo Est. P1	Tempo Est. P2	Tempo Est. P3
0	5	10	5	6	7	8
1	4	20	25	22	12	13
2	1	20	15	12	20	21
3	1	18	26	11	12	10
4	2	15	20	20	20	20
5	2	15	10	9	10	4
6	3	15	10	10	10	10
7	1	30	14	31	17	18
8	5	14	30	16	19	18

4.3.1 Heterogeneous FIFO

O HFIFO (*Heterogeneous First In First Out*) é a abordagem mais primitiva e simples possível para a realização do escalonamento. Trata-se um algoritmo que mantem uma fila com as tarefas que chegam e escalona a tarefa do início da fila para a primeira máquina disponível, sem preocupar-se com *deadlines* ou estimativas de tempo.

Para exemplificar, utilizando como entrada as mesmas tarefas da Tabela 4.2, o resultado obtido pode ser visualizado na Figura 4.15.

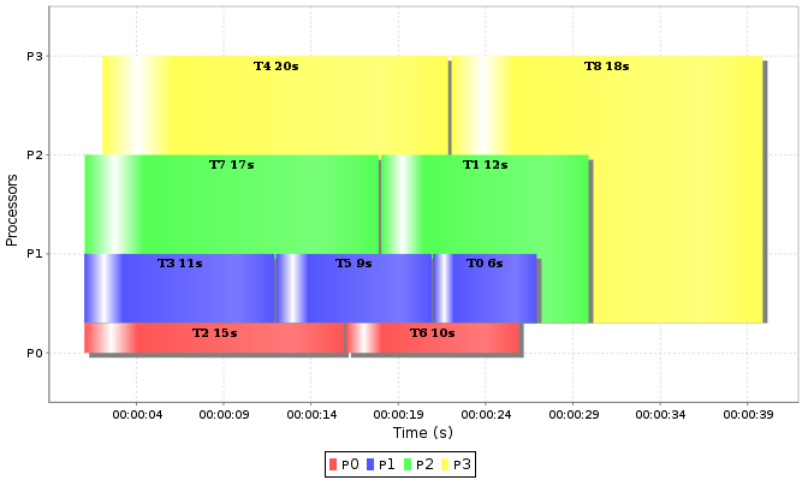

Figura 4.15: Resultado HFIFO

Neste exemplo, apenas 3 das 9 tarefas (33%) foram completadas dentro de seus *deadlines*. Das tarefas que ultrapassaram seu *deadline* (Tarefas 0, 1, 4, 5, 6 e 8), elas ultrapassaram, em média, 85% de seu *deadline* nominal. O tempo total de simulação foi de 52 segundos e o tempo real de processamento do simulador foi de 0,007 segundos.

4.3.2 Heterogeneous EDF

Neste algoritmo, quando se torna grande o número de tarefas em espera na fila do escalonador (não é o caso deste exemplo), o algoritmo tende a apresentar um resultado melhor que o HFIFO, pois ele mantem a fila ordenada de forma decrescente segundo o *deadline* das tarefas. Desta forma,

as tarefas com mais urgência na fila, tendem a ser atendidas primeiro. Sempre que uma nova tarefa chega na fila, ela é reorganizada de modo que a fila esteja sempre em ordem decrescente de *deadlines*. Para o mesmo grupo de tarefas da Tabela 4.2, o resultado pode ser observado na Figura 4.16.

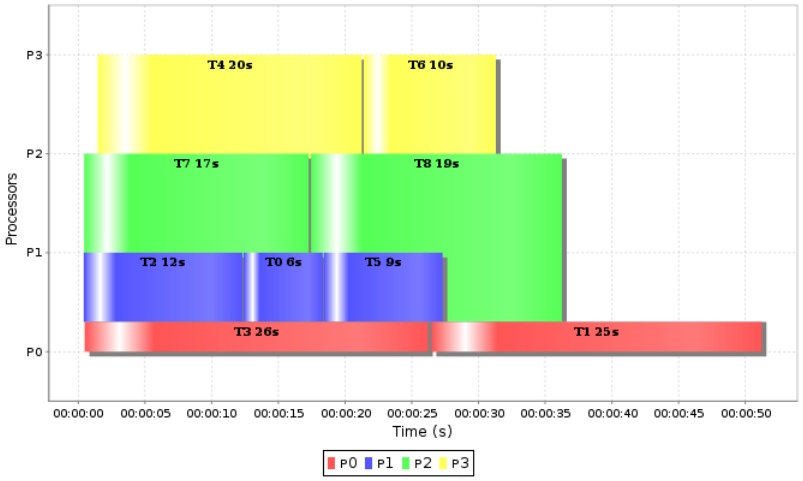

Figura 4.16: Resultado HEDF

Das 9 tarefas, apenas 2 completaram seus *deadlines* (22%). Das tarefas que ultrapassaram seu *deadline* (Tarefas 0, 1, 3, 4, 5, 6 e 8), elas ultrapassaram, em média, 84% de seu *deadline* nominal. O tempo total de simulação foi de 52 segundos e o tempo real de processamento do simulador foi de 0,007 segundos.

4.3.3 Heterogeneous Greedy

O algoritmo HGreedy continua ordenando a fila em ordem descrescente de *deadlines*, mas agora ele passa a utilizar as previsões de tempo para tomar decisões mais otimizadas na escolha da unidade de processamento. Entre as estimativas de tempo para cada uma das unidades de processamento, o escalonador escolhe a máquina disponível mais veloz para executar tal tarefa. Por este motivo, o algoritmo é chamado de *Greedy* (guloso). O objetivo deste algoritmo é executar as tarefas da maneira mais rápida possível. Este algoritmo somente pode escolher a máquina mais veloz, quando houver mais de uma unidade de processamento livre, caso contrário, alocará a tarefa do início da fila para a primeira unidade que se liberar. Em um cenário onde a fila permanece bastante tempo vazia e chegam rajadas de tarefas, este algoritmo pode oferecer um rendimento interessante. É importante ressaltar que, apesar de na simulação as tarefas já chegarem na fila contendo as estimativas de tempo para executar em cada unidade, no escalonamento real, as estimativas vão sendo calculadas pelo mecanismo de previsão em ordem crescente de *deadlines*, ou seja da tarefa mais urgente para a tarefa menos urgente. Desta forma, as decisões que o escalonador toma para as próximas tarefas estão mais coerentes com estado atual do sistema, que pode apresentar ou não uma dinamicidade veloz. Também utilizando as tarefas da Tabela 4.2, o resultado da simulação pode ser observado

na Figura 4.17.

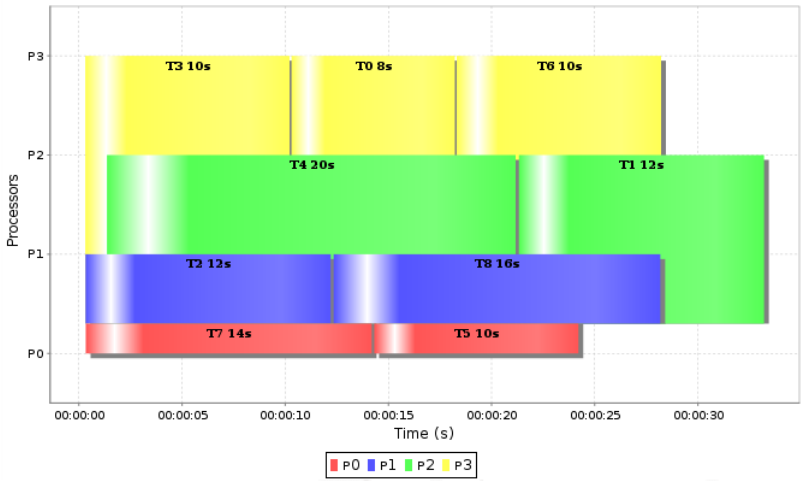

Figura 4.17: Resultado HGreedy

Nesta execução, 3 das 9 tarefas completaram seus *deadlines* (33%). Das tarefas que ultrapassaram seu *deadline* (Tarefas 0, 1, 4, 5, 6 e 8), elas ultrapassaram, em média, 68% de seu *deadline* nominal. O tempo total de simulação foi de 34 segundos e o tempo real de processamento do simulador foi de 0,008 segundos.

4.3.4 Heterogeneous Deadline Comitted

O algoritmo *Heterogeneous Deadline Comitted* utiliza as informações de previsão para tomar decisões mais inteligentes, com o objetivo de aumentar a taxa de cumprimento de *deadlines*. Este algoritmo pode ser mais lento na execução da fila de tarefas que o *Heterogeneous Greedy*, mas preocupa-se exclusivamente em atender as restrições de tempo das tarefas. Para alcançar tal objetivo, este algoritmo parte da suposição de que uma tarefa com maior urgência que todas as outras presentes na fila pode chegar a qualquer momento. A unidade de processamento escohida para executar a tarefa é a unidade mais lenta do grupo capaz de atender o *deadline* da tarefa. Dessa forma, o algoritmo deixa liberadas unidades de maior capacidade, para que elas possam atender eventuais tarefas com maior urgência que venham a chegar. Este algoritmo tende a apresentar melhor desempenho no cumprimento de *deadlines* quando a fila esvazia, e, de tempos em tempos, recebe rajadas de trabalho. No entanto, assim como o *Greedy*, quando a fila está cheia, após todas as unidades já terem iniciado seu trabalho, sempre que liberar um recurso, a primeira tarefa da fila será alocada para ele e o algoritmo passará a ter o mesmo comportamento do HEDF. O algoritmo somente apresenta vantagem na escolha da unidade de processamento adequada, quando mais de uma delas está liberada. Novamente, para o mesmo conjunto de tarefas apresentado na Tabela 4.2, o resultado do escalonamento pode ser observado na Figura 4.18.

Este algoritmo atendeu o *deadline* de 3 das 9 tarefas (33%). Das tarefas que ultrapassaram seu *deadline* (Tarefas 0, 1, 4, 5, 6 e 8), elas ultrapassaram, em média, 68% de seu *deadline* nominal. O

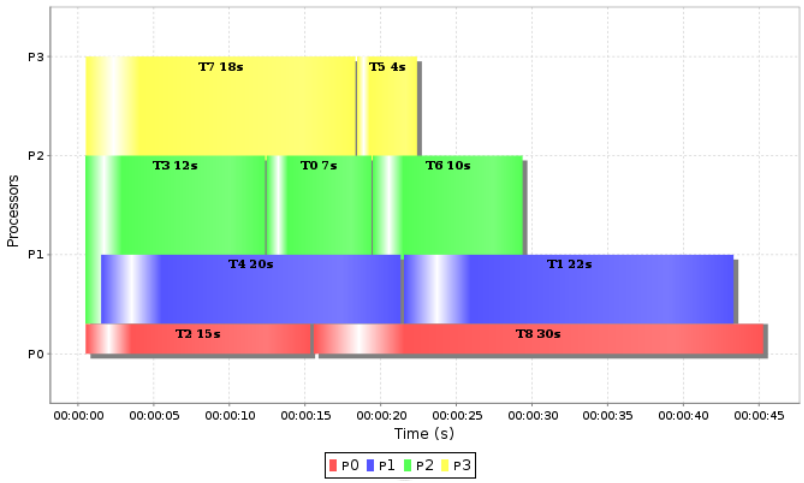

Figura 4.18: Resultado HDeadline Comitted

tempo total de simulação foi de 46 segundos e o tempo real de processamento do simulador foi de 0,009 segundos.

4.3.5 Heterogeneous Deadline Comitted Look Ahead

O algoritmo *Heterogeneous Deadline Comitted Look Ahead* tem o intuito de eliminar as desvantagens do *Heterogeneous Deadline Comitted*, que somente tem benefício quando houver mais de uma unidade livre. Neste sentido, na versão *Look Ahead*, quando há tarefas na fila e alguma unidade de processamento se torna disponível, ele olha para os processos em execução no momento e verifica se algum deles tem previsão de acabar a tempo de cumprir o *deadline* da primeira tarefa da fila (mais prioritária). Havendo algum dentro deste requisito, ele analisa a tarefa da segunda posição da fila. Pulando as unidades livres e a unidade enquadrada nos requisitos da tarefa da primeira posição da fila, o algoritmo procura por uma unidade capaz de atender o *deadline* da segunda tarefa da fila. Se a unidade for encontrada, o algoritmo passa para a análise da terceira tarefa da fila e segue assim até que não seja encontrada uma unidade capaz de atender o *deadline* dentre o grupo das unidades ocupadas. Quando este momento for atingido, a última tarefa analisada passará na frente de todas as outras e será alocada para o recurso livre. Mantendo tarefas prioritárias em espera quando houver possibilidade de cumprir com seus *deadlines* oferece chances para que tarefas que estão em desvantagem na fila e que, por precisarem esperar muito tempo, poderiam não cumprir seus requisitos de tempo, passarem na frente. Quando uma tarefa está em espera no final da fila, se existirem mais de uma unidade capaz de atendê-las quando acabarem seu processo atual, será escolhida a que demorará mais tempo, dando maiores chances de atendimento de requisitos para as outras tarefas. Através da aplicação desta técnica, o resultado esperado para este algoritmo é a redução do tempo médio que as tarefas ultrapassam seus *deadlines*, uma vez que tarefas mais urgentes continuam no início da fila, mas tarefas com probabilidade maior de estourarem seu *deadline* podem passar na frente. O resultado obtido com as tarefas da Tabela 4.2 pode ser observado na Figura 4.19.

O *Heterogeneous Deadline Comitted LA* atendeu o *deadline* de 4 das 9 tarefas (44%). Das

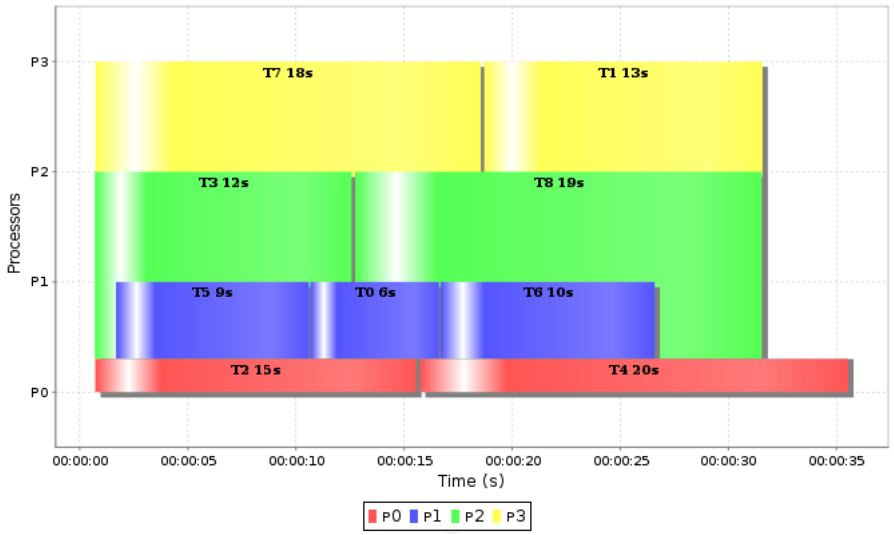

Figura 4.19: Resultado HDeadline Comitted LA

tarefas que ultrapassaram seu *deadline* (Tarefas 0, 1, 4, 6 e 8), elas ultrapassaram, em média, 69% de seu *deadline* nominal. O tempo total de simulação foi de 36 segundos e o tempo real de processamento do simulador foi de 0,011 segundos.

4.3.6 Heterogeneous Forced Deadline

O algoritmo *Heterogeneous Forced Deadline*, desconsiderando as prioridades das tarefas, sempre que um recurso estiver liberado, procura na lista pela primeira tarefa que cumpre *deadline* naquele recurso e a aloca para ele. Este algoritmo cumpre mais *deadlines* que os anteriores, mas, em vista do *trade off* implicito nos algoritmos propostos, ele tende a superar os outros na média de tempo ultrapassado em cada *deadline* perdido e atrasar a execução das tarefas mais prioritárias. O resultado encontra-se na Figura 4.20.

O algoritmo atendeu o *deadline* de 5 das 9 tarefas (55%). Das tarefas que ultrapassaram seu *deadline* (Tarefas 0, 4, 6 e 8), elas ultrapassaram, em média, 79% de seu *deadline* nominal. O tempo total de simulação foi de 35 segundos e o tempo real de processamento do simulador foi de 0,003 segundos.

4.3.7 Comparação entre os Algoritmos de Escalonamento

Na Tabela 4.3, podem ser comparados os resultados dos escalonamentos realizados nesta Seção. Apesar do *HForced Deadline* ter atendido a maior quantidade de *deadlines* ele obteve desempenho pior em relação ao tempo de ultrapassagem de seus *deadline* em relação aos algoritmos que implementam melhorias ao HEDF. Destes, o *HDeadline Comitted LA* apresentou resultado superior. O HFIFO, algoritmo de menor complexidade, apresentou o pior resultado, comprovando que a utilização de algoritmos mais complexos pode melhorar o sistema. Em relação ao tempo de simulação, por ser uma quantidade baixa de tarefas, permitindofazendo com que, na maioria das vezes, o escalon-

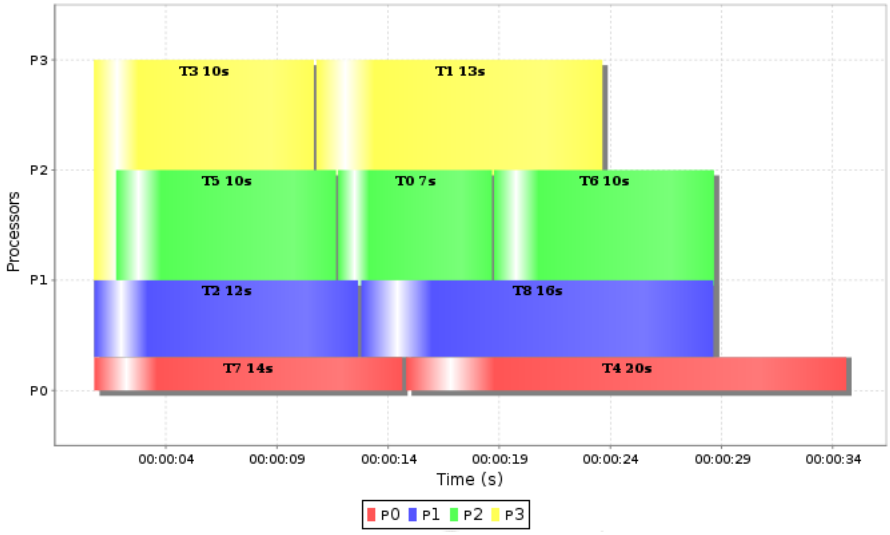

Figura 4.20: Resultado HForced Deadline

ador tenha opção de escolha entre as unidades de processamento disponíveis, o *Greedy* foi o mais rápido, por escolher sempre a máquina mais veloz. Do tempo real de processamento dos algoritmos, poucas informações podem ser extraídas destas execuções, por se tratar de uma quantidade reduzida de tarefas.

Tabela 4.3: *Comparação entre os Algoritmos*

Algoritmo	**Deadlines Atendidos**	**Estouro de Deadlines**	**Tempo de Simulação**	**Tempo de Processamento**
HFIFO	33%	85%	52s	0,007s
HEDF	22%	84%	52s	0,007s
HGreedy	33%	68%	34s	0,008s
HDeadline Comitted	33%	68%	46s	0,009s
HDeadline Comitted LA	44%	69%	36s	0,011s
HForced Deadline	55%	79%	35s	0,003s

5. TESTES, RESULTADOS E VALIDAÇÃO

Este capítulo aborda os testes realizados com cada uma das etapas do sistema proposto. São apresentados resultados de uma análise de estabilidade e velocidade do sistema de recursos, testes que examinam a qualidade do mecanismo de predição e testes gerais com os algoritmos de escalonamento propostos.

5.1 Busca de Recursos

O mecanismo de busca de recursos, nos testes de execução, funcionaram perfeitamente em todas as distribuições Linux testadas (versões lançadas a partir de 2009): Ubuntu, Fedora, Red Hat e Kurumin. As aplicações também foram testadas em ambiente *Windows*, no qual a aplicação mestre não apresentou falhas. A escrava, no entanto, apesar de reconhecer o sistema operacional, ainda precisa que novos métodos para coletar alguns dados do sistema operacional sejam implementados para que as mesmas informações possam também ser coletadas em ambiente *Windows*.

O mecanismo se mostrou estável e eficiente para até 5 escravos. Por funcionar com *threads* no servidor e por possuir baixa necessidade de cálculos, ele tende a continuar apresentando bom rendimento para um número maior de máquinas. O mecanismo foi mantido ativo com 5 escravos, coletando informações de todos a cada 30 segundos por 24 horas e se manteve estável durante todo o período. O tempo gasto para a coleta dos dados de uma única máquina aproxima-se da soma do tempo da comunicação, da amostragem para cálculo da utilização do processador (porcentagem de utilização) e do tempo de rasterização do PDF pivô. Eliminando o tempo de rasterização do PDF, que pode alterar significativamente o tempo de cálculo em cada máquina, o sistema foi testado com uma aplicação mestre variando o número de escravos entre 1 e 5, conforme mostra a Tabela 5.1. Como é um sistema centralizado, os valores de tempo tendem a crescer juntamente com o aumento da quantidade de máquinas, mas para o sistema em questão, o resultado é satisfatório. Os valores apresentados são resultado da média aritmética simples de dez execuções. A máquina que executou a aplicação mestre foi um Pentium Core 2 Duo 1,85GHz 2G RAM. As demais máquinas escravas eram Intel Pentium III 1,2GHz 512MB RAM.

Tabela 5.1: *Tempo da Busca de Recursos*

Quantidade de Escravos	1	2	3	4	5
Tempo de Coleta dos Dados (s)	2,8	2,9	3	3	3

5.2 Mecanismo de Previsão

A fim de testar e validar as predições efetuadas pelo mecanismo de previsão, juntamente com os PDFs já listados na Tabela 4.1, foram realizadas estimativas para uma nova amostragem de

documentos PDF e um novo conjunto de máquinas, diferentes daquelas utilizadas para gerar as equações do sistema. Logicamente, se esta nova amostragem de documentos fosse utilizada nas equações do sistema, ele estaria mais preciso, porém é necessário que uma amostragem diferente seja utilizada nos testes para provar que o método empregado funciona para qualquer conjunto de novos documentos. Diferentemente dos documentos utilizados na criação das equações (Tabela 4.1), que são documentos retirados de clientes reais de grandes casas de impressão, estes novos documentos são documentos gerados especificamente para estes testes. Os documentos foram criados em Java, com a bilioteca *iText* [48]. A diferença entre eles está na quantidade de imagens e suas resoluções, quantidade de transparência e tipos de transparência utilizados, presença de maior ou menor reusabilidade de objetos, entre outros. A lista dos documentos encontra-se na Tabela 5.2, onde estão o nome e o custo de cada PDF, arredondado para 6 casas decimais.

Tabela 5.2: *Segunda Amostragem de Documentos Selecionados*

Nome do Documento	**Custo**
folheto.1pgtexto.pdf	0,020511
flyer2.PrimeiraPg.pdf	0,032775
jornal.2PrimeiraPg.pdf	0,045038
jornal.PrimeiraPg.pdf	0,045038
cartanoticias.2PrimeiraPg.pdf	0,060644
cartanoticias.2SegundaPg.pdf	0,060644
folheto.2QuartaPg.pdf	0,060644
poster2.2PrimeiraPg.pdf	0,060644
folheto.2SextaPg.pdf	0,072920
cartao.2PrimeiraPg.pdf	0,104793
cartao.2SegundaPg.pdf	0,104793
snowmap2.highres.pdf	0,354833
poster.reusable.100.pdf	1,062842
poster.transparencyReusable.100.pdf	1,254141
carta3.50QuartaPg.pdf	1,348501
poster.2.70Documents.70Pages.pdf	1,770716
semtsemr1.pdf	2,487091
flyer2.100PrimeiraPg.pdf	2,791827
transp1.pdf	3,216068
transp5.pdf	3,641616
transp3.pdf	3,785343
cartanoticias.150TerceiraPg.pdf	4,185289
menostmaisr1.pdf	4,197170
maistmenosr1.pdf	4,295175
menostmaisr5.pdf	4,404109
cartao.100PrimeiraPg.pdf	4,999296
flyer3.250pgtexto.pdf	5,127750
maistmenosr4.pdf	5,172048
jornal.200PrimeiraPg.pdf	7,055303
letter8.90Documents450Pages.pdf	9,105436
folheto3.pdf	13,079724
cartanoticias2.pdf	17,673644
folheto2.pdf	19,087676
cartao2.pdf	19,087676

Os primeiros testes foram executados com documentos da primeira amostragem, mas utilizando máquinas com diferentes configurações. Conforme citado na Seção 4.2, diversos documentos não terminam sua execução em máquinas com limitações de memória. Em consequência disto, somente aparecem nos resultados as estimativas que puderam ser comparadas com um valor real de execução na máquina em questão. Cabe ressaltar que todos os valores apresentados nesta seção para o

tempo real de execução são médias aritméticas simples de 10 execuções. As casas decimais foram arredondadas para 3.

Na Tabela 5.3, são apresentados resultados obtidos através da comparação do tempo de rasterização real dos documentos com a estimativa calculada pela aplicação. Neste conjunto de testes foi utilizada a resolução 100dpi e formato de imagem de saída JPG. O computador utilizado foi um *AMD Athlon 64 FX 1,2GHz 1GRAM*.

Tabela 5.3: *Documentos da Primeira Amostragem: AMD Athlon 64 FX 1,2GHz 2GRAM*

Nome do Documento	Custo	Tempo Real (s)	Tempo Estimado (s)	Erro Relativo (%)
Transparent1Image1Page.pdf	0,024	6,036	4,079	32,008
Alphatransp5images5overlaps.pdf	0,138	9,7	4,384	54,804
Card.2.40Documents.40Pages.pdf	0,568	50,3	65,135	-29,493
Postcard.8.20Documents.40Pages.pdf	0,618	77,2	74,57	3,445
Poster.2.70Documents.70Pages.pdf	1,770	322	274,132	14,865
Poesy.Book.139p.pdf	1,781	141,252	274,432	-94,285
PhotoBook.notransp.60p.pdf	4,162	509,623	683,003	-34,021
Flyer1.80Documents160Pages.pdf	4,258	769,784	731,503	4,973
Flyer2.100Documents200Pages.pdf	4,444	900,282	781,504	13,193
Flyer4.75Documents150Pages.pdf	6,643	970,173	959,223	1,128
Letter8.90Documents150Pages.pdf	9,105	1413,664	1531,818	-8,258
Poster1.500Documents500Pages.pdf	10,255	1818,005	1729,286	4,880

Objetivando analisar a qualidade da previsão, a medida utilizada em [49] e nos outros trabalhos relacionados com predição para tarefas genéricas descritos no Capítulo 3.3, foi o erro relativo (Equação 5.1). Nestes trabalhos são geradas tarefas com valores de tempo dentro de uma faixa determinada (10 a 15 minutos, 20 a 30 minutos, ...). Nestes casos, o erro relativo apresenta bons resultados. Em [50], por exemplo, o erro relativo variou entre -50% e 50% nos casos de teste. No caso dos documentos PDF, esta medida nem sempre mostra o resultado desejado. O objetivo maior desta previsão é fornecer uma noção de ordem de grandeza ou domensão aproximada da tarefa e auxiliar o escalonador nas tomadas de decisão. Uma tarefa com tempo real de execução de 1 segundo, se for estimada para 2 segundos (erro relativo de -100%), por exemplo, obteve uma estimativa boa, capaz de auxiliar o escalonador e oferecer uma previsão de término para o cliente. Porém, se uma tarefa estimada para levar 10 horas, demorar 15 horas (erro relativo de 50%), este erro já pode fazer com que o sistema perca eficiência e o cliente fique insatisfeito com a estimativa.

Neste cenário, é necessário, portanto, dar atenção à ordem de grandeza dos valores e associar o erro relativo com a dimensão dos mesmos. O erro relativo para os resultados apresentados na Tabela 5.3 variou de -94% até 54%, com erro relativo médio (média aritmética simples dos erros relativos) de -2,9%. Apesar da variação grande, 8 dos 12 documentos ficaram entre -34% e 15%. O documento causador desta variação foi o Poesy.Book.139p.pdf, cujo tempo de execução não está coerente com seu valor de custo atribuído. Tal fator evidencia que as métricas podem ainda não funcionar bem para todos os tipos de documentos e necessitam de ajustes. Para os demais documentos, a estimativa sugeriu valores bastante satisfatórios.

$$ErroRelativo(\%) = \frac{tReal - tEst}{tReal} * 100 \tag{5.1}$$

Ainda para a primeira amostragem de documentos, a Tabela 5.4 apresenta os resultados para estimativas de rasterizações com 100dpi e formato JPG em uma máquina *Intel Core 2 Duo 1,86GHz 2GRAM*.

Tabela 5.4: *Documentos da Primeira Amostragem:Intel Core 2 Duo 1,86GHz 2GRAM*

Nome do Documento	Custo	Tempo Real (s)	Tempo Estimado (s)	Erro Relativo (%)
Transparent1Image1Page.pdf	0,024	2,000	0,233	88,350
Alphatransp5images5overlaps.pdf	0,138	3,501	1,334	61,897
Card.2.40Documents.40Pages.pdf	0,568	8,118	5,459	32,754
Postcard.8.20Documents.40Pages.pdf	0,618	8,912	5,942	33,326
Poster.2.70Documents.70Pages.pdf	1,770	18,400	18,986	-3,185
Poesy.Book.139p.pdf	1,781	7,893	19,344	-145,078
PhotoBook.notransp.60p.pdf	4,162	125,997	105,449	16,308
Flyer1.80Documents160Pages.pdf	4,258	78,498	108,950	-38,793
Flyer2.100Documents200Pages.pdf	4,444	76,665	115,662	-50,867
Flyer4.75Documents150Pages.pdf	6,643	315,697	195,182	38,174
Letter8.90Documents150Pages.pdf	9,105	338,281	284,201	15,987
Poster1.500Documents500Pages.pdf	10,255	198,250	325,786	-64,331
Flyer3.250Documents500Pages.pdf	10,277	474,243	326,581	31,136
Postcard.6.500Documents.250Pages.pdf	10,404	514,673	331,177	35,653

O último teste realizado para este conjunto de documentos foi em uma máquina participante da construção das equações (*Intel Xeon E5520 2.27GHz 16GRAM*), mas os resultados (Tabela 5.5) salientam comportamentos importantes.

Tabela 5.5: *Documentos da Primeira Amostragem: Intel Xeon E5520 2.27GHz 16GRAM*

Nome do Documento	Custo	Tempo Real (s)	Tempo Estimado (s)	Erro Relativo (%)
Transparent1Image1Page.pdf	0,024	2,632	0,222	91,565
Alphatransp5images5overlaps.pdf	0,138	2,967	1,271	57,162
Card.2.40Documents.40Pages.pdf	0,568	5,454	5,200	4,657
Postcard.8.20Documents.40Pages.pdf	0,618	4,601	5,661	-23,038
Poster.2.70Documents.70Pages.pdf	1,770	14,143	18,963	-34,080
Poesy.Book.139p.pdf	1,781	11,713	19,321	-64,953
Flyer2.100Documents200Pages.pdf	4,444	50,527	115,639	-128,866
Letter8.90Documents150Pages.pdf	9,105	29,196	284,178	-873,346
Card.1.400Documents.200Pages.pdf	9,454	262,420	296,808	-13,104
Poster1.500Documents500Pages.pdf	10,255	112,231	325,763	-190,261

Mesmo com erro relativo médio de 5,6%, os resultados da 5.5, apresentaram erros mais significantes que os resultados anteriores. Novamente os erros estão naqueles casos nos quais o custo não corresponde bem ao tempo de execução. A aplicação permanece utilizando a reta com a mesma angulação para todos os documentos estimados na máquina e os resultados são gerados a partir do custo de cada documento. O acerto na ordem de grandeza destes resultados já oferece vantagens na previsão.

Nos testes seguintes, os documentos utilizados passam a ser da segunda amostragem de PDFs e os computadores utilizados estão listados na Tabela 5.6. Na Tabela 5.7 estão os resultados para os documentos rasterizados com 100dpi e com formato de saída JPG. Por executarem todos documentos em todas as máquinas testadas, os resultados puderam ser compilados juntos na mesma tabela.

Tabela 5.6: *Computadores Utilizados na Segunda Amostragem*

Identificador	Configuração
C01	Intel Xeon E5520 2.27GHz 16GRAM
C02	Intel Core(TM)2 Quad CPU Q8200 2.33GHz 4GRAM
C03	Intel Xeon(TM) 3.6GHz 2GRAM
C04	AMD Opteron 246 2GHz 8GRAM

Tabela 5.7: *Resultados da Segunda Amostragem*

Nome.pdf	Custo	C01	Est. C01	C02	Est. C02	C03	Est. C03	C04	Est. C04
folheto.1pgtexto	0,021	0,276	0,193	1,621	0,210	0,873	0,297	0,442	0,331
flyer2.PrimeiraPg	0,033	0,423	0,309	0,496	0,335	0,613	0,475	1,602	0,529
jornal.2PrimeiraPg	0,045	0,612	0,425	0,646	0,461	1,286	0,653	1,132	0,727
jornal.PrimeiraPg	0,045	0,612	0,425	0,647	0,461	0,934	0,653	1,037	0,727
cartanots.2PrimeiraPg	0,061	0,720	0,572	0,628	0,620	1,098	0,879	1,243	0,979
cartanots.2SegundaPg	0,061	0,713	0,572	0,628	0,620	1,023	0,879	2,198	0,979
folheto.2QuartaPg	0,061	0,961	0,572	0,629	0,620	1,392	0,879	1,538	0,979
poster2.2PrimeiraPg	0,061	0,904	0,572	0,629	0,620	1,237	0,879	1,358	0,979
folheto.2SextaPg	0,073	1,739	0,688	1,000	0,746	1,465	1,057	1,760	1,177
cartao.2PrimeiraPg	0,105	3,763	0,988	2,412	1,072	3,313	1,519	3,753	1,692
cartao.2SegundaPg	0,105	2,554	0,988	2,412	1,072	3,185	1,519	3,609	1,692
snowmap2.highres	0,355	17,576	3,347	17,311	3,632	22,476	5,145	32,623	5,730
poster.reuse.100	1,063	66,721	10,026	62,156	10,879	94,888	15,411	102,248	17,164
poster.transpReuse.100	1,254	37,506	11,831	36,989	12,837	53,734	18,185	55,417	20,253
carta3.50QuartaPg	1,349	8,482	12,721	7,370	13,803	11,821	19,553	16,498	21,7761
poster.2.70Docs.70Pgs	1,771	15,216	18,963	16,618	18,125	21,0890	25,675	27,348	30,995
semtsemr1	2,487	29,824	44,866	26,700	48,284	44,200	64,478	52,200	70,028
flyer2.100PrimeiraPg	2,792	27,722	55,885	25,673	60,159	36,467	79,967	56,402	86,632
transp1	3,216	29,200	71,291	26,555	76,691	40,001	101,529	50,100	109,747
transp5	3,642	37,100	86,672	31,610	93,274	48,012	123,159	60,404	132,934
transp3	3,785	37,500	91,834	36,219	98,874	53,821	130,464	68,525	140,765
cartanots.150TercPg	4,185	41,211	106,271	39,650	114,460	55,700	157,792	71,400	162,556
menostmaisr1	4,197	36,523	106,701	36,800	114,923	53,400	151,396	66,400	163,204
maistmenosr1	4,295	42,051	110,245	41,701	118,742	58,300	156,377	74,800	168,543
menostmaisr5	4,404	42,101	114,184	39,058	122,987	56,100	161,914	71,300	174,479
cartao.100PrimeiraPg	4,999	115,900	135,705	118,200	146,180	148,700	192,166	182,907	206,908
flyer3.250pgtexto	5,128	24,400	140,350	15,300	151,186	41,600	198,695	62,566	213,907
maistmenosr4	5,172	46,599	141,951	44,500	152,912	66,500	200,946	84,648	216,321
jornal.200PrimeiraPg	7,055	93,332	210,048	87,600	226,299	118,500	296,666	149,611	318,932
letter8.90Docs450Pgs	9,105	49,686	284,178	36,400	306,189	358,600	400,869	91,734	430,636
folheto3	13,080	233,150	427,884	235,905	461,059	689,500	602,870	396,004	647,180
cartanots2	17,674	249,646	593,995	331,012	640,076	748,400	836,365	422,454	897,485
folheto2	19,088	387,927	645,125	502,446	695,178	1015,600	908,236	664,050	974,530
cartao2	26,824	1144,301	924,875	1213,606	996,663	1915,300	1301,470	1878,100	1396,073

A Tabela 5.7 apresenta resultados para um novo grupo de documentos, para cada umas das máquinas da Tabela 5.6. São apresentados o tempo de execução real e, na coluna seguinte, o tempo estimado, em segundos. Para todas as máquinas, o erro relativo médio foi entre -52% e -50%. A ordem de grandeza das estimativas foi bem respeitada para a maioria dos documentos. Os erros maiores novamente ocorreram em decorrência de documentos com custo e tempo de execução incoerentes.

As correções relativas ao formato da imagem e espaços de cores foram geradas com as duas amostragens de documentos. Assim, funcionariam sem problemas para os documentos apresentados no capítulo.

5.3 Algoritmos de Escalonamento

Os testes relativos aos algoritmos de escalonamento têm o objetivo de compreender os seus comportamentos a fim de identificar limitações, propor melhorias e avaliar suas qualidades. Nas curvas apresentadas na Figura 5.1, foram executados 21 escalonamentos, com cada um dos algoritmos propostos na Seção 4.3. As execuções foram feitas com tarefas de valores aleatórios, variando a quantidade de tarefas, para cada escalonamento, de 1 a 5000 tarefas. O *deadline* das tarefas varia entre 700 segundos e 50000 segundos. O tempo previsto para cada processador varia de 50 segundos até 500 segundos. As tarefas podem chegar na fila em tempos aleatórios uniformemente distribuídos entre 1 segundo e 5000 segundos do início da execução.

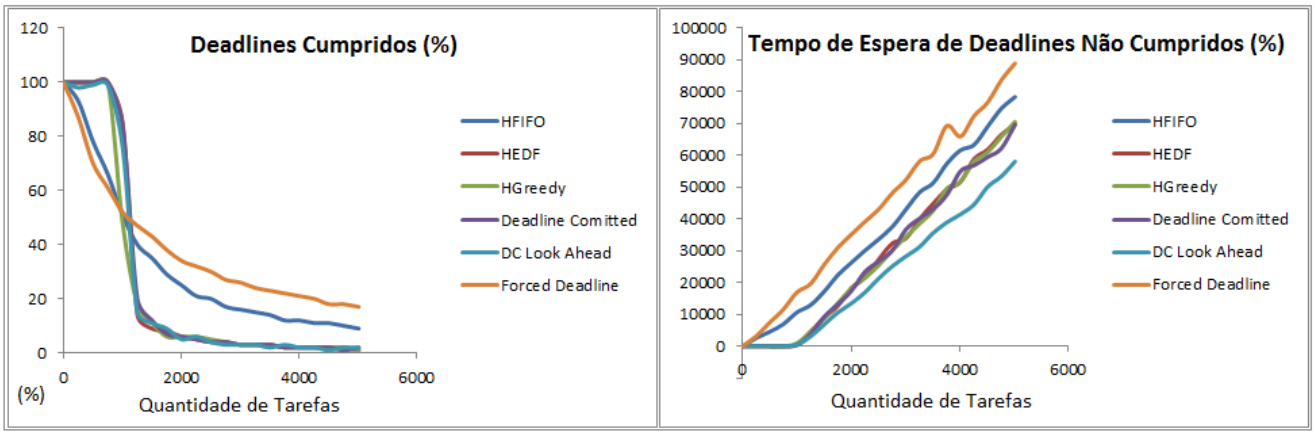

Figura 5.1: Comparações entre os Algoritmos de Escalonamento (a)

Como previsto, quando cresce o número de tarefas, o *Forced Deadline* cumpriu uma quantidade maior de *deadlines*, pois está sempre a procura de qualquer tarefa que possa cumprir sua restrição temporal. O início aparentemente melhor do *Greedy*, do *Look Ahead* e do *Deadline Comitted* também é esperado, uma vez que estes algoritmos apresentam bons resultados quando o escalonador possui mais de uma unidade de processamento livre e pode escolher o melhor recurso. Após a fila encher, seu desempenho cai. O HEDF é um algoritmo com baixa taxa de atendimento de *deadlines*, pois atende as tarefas mais urgentes (com menor *deadline*) primeiro e as tarefas menos urgentes, por esperarem muito tempo, acabam perdendo seus *deadlines*. Este efeito é conhecido no EDF e recebe o nome de efeito dominó [51]. O efeito dominó ocorre em escalonadores EDF com possibilidade de sobrecarga.

A fim de corrigir este efeito, foi introduzido o algoritmo de escalonamento TAFT (*Time-Aware Falut Tolerant*) [52]. Simplificando, este algoritmo adota políticas especiais para tarefas que ultrapassaram seus *deadlines* como se estivesse fazendo um tratamento de exceções. As tarefas que perderam seu tempo são escalonadas, por exemplo, com o algoritmo EDL (*Earliest Deadline as Late as Possible*) (inverso do EDF). Para auxiliar na compreensão deste algoritmo, toma-se por exemplo o algoritmo HEDF apresentado aqui. Se a fila for invertida, com os *deadlines* maiores na frente, uma quantidade maior de tarefas irá cumprir suas restrições. Em contrapartida, as tarefas mais

urgentes ficam para trás e não conseguem terminar. O TAFT trata-se de um forte candidato a ser investigando com maior profundidade e incorporado a estes algoritmos.

Conhecendo o efeito dominó, é possível agora, compreender o segundo gráfico da Figura 5.1. O *Forced Deadline*, apesar de cumprir uma quantidade maior de *deadlines*, quando os ultrapassa, o faz por muito mais tempo (tarefas prioritárias podem levar mais tempo para serem alocadas). Já o *Look Ahead*, que avalia com cuidado as melhores opções para as tarefas de maior prioridade, quando ultrapassa, ultrapassa por menos tempo.

Nas curvas apresentadas na Figura 5.2, as propriedades das tarefas continuaram sendo as mesmas, mas, desta vez, foi fixada quantidade de tarefas para 4000 e a quantidade de processadores variou entre 2 e 40. A análise destas curvas mostra que os 2 algoritmos que não implementam o EDF, tiveram um aumento mais suave na quantidade de *deadlines* cumpridos que os demais. Tal fator se deve ao fato de o EDF manter as tarefas com menor prazo na frente e alocá-las primeiro para os recursos disponíveis. Em sistemas que não geram sobrecarga, o EDF é o algoritmo de maior eficiência [52]. Na segunda curva, é possível observar que o *Look Ahead* obteve o menor tempo de espera e o *Forced Deadline* obteve o maior tempo (comportamento esperado em razão de todos os fatores já discutidos a respeito do efeito dominó).

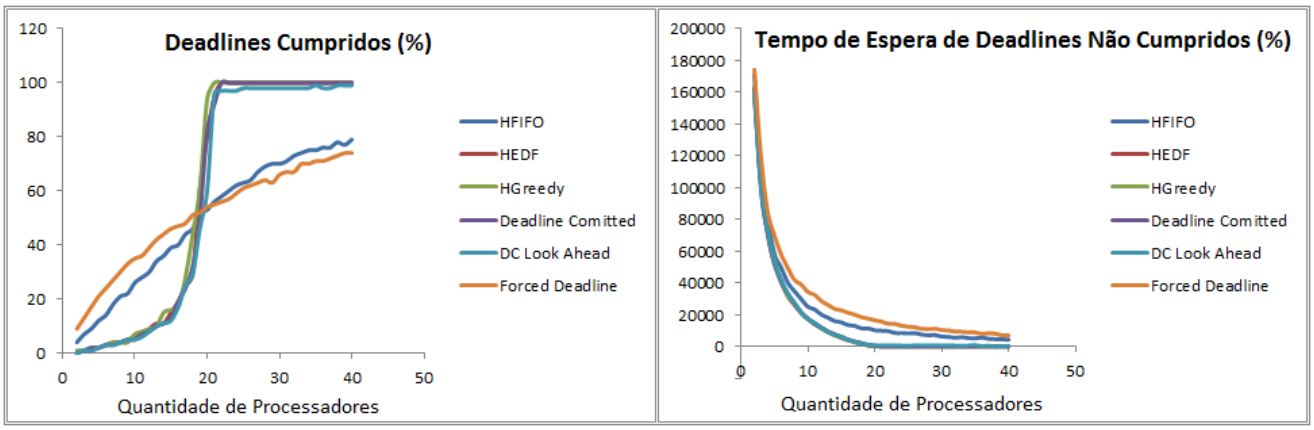

Figura 5.2: Comparações entre os Algoritmos de Escalonamento (b)

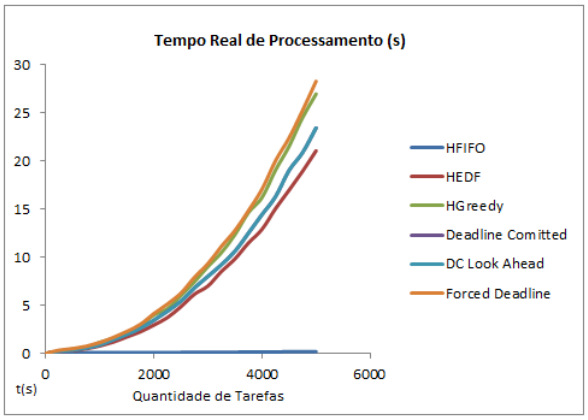

Figura 5.3: Tempo de Processamento dos Algoritmos

Visando observar a complexidade dos algoritmos, foram traçadas curvas (Figura 5.3) que relacionam o tempo real da simulação com a quantidade de tarefas para cada algoritmo. O HFIFO, que não implementa nenhum tipo de inteligência, apresentou tempo desprezível em relação aos outros. O HEDF é o próximo algoritmo em tempo de execução, seguido pelos *Look Ahead* e *Greedy*, os quais já possuem um pouco mais de complexidade. O *Forced Deadline*, provavelmente em razão de precisar deslocar-se na fila pesquisando por uma tarefa ábil a cumprir *deadline*, apresentou o maior tempo de simulação.

Os algoritmos possuem comportamentos diferentes, e podem ser adaptados conforme a necessidade do usuário. Os testes esclareceram bem as diferenças entre eles, vantagens e desvantagens de cada um.

6. CONCLUSÃO E TRABALHOS FUTUROS

Após soluções baseadas na análise do perfil de documentos, que apresentaram ganho de desempenho através da aplicação de diferentes estratégias para escalonamento de tarefas dentro de um *cluster*, este volume descreveu uma solução para um escalonador global que otimiza a distribuição de tarefas entre os diversos *clusters* responsáveis pela rasterização de documentos em casas de impressão. Além das análises a respeito do perfil dos documentos, o escalonador proposto conta com informações providas por dois outros sistemas auxiliares: o mecanismo de previsão e a descoberta de recursos.

A descoberta de recursos foi completamente implementada e funcionou de forma estável em todos os testes realizados. A utilização de *threads* no servidor garante velocidade para o sistema mesmo quando cresce o número de serviços escravos.

Nos testes realizados, o mecanismo de previsão se mostrou uma ferramenta capaz de prover estimativas com acurácia aceitável para auxiliar o escalonador, mantendo a maioria dos resultados dentro da mesma faixa de erro obtida pelos demais trabalhos relacionados com previsão. Os erros relativos foram baixos, salvo as variações ocorridas em função das métricas, que funcionaram muito bem para a maioria dos documentos, mas em alguns casos apresentaram fuga da reta tendencial custo x tempo. Através da realização de testes, foram encontrados comportamentos que tornaram possível calcular as previsões de tempo. Foi detectado comportamento exponencial para o aumento da resolução das imagens de saída e foi interpolada uma curva polinomial de segunda ordem capaz de estimar o coeficiente angular da reta que relaciona o custo de um documento PDF com seu tempo de rasterização de acordo com o desempenho atual de cada máquina rasterizadora. Comportamentos para a mudança do espaço de cores e extensão das imagens de saída também foram analisados e identificados. Após todas as etapas, uma fórmula aproxima ainda mais o resultado da previsão com a realidade, relacionando-o com a disponibilidade de memória RAM da máquina.

Os algoritmos de escalonamento sugeridos melhoram a eficiência e o atendimento de requisitos, em relação às estratégias de escalonamento mais primitivas. Dentre os algoritmos, de acordo com o cenário preferido, ou com as necessidades de cada usuário, algum pode adaptar-se melhor. Uma proposta híbrida pode ser investigada no futuro, fazendo com que o escalonador adapte o algoritmo dinamicamente em função do estado atual da fila e casos passados. Mais estudos, no entanto, ainda se fazem necessários. Solucionar o problema do efeito dominó no HEDF através do TAFT pode aumentar bastante a eficiência e confiabilidade do escalonamento.

O presente trabalho propôs uma solução inicial para a automação de sistemas distribuídos de impressão através de escalonamento heterogêneo de tarefas. A solução envolve muitas etapas e o objetivo não foi esgotar todos os testes necessários para cada uma delas, nem considerá-las como soluções ótimas, mas mostrar que a solução é válida, gera bons resultados, que é possível ser melhorada, e que seu emprego pode trazer ganhos reais. Cada uma das etapas abre um novo ramo que engloba novos estudos e diferentes abordagens. Além disto, a metodologia utilizada na

construção do mecanismo de previsão é perfeitamente extensível à outros tipos de tarefas e pode ser aproveitada em outras áreas de pesquisa. Os algoritmos de escalonamento sugeridos são genéricos, e, por conseguinte, aplicáveis a quaisquer ambientes heterogêneos nos quais as tarefas não tenham possibilidade de preempção, não necessitem comunicação e possam ter seu tempo de execução estimado através de alguma estratégia.

Além de tomar individualmente e estudar com maior profundidade as etapas aqui discutidas, alguns itens específicos podem ser indicados como trabalhos futuros. Mecanismos de replicação e formas de adição de novos métodos ao sistema em tempo de execução podem agregar maior valor ao sistema de busca de recursos. O mecanismo de previsão, se novas máquinas com diferentes configurações entrarem nos testes, poderá se tornar ainda mais preciso, alterando-se os coeficientes da equação polinomial. A rasterização é um processo que utiliza muita memória e a falta dela torna mais importante as preocupações com disco. A fórmula de ajuste que utiliza informações de memória RAM para aprimorar o resultado obtido visa indicar a presença de casos críticos, nos quais a falta de memória pode impedir a rasterização ou prejudicar o escalonamento. No entanto, se testes relacionando o consumo de memória com o custo dos documentos e informações de disco rígido (espaço livre e velocidade) forem adicionados, ajustes nos coeficientes podem oferecer um resultado mais apurado.

O mecanismo de previsão foi testado apenas para estimativas de tempo de execução para uma máquina de cada vez. Uma extensão para que ele possa estimar o tempo da execução de uma tarefa em um *cluster* não é baseada simplesmente na soma dos recursos disponíveis, mas na forma de trabalho, balanceamento de carga e rendimento do escalonador local. Assim, a criação e o aperfeiçoamento desta extensão rende novas pesquisas e descobertas. O mecanismo de previsão, através do custo computacional de uma tarefa, estima o tempo que ela levará para executar de acordo com os recursos computacionais disponíveis. Uma extensão para que o processo inverso possa ser utilizado também é uma funcionalidade útil para o futuro: informando o custo da tarefa e o tempo desejado para sua conclusão, a quantidade de recursos computacionais necessários será informada. Os recursos poderão ser fornecidos em uma estimativa de MIPS (Milhões de Instruções Por Segundo) e memória total necessária, ou até mesmo uma estimativa de quantidade de máquinas (interessante em ambientes homogêneos). Tal funcionalidade pode auxiliar o usuário no momento de fazer um planejamento de capacidade ou um dimensionamento dos *clusters*.

O algoritmo de escalonamento *Heterogeneous Deadlines Comitted Look Ahead* olha para o futuro no sentido de verificar quais tarefas possuem chances de terminar sua execução dentro do prazo estipulado de acordo com o estado atual das unidades de processamento e com estados futuros próximos que ocorrerão logo após o término das tarefas em execução. Esta previsão do comportamento futuro pode ir mais longe, prevendo o comportamento para um futuro mais longínquo, através da análise de todas tarefas da fila. Um algoritmo nesta linha se tornaria mais complexo, mas poderia encontrar soluções melhores para o escalonamento, ou, simplesmente, funcionar como uma heurística a fim de melhorar um algoritmo baseado em força bruta. Um algoritmo força bruta baseado em função de *fitness*, embora possa tornar-se demorado ou pouco eficiente, trata-se também de uma

outra abordagem que pode vir a ser comparada com os algoritmos sugeridos aqui.

Focando os documentos PDF, outras características físicas podem vir a fazer parte do sistema de escalonamento. Além de lidar com *deadlines*, os algoritmos podem ser aperfeiçoados para aproveitar melhor algumas características do documento que possam aumentar a velocidade de rasterização. Alguns computadores ou *clusters*, por exemplo, podem possuir *software* otimizado para transparência. Desta forma, seria mais vantajoso enviar tarefas com maior quantidade de transparência para estas unidades. Imagens e objetos de documentos rasterizados são guardados em *cache* pelo *ImageMagick*. Então, agrupar tarefas semelhantes na mesma localidade também pode trazer ganhos.

Um projeto mais ambicioso, focado em um futuro mais distante é, após os trabalhos focados em escalonamento local e global dentro de uma casa de impressão, extender o sistema para um ambiente cooperativo de grade computacional, no qual casas de impressão de mesmas ou diferentes companhias possam cooperar disponibilizando seus recursos umas para as outras, a partir de diferentes localidades. No caso de casas de diferentes companhias, tornar-se-ia interessante a implantação de um sistema de mercado virtual de recursos, como o sistema de leilão virtual *Bellagio*, descrito em [53]. Seria, então, atingido o nível mais alto de paralelização e cooperação de processamento, restando apenas a computação em nuvem [54].

Bibliografia

[1] Zerillo, S. D. Dot Matrix Printing Device Employing a Novel Image Transfer Technique to Print on Single or Multiple Ply Print Receiving Materials. United States Patent 4194846, 1980.

[2] Giannetti, F.; Fernandes, L. G.; Timmers, R.; Nunes, T.; Raeder, M.; Castro, M. High Performance XSL-FO Rendering for Variable Data Printing. In: ACM-SAC06: Proceedings of the 2006 ACM Symposium on Applied Computing. New York, NY (USA): ACM Press, 2006, pp. 811-817.

[3] Adobe System. PostScript Language Reference Manual. 2nd. ed. USA, 1990, 784p.

[4] Adobe Systems. PDF Reference. 5th. ed. USA, 2004, 1248p.

[5] Baker, M.; Buyya, R.; Hyde, D. Cluster Computing: A High-Performance Contender. Computer, vol. 32, no. 7, July, 1999, pp. 79-83.

[6] Fernandes, L. G.; Nunes, T.; Raeder, M.; Giannetti, F.; Cabeda, A.; Bedin, G. An Improved Parallel XSL-FO Rendering for Personalized Documents. In: Euro PVM/MPI07. Recent Advances in Parallel Virtual Machine and Message Passing Interface (LNCS). Heidelberg : Springer Berlin, 2007, pp. 56-63.

[7] Buckwalter, C. Integrating Systems in the Print Production Workflow: Aspects of Implemeting JDF. Digital Media Division, 2006, 35p.

[8] Oakley Norris Department; Oakley, A. L.; Norris, A. C. Page Description Languages: Development, Implementation and Standardization. In: Electronic Publishing Origination, Dissemination and Design, 1988, pp. 79-96.

[9] Purvis, L.; Harrington, S.; O'Sullivan, B.; Freuder, E. C. Creating Personalized Documents: An Optimization Approach. In: Proceedings of the 2003 ACM symposium on Document engineering (DocEng '03). ACM, New York, NY, USA, 2003, pp. 68-77.

[10] Davis, P.; Debronkart, D. PPML (Personalized Print Markup Language): a New XML-Based Industry Standard Print Language. In: XML Europe 2000. Paris, France: International Digital Enterprise Alliance, 2000.

[11] Nunes, T.; Raeder, M.; Kolberg, M.; Fernandes, L. G.; Cabeda, A.; Giannetti, F. High Performance Printing: Increasing Personalized Documents Rendering through PPML Jobs Profiling and Scheduling. In: Computational Science and Engineering, IEEE International Conference on, vol. 1, 2009 International Conference on Computational Science and Engineering, 2009, pp. 285-291.

[12] Nunes, T.; Giannetti, F.; Kolberg, M.; Nemetz, R.; Cabeda, A.; Fernandes, L. G. Job Profiling in High Performance Printing. In: 9th ACM Symposium on Document Engineering, 2009, Munique. Proceedings of the 9th ACM Symposium on Document Engineering (ACM DOCENG). New York, NY (USA): ACM Press, 2009, pp. 109-118.

[13] Nunes, T. Aplicando Estratégias de Escalonamento Através da Análise do Perfil de Jobs para Ambientes de Impressão Distribuídos. Dissertação (Mestrado). Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre, Brasil, 2009, 93p.

[14] Fernandes, L. G.; Nunes, T.; Kolberg, M.; Giannetti, F.; Nemetz, R.; Cabeda, A. Job Profiling and Queue Management in High Performance Printing. In: Computer Science - Research and Development. Heidelberg, Germany, September 2011, v. 26, pp. 1-20.

[15] Fonseca, C. M.; Raeder, M.; Kolberg, M. L.; Fernandes, Luiz Gustavo. Otimizando o Escalonamento de Jobs no Processo de Rasterizção de Documentos Personalizáveis. In: Escola Regional de Alto Desempenho, 2010, Passo Fundo. 10a Escola Regional de Alto Desempenho - Fórum de Pós-Graduação, 2010, pp. 125-126.

[16] CIP4. JDF Specification Release 1.3, 2005. http://www.cip4.org, acessado em Dezembro de 2009.

[17] Kuhn, W. Simulation of the Producttion Chain by use of an XML-based Job Definition Format. In: 14th European Simulation Symposium and Exhibition. University of Wuppertal, Wuppertal, Germany, 2002, v. 24, 5p.

[18] Buckwalter, C. A Messaging-Based Integration Architecture for Print Production Workflow Systems. In: Proceedings of Printing Technology SPb06, 2006, pp. 13-17.

[19] Fielding, R.; Gettys, J.; Mogul, J.; Frystyk, H.; Masinter, L.; Leach, P.; Berners-Lee, T. Hypertext Transfer Protocol – Http/1.1. RFC. RFC Editor, 1999.

[20] Oppenheimer, D.; Albrecht, J.; Patterson, D.; Vahdat, A. Distributed resource discovery on PlanetLab with SWORD. In: Proc. 1st WORLDS. San Francisco, CA, 2004.

[21] Awerbuch, B. and Scheideler, C. 2006. Towards a Scalable and Robust DHT. In Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures (Cambridge, Massachusetts, USA, July 30 - August 02, 2006). SPAA '06. ACM, New York, NY, 2006, pp. 318-327.

[22] Dunne, C. R. Using Mobile Agents for Network Resource Discovery in peer-to-peer Networks. SIGecom Exch, 2001, pp. 1-9.

[23] Aversa, R.; Di Martino, B.; Mazzocca, N.; Venticinque, S. A Resource Discovery Service for a Mobile Agents based Grid Infrastructure. In High Performance Scientific and Engineering Computing: Hardware/Software Support, L. T. Yang and Y. Pan, Eds. Kluwer Academic Publishers, Norwell, MA, 2004, pp. 189-200.

[24] Ramos, T. G.; Melo, A. C. M. A. An Extensible Resource Discovery Mechanism for Grid Computing Environments. In: Cluster Computing and the Grid, IEEE International Symposium on, Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), 2006, pp. 115-122.

[25] Alonso, G.; Casati, F.; Kuno, H.; Machiraju, V. Web Services: Concepts, Architecture and Applications. Springer Verlag, 2004, 374p.

[26] Berman, F.; Fox, G.; Hey, A. J. Grid Computing: Making the Global Infrastructure a Reality. John Wiley and Sons, Inc, 2003, 1080p.

[27] Spooner, D. P.; S.A. Jarvis, S. A.; Cao, J.; Saini, S.; Nudd, G. R. Local Grid Scheduling Techniques Using Performance Prediction. In: IEEE Proceedings Computers and Digital Techniques, 2003, pp. 87-96.

[28] Smith, W.; Foster, I. T.; Taylor, V. E. Predicting Application Run Times Using Historical Information. In: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, 1987, pp. 122-142.

[29] Dinda, P. A.; O'Hallaron, D. R. An Evaluation of Linear Models for Host Load Prediction. In: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing, 1999, 10p.

[30] Samadani, M.; Kaltofen, E. Prediction Based Task Scheduling in Distributed Computing. In: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing, 1995, pp. 261-265.

[31] van der Aalst, W. M. P.; Schonenberg, M. H.; Song, M. Time Prediction Based on Process Mining. In: Proceedings of Inf. Syst. Oxford, UK, April 2011, v. 36, pp. 450-475.

[32] Pino-Mejias, R.; Cubiles-de-la-Vega, M.D.; Silva-Ramirez, E.L.; Lopez-Coello, M. Non-Linear Modelling Time Series from ARIMA Fitting. In: Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on, 2005, pp. 1104-1109.

[33] Sugaya, Y.; Tatsumi, H,; Kobayashi, M.; Aso, H. Long-Term CPU Load Prediction System for Scheduling of Distributed Processes and its Implementation. In: Proceedings of

the 22nd International Conference on Advanced Information Networking and Applications, 2008, pp. 971-977.

[34] Davis, R. Dual Priority Scheduling: A Means of Providing Flexibility in Hard Real-time Systems. In: Report No. YCS230, University of York, UK, 1994, pp. 100-109.

[35] Chen, S. An introduction to heterogeneous optimal selection theory. In: ENCON '93. Proceedings. Computer, Communication, Control and Power Engineering.1993 IEEE Region 10 Conference on, 1993.

[36] Kabalan, K. Y.; Samari, W. W.; Hakimian, J. Y. Adaptative Load Sharing in Heterogeneous Systems: policies, modifications, and simulation. In: IEEE Transactions on Software Engineering, 1986, pp. 138-141.

[37] Lilja, D. J. Experiments with a Task Partitioning Model for Heterogeneous Computing. In: Proceedings of the 2 nd Workshop on Heterogeneous Processing, 1992, pp. 29-35.

[38] Iqbal, M. A. Partitioning Problems in Heterogeneous Computer Systems. In: Workshop on HeterogeneousProcessing, International Parallel Processing Symposium, 1993, pp. 23-28.

[39] Sridharan, S. V.; Zhou, Z. Dynamic Non-Preemptive Single Machine Scheduling. In: Computers and Operations Research. Oxford, UK, December 1996, v. 23, pp. 1183-1190.

[40] Romero, A. V. VirtualBox 3.1: Beginner's Guide. Packt Publishing, 2010, 348p.

[41] Muller, A. Virtualization with VMware ESX Server. Syngress, 2010, 608p.

[42] ImageMagick Home Page. Acessado em Julho de 2010. Disponível em: http://www.imagemagick.org.

[43] Cottet, F.; Delacroix, J.; Kaiser, C.; Mammeri, Z. Scheduling in Real-Time Systems. Wiley, 2002, 282p.

[44] Li, W.; Kavi, K.; Akl, R. A Non-Preemptive Scheduling Algorithm for soft Real-Time Systems. In: Computers and Electrical Engineering. New York, NY (USA), January 2007, v. 33, pp. 12-29.

[45] Georges, L.; Muehlethaler, P.; Rivierre, N. A few Results on Non-Preemptive real-time Scheduling. In: INRIA Research Report nRR3926, 2000, 34p.

[46] Singhoff, F.; Legrand, J.; Nana, L.; Marcé, L. Cheddar: a Flexible Real Time Scheduling Framework. In: ACM SIGAda Ada Letters Volume 24, 2004, pp. 1-8.

[47] Kurowski, K.; Nabrzyski, J.; Oleksiak, A.; Weglarz, J. Grid Scheduling Simulations with GSSIM. In: Proceedings of the 13th International Conference on Parallel and Distributed Systems. Washington, DC, USA, 2007, v. 2, pp. 1-8.

[48] Lowagie, B. iText in Action. In: Manning Publications, 2007, 656p.

[49] Li, H. Machine Learning for Performance Predictions on Space-Shared Computing Environments. In: International Transactions on Systems Science and Applications. New York, NY (USA), 2007, 13p.

[50] Vinicius de Andrade, F. Predição de Tempo de Execução de Tarefas em Grades Computacionais para Recursos não Dedicados. Dissertação (Mestrado). Universidade Federal de Minas Gerais. Belo Horizonte, Brasil, 2008, 107p.

[51] Lipari, G. EDF Scheduling. Scuola Superioure Sant'Anna, Pissa, 2008.

[52] Becker, L. B.; Gergeleit, M.; Schemmer, S.; Nett, E.;. Using a Flexible Real-Time Scheduling Strategy in a Distributed Embedded Application. Emerging Technologies and Factory Automation. Proceedings. ETFA '03. IEEE Conference, vol.2, 16-19 Sept, 2003, pp. 652- 657.

[53] Auyoung, A. Resource Allocation in Federated Distributed Computing Infraestructures. In: Proceedings of OASIS, 2004, 10p.

[54] Hayes, B. Cloud Computing. In: Communications of the ACM. New York, NY (USA), July 2008, v. 51, pp. 9-11.