Aplicando Estratégias de Escalonamento através da Análise do Perfil de *Jobs* para Ambientes de Impressão Distribuídos

Thiago Tasca Nunes

Porto Alegre
2009

Pontifícia Universidade Católica do Rio Grande do Sul
Faculdade de Informática
Programa de Pós-Graduação em Ciência da Computação

Aplicando Estratégias de Escalonamento através da Análise do Perfil de *Jobs* para Ambientes de Impressão Distribuídos

Thiago Tasca Nunes

Dissertação apresentada como requisito parcial à obtenção do grau de mestre em Ciência da Computação

Orientador: Prof. Dr. Luiz Gustavo Leão Fernandes

Porto Alegre
2009

Dados Internacionais de Catalogação na Publicação (CIP)

N972a Nunes, Thiago Tasca
Aplicando estratégias de escalonamento através da análise do perfil de Jobs para ambientes de impressão distribuídos / Thiago Tasca Nunes. – Porto Alegre, 2009.
94 f.

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

1. Informática. 2. Documentos – Personalização. 3. Rasterização. 4. Escalonamento. 5. Métricas. 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 "**Aplicando Estratégias de Escalonamento Através da Análise do Perfil de *Jobs* para Ambientes de Impressão Distribuídos**", apresentada por Thiago Tasca Nunes, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 16/01/09 pela Comissão Examinadora:

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

Prof. Dr. César Augusto Fonticielha De Rose – — PPGCC/PUCRS

Prof. Dr. Gerson Geraldo Homrich Cavalheiro – — UFPel

Homologada em 18/08/09, conforme Ata No. 14/09. pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
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

Dedico este trabalho aos meus pais Hugo César Quevedo Nunes e Sandra Maria Tasca Nunes, que me apoiaram nesta conquista, assim como em todos os demais aspectos de minha vida.

Agradecimentos

Gostaria de agradecer primeiramente aos meus pais que tornaram possível esta conquista. Aos companheiros de trabalho Mateus Raeder e Márcio Castro, que participaram desta luta comigo. Mateus, obrigado por ser o amigo que tu és, por me apoiar nos momentos difíceis e suportar minhas crises constantes. Márcio, obrigado por enfrentar os problemas junto à mim, não descansando até obter as melhores soluções possíveis. Ao meu "novo" colega de trabalho, Rafael Nemetz, que participou da pesquisa comigo, ajudando-me no trabalho durante estes dois últimos anos. Aos meus antigos colegas de trabalho Pedro Velho e Lucas Baldo, que foram mentores para que eu aceitasse o desafio de me tornar um mestre. Pedro, sempre me lembrarei dos dias engraçados que passamos, assim como da pessoa alegre e contagiante que és. Lucas, não me esquecerei da pessoa inovadora que se mostrasse, sempre propondo novos desafios e sugestões. Ao meu amigo Márcio Dorn, pelos inúmeros ensinamentos, aprendi muito com o teu brilhantismo e inteligência. Aos meus colegas de mestrado, que se tornaram amigos para uma vida inteira, entre eles: Andriele Busatto do Carmo, Elder Macedo Rodrigues, Everton Alexandre e Ricardo Piccoli. Finalmente, gostaria de agradecer ao meu amigo e orientador Luiz Gustavo Leão Fernandes pelos ensinamentos, companheirismo e amizade que apresentou durante todos esses anos em que estive no projeto de pesquisa. Com certeza não me esquecerei das diversas situações que enfrentamos juntos, assim como as inúmeras conversas não tão usuais nos momentos de descontração.

RESUMO

A impressão digital de documentos vem se tornando cada vez mais eficiente ao passar dos anos, o que provocou a criação de uma nova tendência: a personalização de documentos. Com a finalidade de suprir esta necessidade foram criadas linguagens para a descrição de conjuntos de documentos personalizados (*jobs*) e processos para permitir a impressão correta de tais *jobs*. Um dos processos que se destaca em termos de custo computacional é a rasterização de documentos, realizado sobre uma fila de *jobs*. No ambiente de impressão tradicional, algumas estratégias foram introduzidas para aumentar o desempenho desta fase através do emprego do uso de técnicas relacionadas ao processamento paralelo e distribuído. Entretanto, tais estratégias apresentam diversos problemas, dos quais o mais grave é relativo à impossibilidade da garantia de um balanceamento de carga justo para quaisquer seqüências de *jobs*. Assim, este trabalho vem a propor novas estratégias para aumentar o desempenho da fase de rasterização, através da análise do perfil dos jobs, para que então seja possível utilizar os recursos disponíveis de uma maneira mais eficiente. Para tanto, são propostas métricas que avaliam o custo computacional de cada job e ferramentas para permitir o escalonamento destes, de forma a superar o ganho de desempenho das estratégias existentes no âmbito da fila como um todo.

Palavras-chave: RIP, PDF, rasterização, métricas, escalonamento, balanceamento de cargas.

ABSTRACT

Digital Printing has become more efficient over the past few years, what led to the creation of a new tendency: the documents personalization. In order to fullfil this need, languages to describe a set of personalized documents (jobs) were designed, along with processes to allow the correct printing of such jobs. One of these processes, which demands a high computational effort, is the documents RIPping phase performed over a queue of jobs. In traditional printing environments, some strategies are applied to increase the performance of that phase through the usage of parallel and distributed computing techniques. However, such strategies present several issues, in which the most severe one is the impossibility to guarantee a fair load balancing for any job sequence. In this scenario, this work proposes new strategies to increase the performance of the RIPping phase, through the profiling of jobs in order to enable a more efficient usage of the available resources. For this purpose, metrics that evaluate the computational cost of each job and tools to permit a better scheduling of such jobs are proposed, overcoming the performance gain of the existing strategies over the whole job queue.

Keywords: RIP, PDF, RIPping, metrics, scheduling, load balancing.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

VDP	*Variable Data Printing*	15
PSP	*Print Service Provider*	15
PDF	*Portable Document Format*	16
PPML	*Personalized Print Markup Language*	16
VIP	*Variable Information Printing*	18
RIPping	*Raster Image Processing*	20
RIP	*Raster Image Processor*	20
VDT	*Variable Data Templates*	21
DVD	*Digital Versatile Disc*	23
PDL	*Page Description Language*	23
PCL	*Printer Control Language*	23
PS	*Postscript*	23
JPEG	*Joint Photographics Experts Group*	27
LZW	*Lempel-Ziv-Welch*	27
XObjects	*External Objects*	35
PNG	*Portable Network Graphics*	36
GIF	*Graphics Interchange Format*	36
LIFO	*Last In First Out*	38
CTM	*Current Transformation Matrix*	38
RGB	*Red Green Blue*	40
CMYK	*Cyan Magenta Yellow Key*	40
LS	*List Scheduling*	46
LPT	*Largest Processing Time first*	46
FFD	*First Fit Decreasing*	49
DPI	*Dots Per Inch*	52
E/S	Entrada e Saída	53
COW	*Cluster Of Workstations*	73
GHz	*Gigahertz*	73

GB	*Gigabytes*	73
RAM	*Random Access Memory*	73
MPI	*Message Passing Interface*	74
MPICH	*MPI CHameleon*	74
LAM/MPI	*Local Area Multicomputer / MPI*	74
HP	*Hewlett Packard*	88
MTAP	*Multimedia Tools and Applications*	88

Sumário

1 Introdução

O surgimento de impressoras digitais permitiu a evolução da área de publicação a novos patamares. Com estes dispositivos, a impressão de documentos de alta qualidade deixou de ser um empecilho para os usuários, podendo, então, ser realizada de forma eficiente. Este foi o primeiro passo para a consolidação de uma tendência emergente: a personalização de documentos. No passado, uma única instância de documento era produzida para um grande número de recipientes, ou seja, a mesma mensagem era passada para todos. Com a introdução da personalização, o desejado era adaptar os documentos de forma que diferentes mensagens fossem transmitidas para os correspondentes recipientes.

Neste sentido, procedimentos automatizados para a criação e transformação de documentos se tornaram necessários, com a finalidade de suprir a demanda existente. Uma nova disciplina - *Variable Data Printing* (VDP) [1] - foi introduzida, provendo diversas técnicas, tecnologias, conceitos e padrões para permitir a criação de documentos com conteúdo dinâmico. Diversas ferramentas foram desenvolvidas para auxiliar o *designer* a criar um *template*, do qual serão geradas diferentes instâncias de documentos. Desta forma, o mesmo *layout* é aplicado sobre diversas informações, gerando um *job* constituído por um conjunto de documentos personalizados (variáveis). Neste âmbito, linguagens capazes de apresentar o grau de flexibilidade necessário foram desenvolvidas, permitindo a definição de áreas estáticas e dinâmicas de um documento, assim como a formatação de seus respectivos conteúdos.

No cenário atual, a maioria das impressoras não é capaz de interpretar estas linguagens, portanto se tornam necessários diversos processos para possibilitar a impressão de forma correta dos documentos. Dois destes processos são a renderização e rasterização, que tem como objetivo final fornecer ao dispositivo de impressão o documento no formato necessário para que se consiga imprimí-lo de forma correta. Com a introdução do VDP, empresas especializadas na publicação digital (*Print Service Providers* - PSPs) necessitam realizar estes procedimentos sobre cada página dos documentos personalizados. Tal fato, acaba provocando um alto custo computacional para o processamento destes documentos.

1.1 Motivação

De uma forma geral uma PSP gerencia uma fila de n *jobs* iniciais a serem processados e **novos *jobs* podem ser inseridos na fila a qualquer momento**. Assim, o que deseja-se atingir é o

melhor desempenho possível no âmbito da fila como um todo e não de apenas um *job*. Para tal propósito, é comum a adoção de uma impressora com uma alta capacidade de processamento, em alguns casos sendo capaz de imprimir até 1 página por segundo [2]. Neste cenário, todas as atividades relacionadas à preparação dos *jobs* devem ser finalizadas em um espaço de tempo limitado, para que não ocorra a subutilização da impressora disponível. Além disso, PSPs geralmente utilizam impressoras em paralelo para aumentar o consumo dos documentos pertinentes a um dado *job*. Desta maneira, o desempenho das fases de pré-processamento deve aumentar proporcionalmente para mantê-las continuamente trabalhando.

Com o intuito de amenizar a alta demanda por desempenho trazida pelo VDP, passou-se a adaptar as linguagens de descrição de documento, como o *Portable Document Format* (PDF) [3], objetivando oferecer conteúdos re-utilizáveis. Até mesmo novos formatos *ad-hoc* foram introduzidos, dos quais o mais difundido é o PPML (*Personalized Print Markup Language*) [4].

A utilização destes formatos reduz a quantidade de conteúdo a ser rasterizada. Isto se deve ao fato de que as *Raster Image Processing* (RIP) *engines* (ou simplesmente RIPs), aplicações responsáveis por realizar a rasterização dos documentos, são capazes de capturar a re-usabilidade oferecida e, desta forma, processar os objetos uma única vez, aplicando o resultado obtido à medida do necessário. Entretanto, como mencionado anteriormente, a variabilidade dos documentos personalizáveis pode ser muito alta, portanto o recurso da re-usabilidade não consegue suprir por completo a demanda de desempenho existente, pois todas as porções variáveis devem ser rasterizadas uma a uma. Neste sentido, está claro que o tempo e vazão da fase de rasterização tem repercussões no processo de impressão como um todo, podendo prejudicar a competitividade das PSPs no seu mercado.

Em trabalhos passados [2,5] foram apresentadas estratégias para elevar a vazão do processo de renderização realizado pela *engine*, através da aplicação de técnicas de alto desempenho. Similar ao conceito de utilizar impressoras em paralelo para consumir o *job* de entrada rapidamente, se propôs o uso de *engines* de renderização em paralelo. No contexto do processo de rasterização, estratégias existentes obtêm um aumento de desempenho através do emprego de sistemas paralelos e distribuídos. Contudo, as soluções aplicadas no cenário de rasterização, na maioria das vezes, não proporcionam um desempenho satisfatório, devido a diversas limitações e problemas que podem ser explorados. Este fato motiva um estudo mais aprofundando de como melhor aproveitar os recursos existentes para maximizar o desempenho obtido em tal fase de processamento.

1.2 Objetivos

No cenário de rasterização de documentos nenhuma informação é conhecida sobre os *jobs* da fila e, portanto, estes são distribuídos sem uma maior preocupação quanto ao balanceamento de carga. Neste sentido, as estratégias existentes para a rasterização de documentos aplicam um

algoritmo simples para o escalonamento das tarefas entre os RIPs disponíveis, de maneira que à medida que um deles esteja livre, este receberá a primeira tarefa da fila. Esta distribuição simples não oferece nenhuma garantia de uma distribuição de cargas igualitária, podendo provocar a sobrecarga e sub-carga dos rasterizadores. Frente a isso, o tempo de rasterização de todos os *jobs* da fila torna-se insatisfatório e acaba por apresentar comportamentos imprevisíveis. Desta maneira, o ambiente de rasterização das PSPs, torna-se não confiável, causando a necessidade de estimar o término do processamento dos *jobs* da fila sempre como o pior caso possível.

O objetivo deste trabalho trata-se, então, da criação e aplicação de estratégias de escalonamento que garantam uma distribuição de cargas melhor. Com isso, o intuito é o de **melhorar o desempenho da fase de rasterização sobre a fila de *jobs* inteira**, provocando um uso mais inteligente e justo dos recursos disponíveis. Para atingir tal objetivo, diversos passos intermediários tornam-se necessários. Entre eles a criação de métricas capazes de estimar o custo computacional associado a cada *job*. Neste sentido, uma ferramenta capaz de prover as informações necessárias sobre cada *job* para a aplicação das métricas definidas deve ser desenvolvida. Com o uso desta ferramenta e das métricas definidas, será possível realizar a análise do perfil dos *jobs*, fornecendo assim detalhes suficientes para a aplicação de algoritmos de escalonamento com eficiências melhores do que o já utilizado. Na análise realizada neste trabalho, se está concentrando no formato PDF, por tratar-se de um formato difundido na descrição de documentos personalizados. Neste sentido, os resultados apresentados poderão ser aplicados sobre um conjunto de *job* totalmente descritos pelo formato PDF, ou sobre formatos de abstração de níveis mais altos, que utilizam tal formato para descrever o conteúdo de seus documentos, como o PPML.

1.3 Estrutura do trabalho

Este volume organiza-se como segue: o Capítulo 2 descreve alguns aspectos ligados à construção de documentos personalizáveis; o Capítulo 3 discorre sobre algumas características principais do formato PDF, explicando a composição de um documento descrito por tal formato; o Capítulo 4 apresenta problemas, classificações, algoritmos e formalismos relacionados ao assunto de escalonamento, com o propósito de justificar e esclarecer as escolhas realizadas para o desenvolvimento das estratégias propostas; o Capítulo 5 demonstra as abordagens propostas, explicitando as métricas propostas e as novas estratégias de escalonamento utilizadas, assim como as ferramentas auxiliares empregadas para a construção destas; o Capítulo 6 trata sobre o ambiente de teste utilizado para efetuar a validação das métricas propostas sobre *jobs* reais retirados do cenário de mercado, em conjunto com uma análise de desempenho sobre a ferramenta auxiliar utilizada para a análise do perfil dos *jobs* e das novas estratégias de escalonamento, comparando-as com as já existentes; por fim, o Capítulo 7 conclui o estudo e apresenta possíveis pesquisas futuras a serem realizadas a partir dos resultados obtidos neste trabalho.

2 O Processo de Impressão

O processo de impressão reúne diversas técnicas, tecnologias, padrões e fases de processamento que permitem a obtenção de mais flexibilidade, eficiência e eficácia na criação de documentos. Desta maneira, possibilita-se às PSPs a atenderem mais facilmente os seus clientes e a conseguirem lidar com suas necessidades. Este capítulo expõe alguns dos aspectos envolvidos no processo de impressão, explicitando os passos e fases existentes desde a criação de um documento personalizável até a sua impressão. Além disso, são apresentadas as estratégias existentes no âmbito de rasterização de documentos, visando explicitar suas principais vantagens e desvantagens.

2.1 Variable Data Printing

A impressão de dados variáveis ou *Variable Data Printing* (VDP) [1] se trata de uma tendência para possibilitar a criação de documentos com conteúdo dinâmico. Também conhecido como *Variable Information Printing* (VIP), *Personalized Printing*, *Database Publishing*, entre outros, VDP traz diversas facilidades para a construção de documentos flexíveis. Com o uso de VDP, o *designer* pode definir áreas personalizáveis de um documento que correspondem às porções variáveis deste. Contrastando com a simples impressão digital, ao invés de se criar um documento contendo uma única mensagem para diversos clientes, com VDP é possível utilizar-se um documento para passar tantas mensagens quanto necessárias, personalizadas para cada cliente. Portanto, a grande vantagem do uso de VDP é o que se pode chamar de personalização em massa. A Figura 1 ilustra a dinamicidade que pode ser encontrada em um documento com conteúdo personalizável, tal como variações de formatação, texto, imagens ou até mesmo de sua própria estrutura interna.

VDP foi adotado por diversas empresas, por facilitar o processo de criação dos documentos. Tratando-se de situações reais de mercado pode-se citar o caso das Olimpíadas de Sydney [6], em que os resultados da competição foram publicados através do uso de VDP. Diversas outras podem ser observadas no cotidiano, como, por exemplo, a fatura de um banco. Esta possui sempre o mesmo *layout*, mas difere no conteúdo, mais especificamente nos dados pessoais e financeiros de cada cliente. Desta maneira, o banco pode definir um único documento com áreas personalizáveis, onde estas correspondem as informações que diferenciam-se de um cliente para o outro. Outro exemplo muito comum que se pode citar é o caso dos encartes de supermercados.

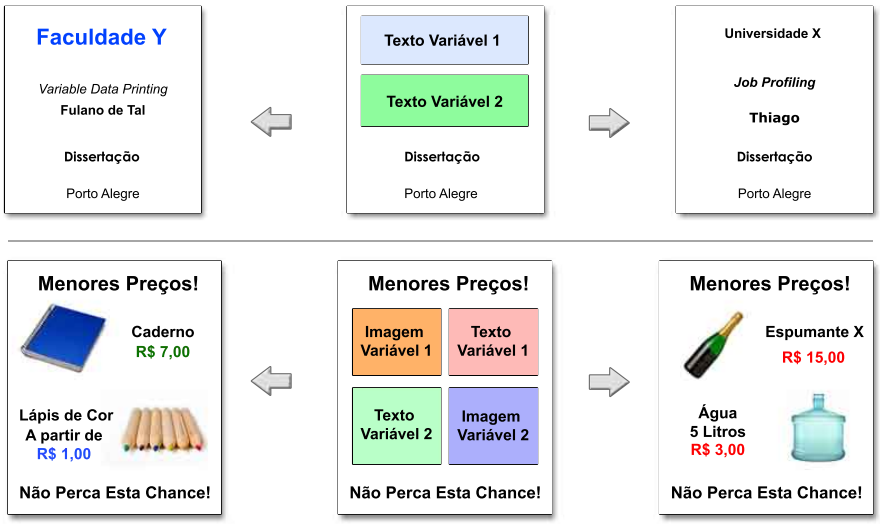

Figura 1 – Flexibilidade oferecida pelo VDP.

Este tipo de documento possui sempre a mesma estrutura: figuras de produtos espalhados na página com seus respectivos preços. Assim, o *designer* pode fixar as posições das imagens e de seus preços, mas considerar que tanto um quanto outro são porções variáveis em um documento. Com isso, o *template* criado pode ser reutilizado para a construção de diversos encartes de produtos diferentes.

Contudo, para que VDP seja realmente aplicável à construção de documentos mais flexíveis são necessárias linguagens para descrever as partes variáveis e estáticas destes. Entretanto, a maioria das impressoras não são capazes de lidar diretamente com estas linguagens, gerando a necessidade da existência de fases de pré-processamento.

2.2 Renderização e Rasterização

Dois processos pertinentes ao contexto do pré-processamento na impressão de dados variáveis são a renderização e a rasterização de documentos. Apesar de normalmente serem conhecidos como sinônimos, o propósito de cada um é distinto, mas complementar, no sentido de que a saída gerada por um é utilizada como entrada por outro.

A renderização refere-se ao processo de interpretação de uma determinada linguagem de formatação não usual (ou óbvia) a um leitor humano, objetivando apresentar o conteúdo por ela encapsulado de forma inteligível e usual. Neste escopo, são utilizadas técnicas de *Pretty-printing* [7], que preocupam-se em ilustrar um determinado conteúdo da melhor forma possível para os usuários finais (Figura 2). Através da aplicação da renderização a um documento personalizável cru, ou seja, descrito por linguagens de formatação com uma apresentação não óbvia a um leitor humano, é produzido um documento com parágrafos, fontes, cores, figuras, margens, páginas, numeração, etc. já definidos, de acordo com o especificado pela linguagem de forma-

tação utilizada. A aplicação responsável por realizar este processo é a *engine* de renderização.

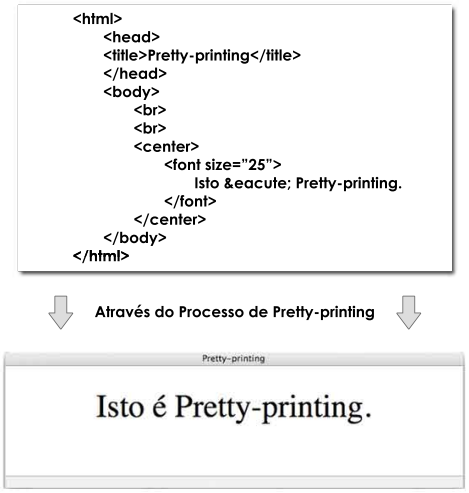

Figura 2 – *Pretty-printing*.

O resultado do processo de renderização é um documento descrito por uma linguagem de alto nível de abstração, pois acaba provendo portabilidade ao documento, tornando possível sua visualização ou impressão em diferentes dispositivos. Neste documento resultante, boa parte do conteúdo é representado por gráficos vetoriais que independem de resolução, significando que caso se deseje visualizar mais ou menos detalhes de tal conteúdo, não haverá perda de qualidade. Entretanto, estes gráficos vetoriais não fornecem a informação de como o dispositivo deve apresentar o conteúdo, mas dispõem de uma abordagem genérica para representá-lo.

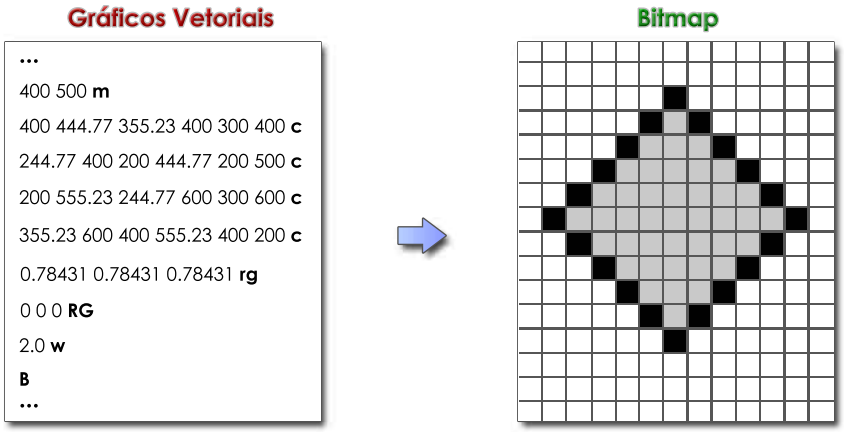

Figura 3 – Rasterização.

O processo de rasterização, também conhecido como *RIPping* (*Raster Image Processing*), se trata da interpretação de um documento descrito através de gráficos vetoriais para um formato *bitmap*, ou seja, para uma matriz de pontos (com suas correspondentes cores), que indicam corretamente ao dispositivo de impressão em questão, como apresentar o conteúdo descrito no documento (Figura 3). Neste contexto, as chamadas RIP *engines* (*Raster Image Processor engines*), ou simplesmente RIPs, são os responsáveis por realizarem a rasterização dos documentos.

2.3 Ambiente de Impressão Tradicional

Esta seção apresenta o ambiente de impressão empregado em uma PSP tradicional que utiliza a técnica de VDP. Neste sentido, são explicitados dois componentes principais: o fluxo de trabalho, através da técnica de VDT (*Variable Data Templates*) [8] e arquitetura de *hardware*, especificando como são dispostos os componentes para realizar o processamento necessário para a impressão dos *jobs* da fila.

2.3.1 Variable Data Templates

Uma técnica amplamente utilizada no cenário de VDP é a VDT. Esta propõe um fluxo de trabalho (Figura 4), com o objetivo de padronizar o processo de criação e impressão dos documentos, tornando-o confiável, previsível e aumentando a vazão de documentos impressos. Deve-se ressaltar que o fluxo de trabalho apresentado é um modelo genérico, podendo ser extendido e modificado para suprir necessidades específicas de diferentes cenários.

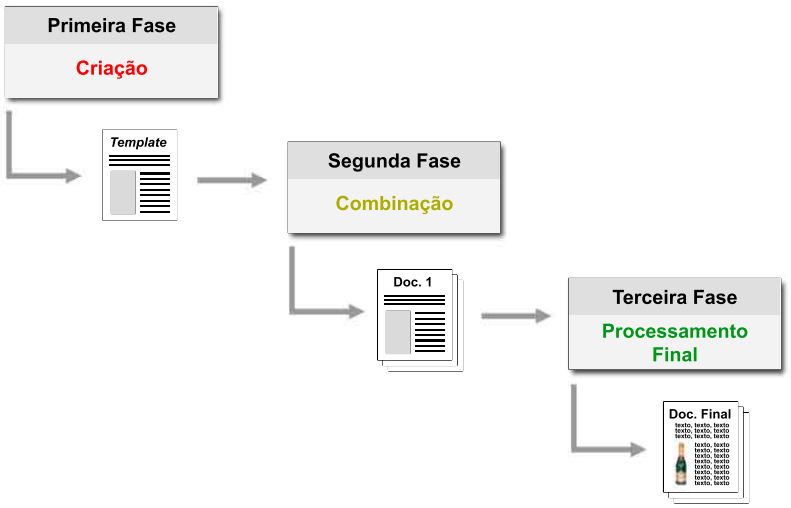

Figura 4 – Fluxo de trabalho VDT.

A implementação deste fluxo não estabelece o uso de uma tecnologia específica, podendo se adaptar a qualquer uma que realize a função necessária. Existem três fases básicas: criação, combinação e processamento final. A fase de criação gera como saída um *template* de documento que descreve o *layout* a ser empregado, além de elementos fixos e estáticos nele contidos. Na segunda fase, deste *template* são instanciados diversos documentos contendo dados específicos provenientes de uma base de dados. Cada instância é repassada para um processo de formatação, renderização e rasterização (que em conjunto compõem a terceira fase), gerando documentos prontos para serem impressos. É importante ressaltar que os três processos pertencentes à última fase são complementares: primeiramente o documento é formatado por uma

determinada linguagem, logo após é aplicado o processo de renderização e, por fim, para que este seja impresso, faz-se o processo de rasterização.

Diversas vantagens são obtidas através da aplicação deste fluxo de trabalho segmentado: *(i)* as PSPs são capazes de lidar mais facilmente com impressões de uma grande quantidade de dados, devido à existência de fases bem definidas e corretamente especificadas; *(ii)* separou-se o *design* do conteúdo; *(iii)* permite-se a composição do *layout* independentemente da disponibilidade dos dados em um primeiro momento; *(iv)* liberdade para aplicar a tecnologia que melhor se adapta à necessidade de cada empresa, além do fluxo ser facilmente extensível *(v)*.

2.3.2 Arquitetura Empregada

Levando-se em consideração o uso da técnica de VDT, pode-se apresentar uma visão geral da arquitetura de *hardware* empregada neste cenário. A Figura 5 ilustra os diferentes passos sobre os quais os documentos são submetidos para que finalmente possam ser impressos, demonstrando uma visão geral do *hardware* que pode ser utilizado.

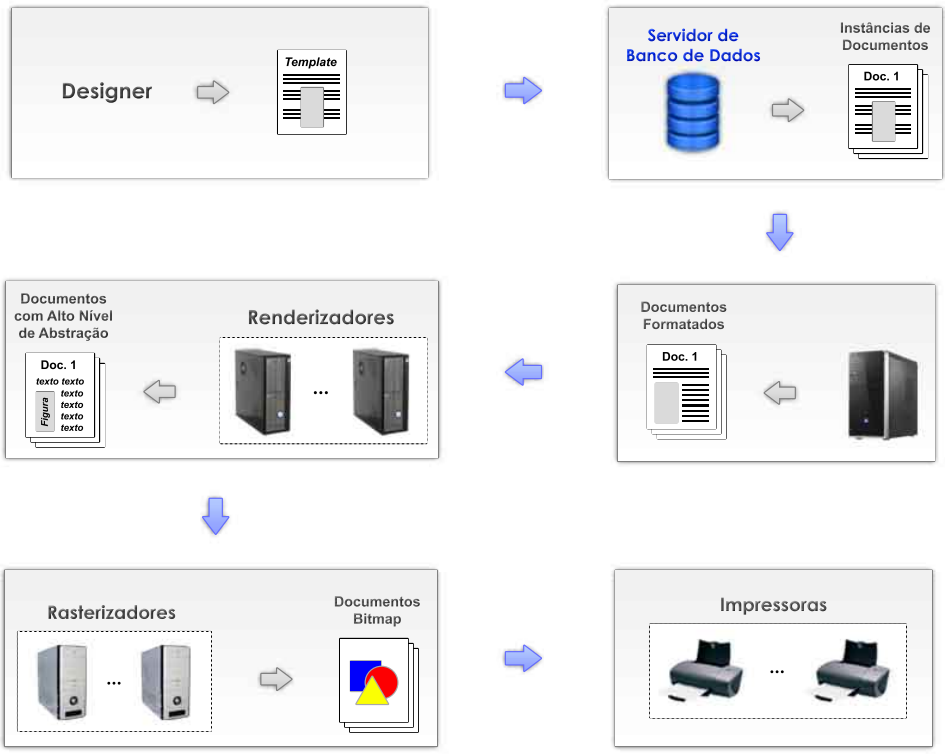

Figura 5 – Arquitetura empregada para VDT.

Como mencionado anteriormente, o *designer* será o responsável por estebelecer um *template* de documento contendo as diretivas de *layout* e apresentação. Este *template*, então será combinado com registros do banco de dados gerando as diferentes instâncias dos documentos personalizados. Neste sentido, as PSPs dispõem de um servidor de banco de dados contendo as

informações personalizadas que serão acopladas a cada documento. Com isso, os documentos são repassados a uma fase de formatação. Esta formatação normalmente é realizada de forma automatizada em uma estação, onde para cada instância existente, diversos estilos de texto são aplicados.

Partindo dos documentos formatados em mãos, passa-se à fase de renderização, onde geralmente realiza-se a renderização de forma centralizada. Entretanto, pode-se aplicar neste passo estratégias de renderização paralelas, como apresentado em trabalhos passados. O resultado da renderização, então, será o conteúdo dos documentos descritos através de uma linguagem de alto nível de abstração. Considerando-se que é comum a existência de *jobs* com milhares de instâncias de documentos, aqueles resultantes da renderização serão arquivos muito grandes, nos quais um único *job* poderá ocupar *Gigabytes* de disco. Sendo assim, estes serão transmitidos via redes de alta vazão ou através de mídias portáteis, como o DVD (*Digital Versatile Disc*) para uma estação contendo diversos RIPs. Neste contexto, os *jobs* são armazenados em um diretório de uma aplicação escalonadora, que é visível por todas unidades de processamento. O escalonador dirá, então, qual *job* ou porção deste deverá ser processado por cada RIP. Cabe ressaltar que antigamente os RIPs estavam presentes dentro das impressoras, mas estes foram trazidos para fora devido a motivos de escalabilidade. Por fim, o resultado da rasterização será repassado para as impressoras disponíveis pelos próprios RIPs.

2.4 Page Description Language

PDL (*Page Description Language*), também conhecido como PCL (*Printer Control Language*), se trata de uma linguagem de alto nível de abstração, utilizada para descrever o conteúdo de uma ou mais páginas. Através do emprego deste tipo de linguagem, o conteúdo de um documento pode ser descrito através de comandos específicos, ao invés de um mero conjunto de pontos. Desta maneira, obtém-se um nível de abstração mais alto, não prendendo o documento a formatos dispositivos específicos.

Entre os formatos PDL mais conhecidos pode-se citar o PS (*Postscript*) [9], que é considerado por muitos como uma linguagem de programação completa, o PDF [3], que nada mais é do que uma linguagem baseada no PS, e o *Hewlett Packard Printer Control Language* [10] (também conhecido simplesmente como PCL). Cabe ressaltar que devido ao fato do PDL prover uma linguagem de um alto nível abstração, os dispositivos de impressão e de apresentação não são capazes de reconhecer estas linguagens. Assim, uma conversão deve ser realizada para apresentar o documento no dispositivo desejado. Este processamento, então, será realizado pelas *engines* de rasterização.

2.5 Estratégias de Rasterização Existentes

As estratégias existentes em um ambiente de rasterização tradicional baseiam-se em sistemas paralelos e distribuídos para aumentar a vazão e desempenho de tal fase. Neste sentido, diversos RIPs são aplicados em conjunto para rasterizar uma dada fila de *jobs* de forma paralela. Desta forma, através da análise das estratégias é possível verificar as vantagens e desvantagens existentes. Neste contexto, são aplicadas duas maneiras para rasterizar os *jobs* da fila de entrada: através da quebra de um único *job* em partes que serão processadas em paralelo, ou através da distribuição de *jobs* inteiros para RIPs separados. Assim, são identificadas três estratégias distintas que são comumente aplicadas para acelerar a rasterização dos *jobs*:

1. alocar um RIP por *job*;
2. alocar todos os RIPs para um único *job*;
3. alocar um número fixo de RIPs por *job*.

Cabe ressaltar que no cenário das estratégias existentes os *jobs* são disponibilizados em um diretório visível por todos os RIPs, que realizam a leitura necessária para a execução do processo de rasterização. Por outro lado, o resultado do processamento de cada RIP é escrito no disco local, para então ser transmitido para sua impressora associada.

Através do emprego da primeira estratégia, cada RIP existente irá processar um *job* inteiro (Figura 6). Neste contexto, uma aplicação escalonadora irá repassar um *job* para cada RIP livre e à medida que cada RIP terminar o seu trabalho, este irá requisitar mais tarefas para o escalonador. Este cenário funciona bem para *jobs* pequenos, com alta re-usabilidade, permitindo assim que cada RIP consiga tirar vantagem do recurso da re-usabilidade e possa finalizar a computação de sua tarefa em tempo hábil para continuamente alimentar as impressoras. Entretanto, se houver *jobs* grandes onde o tempo de processamento seja muito alto, é provável que não seja possível realizar o processamento com o desempenho desejado, prejudicando o processo de impressão como um todo. Além disso, caso o tamanho dos *jobs* variar muito (o que é comum), ocorrerá a sub-carga e sobrecarga das unidades das *engines* de rasterização, o que por sua vez também afetará o ganho de desempenho obtido. Por outro lado, outra desvantagem presente nesta estratégia específica, é o fato de que diversos RIPs podem não ser utilizados à medida que a quantidade de *jobs* na fila for menor que o número de RIPs disponíveis.

A segunda estratégia visa a utilização de todos os RIPs disponíveis para um único *job* por vez (Figura 7). Esta é a abordagem "força-bruta", onde cada RIP irá processar uma porção de um dado *job*. Cada porção será composta por uma ou mais páginas dos documentos do *job* em questão, sendo que nunca existirão porções com páginas fracionadas, ou seja, a unidade atômica de cada porção será uma página. Desta maneira, cada *job* da fila será sempre dividido em p porções, onde p corresponde aos n RIPs disponíveis ou a pg páginas, caso o número de

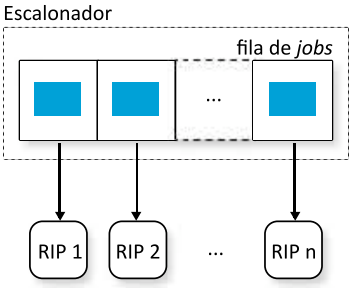

Figura 6 – Alocação de um RIP por *job*.

páginas do dado *job* seja menor do que a quantidade de rasterizadores. O responsável por definir estas porções será o escalonador, repassando os limites encontrados para cada RIP disponível. Estes RIPs, então, quebrarão os *jobs* gerando suas porções e processando-as. Portanto, a quebra propriamente dita dos *jobs* em fragmentos não será feita pelo escalonador, que apenas definirá quais são as páginas de cada fragmento, mas sim por cada RIP. Uma desvantagem encontrada nesta estratégia é a quebra do recurso da re-usabilidade entre porções, pois a divisão dos *jobs* em arquivos distintos acaba impossibilitando o uso deste recurso entre eles. Além disso, há a possibilidade de ocorrer um desbalanceamento de carga entre as unidades de processamento prejudicando o ganho de desempenho que poderia ser obtido, pois não existe garantia de que o esforço gasto para rasterizar diferentes porções seja o mesmo.

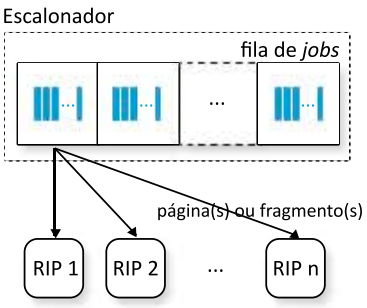

Figura 7 – Alocação de todos os RIPs para cada *job*.

Finalmente, a terceira estratégia estipula o uso de um número fixo r de RIPs para cada *job* da fila. A Figura 8 exemplifica esta estratégia considerando a situação em que cada grupo comporta três RIPs ($r = 3$). Esta configuração é baseada no fato de que os *jobs* "padrões" das PSPs necessitam do número especificado de recursos para manterem as impressoras continuamente trabalhando. Assim, diversos grupos de r RIPs estarão disponíveis para processar os *jobs* e cada um será direcionado para um grupo distinto. Nesta configuração, cada *job* será quebrado em r fragmentos e novamente, a quebra propriamente dita será realizada pelos RIPs, não pelo escalonador, que apenas definirá as páginas de cada porção. Como qualquer configuração estática, esta só irá funcionar da maneira esperada caso as PSPs imprimam *jobs* que se encaixem no perfil de *jobs* utilizado. Caso contrário, o número de RIPs alocados para os *jobs* pode não ser o suficiente para manter as impressoras continuamente alimentadas. Além disso, uma grande

desvantagem apresentada nesta estratégia é o fato de que um conjunto de RIPs somente será alocado para um novo *job*, assim que todos os RIPs estejam livres, ou seja, no momento em que o *job* atual for processado por completo pelo grupo. Portanto, diversos RIPs de um grupo podem ficar parados, enquanto outros do mesmo grupo não finalizaram a computação de sua porção, sub-utilizando os recursos disponíveis.

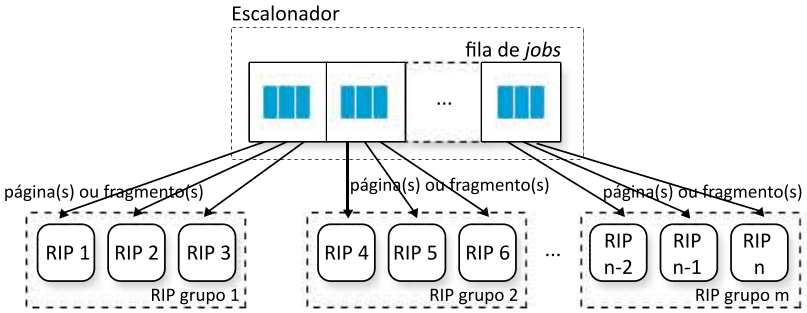

Figura 8 – Alocação de um número fixo de RIPs para cada *job*.

3 Formato PDF

O PDF (*Portable Document Format*) [3] é um formato altamente difundido dentre as PSPs para a descrição de documentos, com uma riqueza maior em termos de recursos visuais, do que formatos antigos como o PS. Além disso, tal formato está em uma constante evolução para atender a novas necessidades dos usuários, o que torna este formato atual e aplicável a situações que os demais formatos não conseguiriam enfrentar. Este capítulo vem a apresentar as principais características do formato PDF, objetivando fornecer uma visão geral sobre os elementos que o compõem.

PDF é uma PDL utilizada para descrever gráficos de um documento desenvolvida pela empresa Adobe® . Este formato provê uma representação que independe de *software*, *hardware* ou sistema operacional utilizado para criá-lo, ou do dispositivo que será empregado para apresentar o conteúdo por ele descrito. Abaixo são explicitadas algumas características e vantagens fornecidas pelo formato PDF:

- **Portabilidade**: o PDF foi desenvolvido para ser portável para quaisquer plataformas. Um documento PDF é armazenado como um arquivo binário, ao invés de ser representado como texto puro, evitando a tradução de caracteres nativos a determinados sistemas operacionais, como caracteres de fim de linha, acentos, etc.

- **Compressão**: documentos PDF suportam padrões difundidos de compressão, como JPEG (*Joint Photographics Experts Group*) *compression* [11], LZW (*Lempel-Ziv-Welch*) [12, 13], entre outros, permitindo a redução do tamanho em disco do PDF.

- **Gerenciamento de Fontes**: com o uso de PDF é possível adicionar ao documento diversas fontes de distintos formatos (*Type 1*, *TrueType*, etc.) e acoplar aquelas que não possuem restrições de *copyright* diretamente no documento, possibilitando a apresentação do documento, tal como foi criado, em quaisquer outras plataformas. Além disso, em um documento PDF estão contidos descritores de fontes para cada fonte empregada no documento. Estes descritores provêem métricas e estilos das fontes empregadas no documento, possibilitando a seleção de um conjunto de fontes que se assemelham as primeiras, caso estas fontes originais não existam no sistema operacional utilizado.

- **Acesso Randômico**: devido à maneira de representação de dados em um documento PDF, a ordem da descrição das páginas ou objetos no arquivo não é importante, pois se utiliza um sistema de referências. Isto pode ser vantajoso para qualquer documento interativo ou para uma aplicação que necessite ler o documento em uma ordem arbitrária. Neste

contexto, uma tabela (chamada de *cross-reference table*), que contém a localização de páginas e outros objetos importantes no PDF, está sempre presente no final do arquivo, possibilitando o acesso direto a tais elementos.

- **Segurança**: um documento PDF pode ser criptografado (ou assinado digitalmente) e decriptografado por diversos meios, seja com uma chave pública e privada ou através de um meio biométrico, como uma impressão digital.

- **Extensibilidade**: o formato PDF foi desenvolvido com o intuito de ser inteiramente extensível, possibilitando a criação de novas características que venham a se tornar interessantes ou necessárias para os usuários. Além disso, o PDF proporciona para aplicações baseadas em versões mais antigas do PDF, a capacidade de lidar elegantemente com recursos que elas não compreendem.

Cabe ressaltar que algumas destas vantagens já existiam em um formato, o qual o PDF é baseado: o PS (*PostScript*). Neste contexto, o PDF herdou alguns outros atributos de tal formato, como o fato de que um documento será descrito através de uma série de objetos. Estes objetos em conjunto irão ser os responsáveis pela descrição da aparência de uma ou mais páginas. Desta forma, para uma melhor compreensão do formato PDF, a seguir são apresentados aspectos relacionados à estrutura de um documento PDF e os objetos que o compõem, assim como suas características principais.

3.1 Composição de um Documento PDF

Nesta seção serão abordados aspectos relacionados a um documento PDF, mais especificamente quanto a sua estrutura interna. Neste sentido, apresentam-se dois níveis distintos de especificação: quanto a estrutura do arquivo e quanto a estrutura do documento. O primeiro diz respeito a um aspecto mais físico, descrevendo como os objetos são armazenados no arquivo PDF, assim como estes são acessados. Já o segundo trata sobre um nível de abstração mais alto, relatando como os objetos são utilizados para representar os elementos de um documento PDF, como por exemplo, páginas, fontes, etc.

3.1.1 Estrutura do Arquivo

Um arquivo PDF comum é composto por quatro elementos básicos, que possuem funções distintas para a representação de conteúdo. Estas partes constituíntes estão dispostas fisicamente como apresentado na Figura 9.

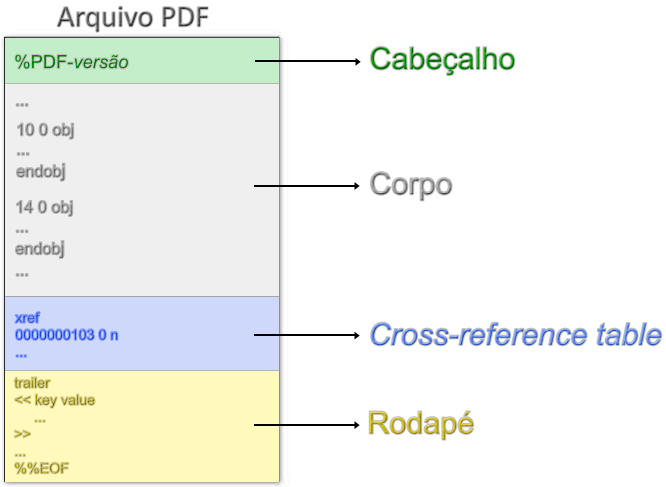

Figura 9 – Estrutura de um arquivo PDF.

No elemento cabeçalho é descrita apenas uma linha, que indica a versão da especificação do PDF, a qual o arquivo utiliza. Esta linha emprega a seguinte sintaxe: o símbolo %, a palavra PDF seguida de um hífen e a versão utilizada. Neste contexto, um arquivo que foi descrito através da especificação 1.7 do PDF, por exemplo, terá o seguinte cabeçalho:

%PDF-1.7

Cabe ressaltar que a partir da quinta versão do PDF (versão 1.4), o cabeçalho pode ser substituído pela entrada **Version** em uma estrutura chamada de *document catalog*, que trata-se de um elemento chave em um documento PDF (mais detalhes sobre este elemento serão apresentados na Seção 3.1.2). Desta maneira, a especificação torna-se atualizável, adicionando mais flexibilidade ao documento. Evidentemente, uma aplicação capaz de interpretar uma dada versão do PDF, também conseguirá compreender as respectivas versões mais antigas.

O elemento seguinte ao cabeçalho trata-se do corpo. Esta é a porção principal do documento, onde será descrito o conteúdo propriamente dito do arquivo PDF. Em tal elemento estarão os objetos representando os componentes do documento, como fontes, páginas, imagens, textos, etc. Assim sendo, o corpo irá englobar uma série de objetos, que podem estar descritos em uma ordem arbitrária qualquer.

Como mencionado anteriormente, a *cross-reference table* é uma tabela que contém a localização dos objetos dentro do arquivo. Esta tabela possui uma entrada para cada objeto indireto no documento. Desta maneira, cada linha sua apontará para a localização de um objeto em questão no corpo do arquivo, evitando a leitura do documento inteiro para a obtenção deste.

Por fim, o elemento rodapé possui a localização da *cross-reference table* e de alguns objetos especiais. Frente a isto, a aplicação que está interpretando o documento deve ler o documento a partir do final, podendo obter informações sobre componentes de interesse rapidamente. Evita-se, então, a realização de buscas custosas para encontrar os elementos desejados. Neste contexto, é importante mencionar que um dos objetos que podem ser encontrados através do rodapé, trata-se da estrutura *document catalog*.

3.1.2 Estrutura do Documento

Como mencionado anteriormente, um documento PDF trata-se de uma série de objetos que compõem a aparência de sua(s) página(s). Neste contexto, tais objetos estão organizados de forma hierárquica, definindo desta maneira, como deverá ser interpretado o documento. Para uma melhor visualização de tal hierarquia, a Figura 10 ilustra os objetos com suas respectivas posições em tal estrutura, assim como as relações existentes entre estes elementos.

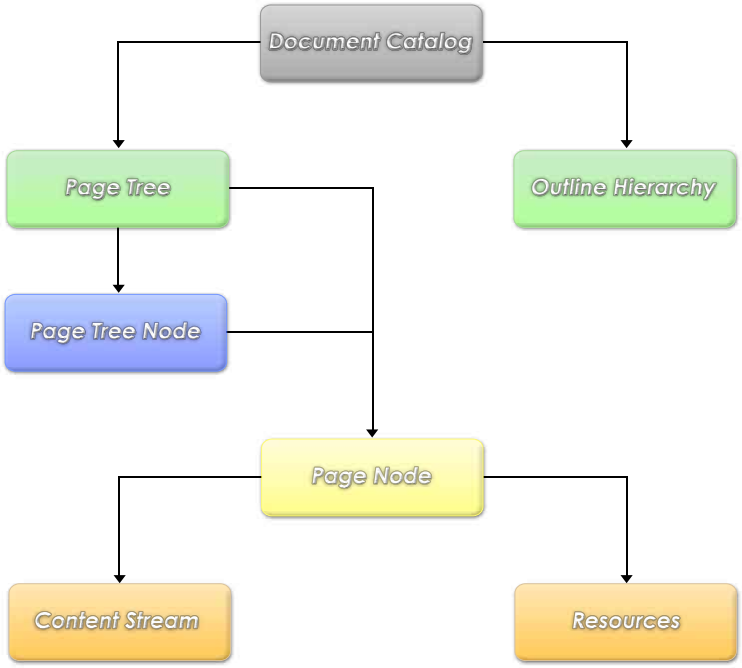

Figura 10 – Estrutura de um documento PDF.

No topo da hierarquia está o *document catalog*. Este elemento trata-se de um dicionário de dados, que contém referências para as páginas do documento, através da estrutura *page tree*. Assim sendo, é a partir deste elemento que poderá ser obtido, posteriormente, o conteúdo de cada página do documento. Além disso, neste catálogo estão presentes diretivas de formatação do documento, como: o *layout* das páginas (uma ou duas colunas), a configuração de contorno de cada uma destas (via o elemento *outline hierarchy*), se devem ser mostradas fotos miniaturas de cada uma das páginas, se o documento deve ser visualizado em tela cheia, etc. Como mencionado anteriormente, a partir da versão 1.4 do PDF, no *document catalog* também pode estar presente a versão da especificação PDF empregada, inutilizando a informação fornecida no cabeçalho do documento.

As páginas do documento PDF são acessadas através de uma estrutura conhecida como *page tree*, que define a ordem das páginas e contém referências aos objetos que possuem o conteúdo de cada página. Através desta estrutura, aplicações são capazes de apresentar as páginas do documento rapidamente, utilizando uma quantidade limitada de memória. Neste contexto, a

estrutura *page tree* possui dois tipos de nodos: nodos intermediários (*page tree nodes*) e nodos folhas (*page nodes*).

Os *page tree nodes* são utilizados para especificar atributos em comum entre as páginas. Estes atributos são relacionados a apresentação de cada página, como por exemplo, o seu tamanho, se a página deve ser rotacionada, etc. Neste sentido, as características especificadas para cada *page tree node* são atribuídas a todos os seus filhos. É importante ressaltar, que os *page tree nodes* não estão necessariamente relacionados a estrutura lógica do documento, ou seja, estes elementos podem não representar capítulos, seções e assim por diante.

Os nodos folha, *page nodes*, são as estruturas que representam as páginas propriamente ditas. Estes elementos são dicionários, que especificam os atributos de uma página. Entre os atributos existentes, devem ser ressaltados os seguintes: uma referência para o *page tree node* pai, o tamanho da página (suas dimensões de altura e largura), uma referência para o conteúdo da mesma e outra para os recursos utilizados na página.

Por fim, o conteúdo de uma página trata-se de uma unidade auto-contida, sendo definida através de um elemento chamado de *content stream*, que utilizará se necessário objetos externos ao seu escopo, com o emprego de uma estrutura conhecida como *resources*. Por unidade auto-contida se quer dizer que, todos os objetos necessários para a definição do conteúdo da página em questão deverão estar propriamente definidos no objeto *content-stream*, ou devidamente referenciados via o correspondente elemento *resources*. O *content stream*, então, trata-se de uma sequência de instruções descrevendo os elementos a serem pintados na página. Estas instruções são descritas através de um operador e de um operando. O primeiro trata-se de uma palavra reservada do PDF especificando uma ação a ser tomada. O segundo nada mais é do que um objeto PDF ou uma referência interna ao mesmo. Portanto, o operando deve ser um objeto direto, significando que não podem ser utilizadas referências a objetos externos ao par *content stream* e *resources*. Por outro lado, pode ser necessário a utilização de tais objetos e para este fim é empregada a estrutura *resources*. Esta estrutura conterá o mapeamento *nome / objeto externo*, que poderá então ser referenciado através do atributo nome dentro do *content stream*. Deve ser ressaltado que enquanto o documento PDF trata-se de um sistema baseado em referências, **no contexto de um *content stream*, as instruções serão lidas e executadas seqüencialmente**.

3.2 Objetos PDF

O formato PDF utiliza uma sintaxe própria (disponível em [3]), que é embasada em um sistema de referências através do uso de diversos objetos. Neste sentido, a especificação do PDF provê objetos para a representação de dados básicos, que podem ser utilizados em conjunto para a composição de objetos mais complexos ou apenas para auxiliar a descrição do conteúdo de forma mais flexível. Além disso, o formato PDF dispõe de maneiras para repre-

sentar e referenciar elementos, os quais serão aqueles desenhados nas páginas do documento: os objetos gráficos. Os gráficos podem ser subdivididos em grupos específicos conforme suas funcionalidades específicas no documento PDF. Assim, as seguintes seções apresentam uma breve descrição sobre os tipos básicos de objetos e sobre os gráficos encontrados em um PDF, ressaltando algumas das operações oferecidas para estes últimos.

3.2.1 Objetos Básicos

O PDF provê suporte para a definição de tipos de objetos básicos, que em conjunto formam todas as demais estruturas aplicadas no documento. Tais objetos podem ser utilizados de forma direta (objetos diretos), onde juntamente com a definição do objeto, já se está utilizando-o, ou de forma indireta (objetos indiretos), através da definição e utilização separadamente do mesmo. No segundo caso, para que um objeto possa ser utilizado, ele deve ser referenciado e, portanto, a ele é atribuído um identificador único. Este identificador é composto por um número inteiro positivo, chamado de número do objeto e um número de geração não negativo. O número de geração refere-se a versão do objeto que se está utilizando, onde o valor inicial é sempre 0. Caso este objeto seja atualizado por alguma aplicação, pode-se simplesmente atualizar tal número indicando a operação realizada. Desta forma, a definição de um objeto indireto consistirá em seu identificador único em conjunto com os delimitadores *obj* e *endobj*, que irão indicar o início e o fim do conteúdo propriamente dito de tal objeto. Por outro lado, a referência a um objeto indireto deve ser realizada através de seu identificador único e a palavra-chave *R*. A Figura 11 exemplifica os dois tipos de objeto mencionados, para uma melhor compreensão de suas composições.

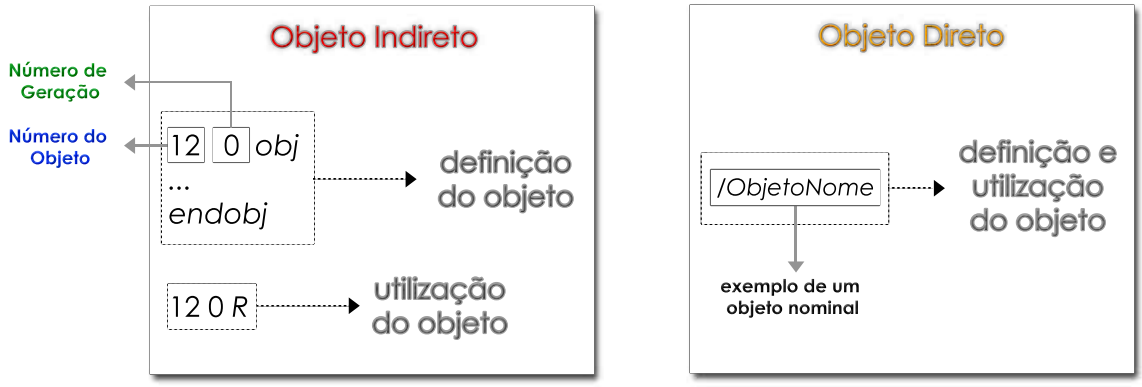

Figura 11 – Objeto direto e indireto.

No contexto da linguagem PDF, então, existem oito tipos básicos de objetos, cada um enquadrando-se na classificação de diretos e/ou indiretos. Estes são:

- Valores booleanos (Objetos booleanos);

- Valores numéricos (Objetos numéricos);
- *Strings*;
- Nomes (Objetos nominais);
- Vetores;
- Dicionários de dados;
- Objetos *stream*;
- Objeto nulo.

Os primeiros objetos, **valores booleanos**, são identificados através das palavras chave *true* ou *false*, podendo ser elementos de vetores ou de dicionários.

Os **valores numéricos** podem assumir valores inteiros ou reais. Na especificação do PDF, valores reais são representados através de um radical decimal (por exemplo 0.01) e não podem ser descritos com o uso de radicais não decimais (como $5\#FFFE$) ou em formato exponencial (como $5.02E06$).

Para a representação de texto são empregadas as ***strings***, que são objetos constituídos de uma série caracteres, representados como *bytes*. Estes objetos podem ser descritos como *strings* literais ou *strings* hexadecimais. As primeiras devem ser delimitadas por parêntesis, formando a seguinte estrutura: *(string literal)*. Assim sendo, as *strings* literais são compostas por caracteres quaisquer, com a ressalva de possuir um tratamento especial para parêntesis não balanceados e o caractere contra-barra (\). Por outro lado, as *strings* hexadecimais devem ser delimitadas pelos caracteres < e >, podendo conter caracteres hexadecimais (números de 0 a 9 e letras maiúsculas ou minúsculas de A a F). Neste contexto, cada par de caracteres hexadecimais representará um *byte* na *string*.

Os **nomes** são símbolos atômicos (sem nenhuma estrutura interna) unicamente definidos através de uma seqüência de caracteres. Isto significa que, quaisquer dois nomes compostos pelo mesmo conjunto de caracteres, representam o mesmo objeto. Neste contexto, um nome é definido através do caractere inicial barra (/), mas este não é parte do nome, apenas indica que os caracteres seguintes representam tal tipo objeto. Assim, um nome não é considerado como um texto que deverá ser apresentado para o usuário final, mas sim como um objeto interno ao documento.

No formato PDF, os **vetores** são elementos heterogêneos, ou seja, constituídos de qualquer combinação dos oito objetos explicitados. Neste sentido, o PDF suporta diretamente a construção de vetores unidimensionais, delimitados pelos caracteres *[* e *]*. Entretanto, devido a heterogeneidade provida pelos vetores, pode-se construí-los de maneira multi-dimensional, fazendo com que elementos de um vetor sejam outros vetores, podendo estar aninhados a qualquer profundidade.

Um **dicionário de dados** representa um mapa contendo um par de objetos: uma chave e um valor. Este par de elementos constituem uma entrada do dicionário. Neste sentido, uma chave deve sempre ser um nome, enquanto o valor pode ser qualquer outro tipo de objeto (incluindo outro dicionário). Para a definição deste tipo de objetos, deve-se utilizar os caracteres « e » como delimitadores de um dicionário. Cabe ressaltar que em um dicionário, por convenção, existe sempre uma entrada **Type** que identifica o tipo do objeto que o dicionário representa. Além disso, outra entrada **Subtype** pode ser utilizada para especificar uma categoria especializada do objeto especificado pelo tipo em questão.

Um objeto **stream** trata-se de uma seqüência de *bytes*, que pode ser lida de forma incremental. Este objeto não possui limitações quanto ao tamanho e é utilizado para representar conteúdos muito grandes, como imagens. Assim, todas as *streams* devem ser objetos indiretos, sendo constituídas por um dicionário, através do qual se especifica o número exato de *bytes* da *stream* em questão com a entrada **Length**, e do conteúdo do objeto propriamente dito, delimitado pelas palavras-chave *stream* e *endstream*.

Por fim, o objeto nulo é um objeto especial para indicar a inexistência de algum valor. Este objeto é único no documento PDF e pode ser utilizado através da palavra chave *null*.

Para uma melhor compreensão da estrutura dos objetos básicos descritos, estes são exemplificados na Figura 12.

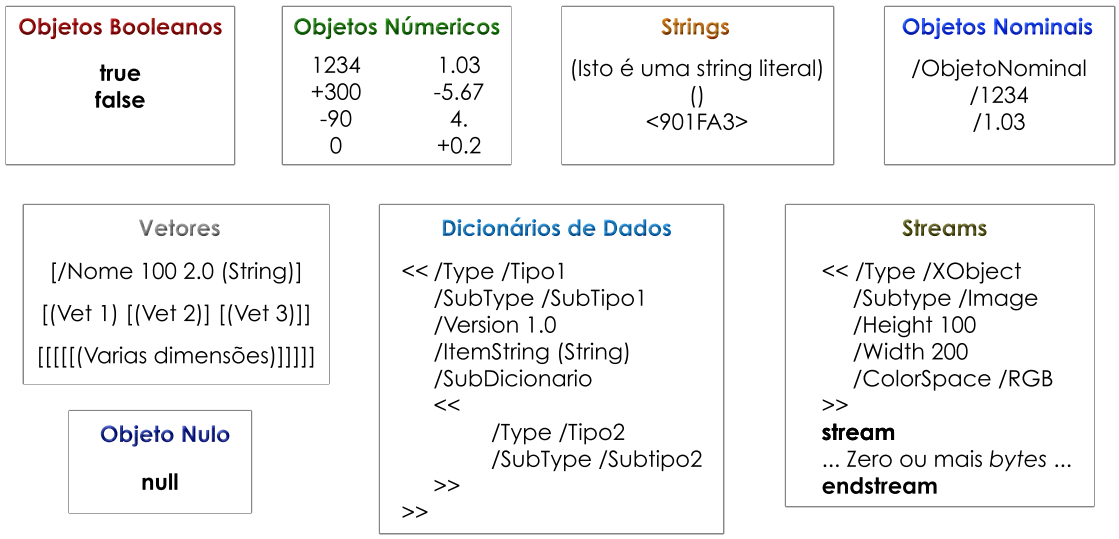

Figura 12 – Objetos básicos PDF.

3.2.2 Gráficos

Gráficos são elementos utilizados pelo formato PDF para representar o conteúdo de uma ou mais páginas. Os objetos gráficos possuem diversas características que os tornam flexíveis

e capazes de suprir diversas necessidades dos usuários para a definição de documentos de alta resolução. Além disso, para estes elementos é disponibilizada uma estrutura com o intuito de definir propriedades em comum para um conjunto de gráficos, chamada de estado dos gráficos (*graphics state*).

Nesta seção são especificados os tipos de gráficos presentes no cenário do PDF, juntamente com suas funcionalidades e a seguir são descritos os recursos oferecidos pelo estado dos gráficos, assim como as características gerais pertinentes a todos os objetos gráficos.

Tipos de Gráficos

Como mencionado anteriormente, o conteúdo que define a aparência de uma página trata-se de uma seqüência de operadores e operandos utilizados em conjunto com os recursos da página em questão. Neste contexto, operadores e operandos para a construção e o uso de objetos gráficos são utilizados no *content stream* de cada página. Assim, existem 5 objetos gráficos distintos que podem ser aplicados: *path objects*, *external objects* (*XObjects*), *inline image objects*, *shading patterns objects* e *text objects*. A seguir, serão descritas as características e funcionalidades destes objetos, ressaltando alguns operadores utilizados no contexto de cada um deles. Neste sentido, cabe lembrar que estes operadores são utilizados unicamente no contexto de um *content stream* de uma página e referem-se a palavras reservadas da linguagem PDF.

Os *path objects* representam caminhos que compõem formas arbitrárias, trajetórias e regiões, através do uso de retas e curvas cúbicas de Bézier. Sendo assim, um caminho é formado por segmentos de linha retos ou curvos, podendo estar conectados ou ser disconexos. Desta maneira, no formato PDF são fornecidas operações para os definir e preenchê-los, assim como estabelecer regiões de corte com o seu uso.

Através de operações de definição de caminhos é possível descrever a geometria dos caminhos propriamente ditos, sem quaisquer restrições. Permite-se, então, que um único caminho contenha regiões convexas, côncavas, áreas disjuntas ou regiões que se interseccionam. Entre os principais operadores para a construção de um caminho estão:

- **m**: dá início a construção de um caminho nas coordenadas x e y recebidas como parâmetro;
- **l**: adiciona uma reta ao caminho em questão, partindo do ponto atual para outro ponto x e y recebido como parâmetro. Nota-se que para a realização desta operação, deve-se ter realizado a operação **m** previamente;
- **re**: adiciona um retângulo ao caminho atual com o ponto inferior esquerdo (coordenadas x e y), sua largura e sua altura definidos respectivamente pelos operandos x, y, $width$ e $height$ passados por parâmetro.

As operações de preenchimento de um *path object* possibilitam desenhar de fato um caminho na página, seja através de funções para preencher as regiões explicitadas (*fill*) ou para pintar

um contorno no caminho em questão (*stroke*). Os principais operadores para tal função são:

- **f**: preenche o caminho atual;
- **s**: pinta um contorno no caminho em questão;
- **b**: preenche e pinta um contorno no caminho;
- **n**: finaliza um caminho sem preenche-lo ou contorná-lo (utilizado para regiões de corte).

Finalmente, as operações para estabelecer regiões de corte permitem que os caminhos sejam utilizados como regiões limites de corte para gráficos subseqüentes. O operador principal para este tipo de operação trata-se do **W**, que modifica a região de corte atual, através de operações com o caminho definido.

Outro tipo de objeto existente no formato PDF trata-se do *external object*, que é uma referência a um objeto externo ao conteúdo (*content stream*) da página em questão, sendo então referenciado através dos recursos de tal página. Estes objetos são *streams*, exclusivamente indiretas, e são pintadas na página através do comando **Do**. O PDF conta com três tipos distintos de *external objects*: *image XObjects*, *group (form) XObjects* e *postscript XObjects*. Para indicar qual o tipo do *XObject* que está definido, existirá sempre uma entrada **Subtype** no dicionário de suas respectivas *streams* indicando tal informação. Neste contexto, os *external objects* são os objetos que podem ser re-utilizados no cenário de rasterização de um documento PDF. A primeira referência a um dado *external object* será aquela realmente rasterizada pelos RIPs, que armazenarão o resultado de tal processamento em sua *cache*. As referências subseqüentes, então, não precisarão ser rasterizadas novamente, pois o RIP poderá apenas aplicar o resultado já armazenado.

Entre os tipos existentes de *XObjects*, as *image XObjects* dizem respeito a imagens que são inclusas no documento em um formato *bitmap* qualquer, como JPEG, PNG (*Portable Network Graphics*) e GIF (*Graphics Interchange Format*). Neste sentido, seu dicionário define parâmetros relacionados à imagem, como sua altura, largura e *color space*, através das entradas **Height**, **Width** e **ColorSpace** respectivamente, enquanto na sua *stream* estará o conteúdo da imagem propriamente dito. Por outro lado, os *group XObjects* são objetos utilizados para definir propriedades em comum dos objetos nele contidos. Assim, na *stream* deste grupo estarão as definições dos objetos pertinentes a ele. A especificação do PDF 1.7, define a existência de apenas um tipo de *group XObjects*, os *transparency group XObjects*, que podem estabelecer a transparência dos objetos do grupo. Finalmente, os *postscript XObjects* são utilizados para expressar comandos através da PDL PostScript, mas estes já não possuem uso efetivo no formato PDF e apenas são mantidos por questões de compatibilidade.

Os *text objects* tratam do texto de um documento. Por possuírem maior importância e mais ampla utilização na descrição de conteúdo, são oferecidas algumas funcionalidades específicas para lidar com estes. No âmbito de um objeto gráfico de texto, cabe ressaltar que o PDF diferencia os termos caractere e caractere tipográfico. O primeiro diz respeito ao elemento que

representa uma letra, número ou qualquer outro símbolo presente quando se está escrevendo um texto. O segundo refere-se ao desenho do caractere, que será pintado no documento, **dependendo da fonte que será escolhida** para um determinado conteúdo. A distinção destes termos deve ser destacada, pois é de grande importância para a compreensão da interpretação feita pelo PDF sobre objetos gráficos de texto. Assim que encontrada uma porção de texto no documento, o PDF realiza um mapeamento de cada caractere para o caractere tipográfico correspondente. Em termos de desempenho, este mapeamento pode ser de muito custoso, portanto o formato PDF provê operações de *caching* e reuso para contornar tal empecilho.

Tratando-se de funcionalidades específicas para textos, o formato PDF provê diversas operações, entre elas: ajuste do espaçamento entre caracteres, palavras e linhas; ajuste do tamanho horizontal e vertical ocupado pelas palavras, e o posicionamento das palavras na frase, possibilitando a aplicação de palavras sobrescritas e subscritas. Para tanto, são definidas diversas operações para os textos, dentre as quais devem ser destacadas as seguintes:

- **BT**: operação que indica o início de um *text object*;
- **Td**: move o objeto de texto atual para o ponto definido pelos operandos x e y recebidos por parâmetro;
- **Tf**: seleciona a fonte para o objeto de texto atual;
- **Tj**: operação para realmente pintar o texto na página;
- **ET**: operação que indica o final de um *text object*.

Os *inline image objects* são objetos auxiliares em um documento PDF e não são muito comuns. Estes são utilizados para definir uma imagem dentro do próprio *content stream* da página. Esta imagem possuirá diversas limitações, das quais a mais marcante trata-se do tamanho da imagem. Neste sentido, diz-se que a imagem em questão deverá possuir 4KB ou menos de conteúdo, caso contrário não poderá ser utilizado este tipo de objeto para a definição de tal imagem. Para a construção de uma *inline image* deve ser utilizado o operador **BI**, dando início ao objeto, seguido das propriedades da imagem com o uso do par nome / valor. Logo após, os operadores **ID** e **EI** serão empregados para delimitar o conteúdo da imagem em questão.

Os *shading patterns objects* também tratam-se de objetos não muito comuns em documentos PDF. Estes provêem uma transição suave entre cores em uma determinada área a ser pintada, independentemente de resolução de qualquer dispositivo de saída. Neste sentido, sua cor é definida por uma função arbitrária. Estes podem ser utilizados para que sejam obtidos efeitos em outras formas gráficas, como por exemplo, sombreamento e degradês. Eles são objetos *stream* ou apenas dicionários, onde poderá ser encontrada a função necessária para pintar as cores especificadas. Estes objetos são diretos e podem ser pintados através do operador **sh** seguidos de um dicionário / *stream*.

Características Gerais e Estado dos Gráficos

Uma estrutura muito importante no formato PDF trata-se do estado dos gráficos. Em suma, este elemento mantém os parâmetros de controle para pintar os gráficos que se encontram no seu escopo. Estes parâmetros definem uma série de informações sobre as quais os operadores dos gráficos são executados. Desta forma, para os gráficos do conteúdo de uma página sempre existirá um estado dos gráficos corrente, que deverá ser considerado para apresentar os gráficos de tal página. No *content stream* de cada página este estado dos gráficos atual é sempre re-inicializado, passando a conter, então, valores padrões que serão explicitados no decorrer desta seção.

Um documento PDF bem estruturado normalmente possui diversos elementos gráficos que são independentes um do outro. Assim, diversas modificações no estado dos gráficos podem ser necessárias, sendo que pode ser desejado utilizar um mesmo estado dos gráficos várias vezes em diferentes pontos do documento. Para tal propósito, o formato PDF provê uma maneira de salvar o estado dos gráficos atual, para que este possa ser restaurado assim que for preciso. Previne-se, assim, a necessidade de estabelecer todos os parâmetros do estado dos gráficos toda vez que se deseje re-utilizar algum previamente definido. Os operadores para salvar e restaurar o estado dos gráficos são respectivamente: **q** e **Q**. Desta maneira, para realizar estas operações sobre os estados dos gráficos é utilizada uma pilha LIFO (*Last In First Out*) (chamada de *graphics state stack*), onde sempre no topo da pilha estará o último estado dos gráficos salvo. Neste contexto, à medida que se salvar o estado dos gráficos atual, este será colocado no topo da pilha. Por outro lado, quando se restaurar um estado dos gráficos, aquele do topo da pilha será removido e atribuído ao estado dos gráficos corrente. Através de tais operadores, então, será possível existir diversos estados dos gráficos no documento, mas vale a pena ressaltar que apenas um estará em uso. Como dito anteriormente, este estado dos gráficos corrente possuirá diversos recursos que influenciarão no comportamento dos operadores dos gráficos, que serão descritos a seguir.

Um dos recursos mais importantes oferecidos pelo estado dos gráficos trata-se da disponibilidade de um sistema de coordenadas especial. Com o objetivo de garantir a portabilidade de um documento PDF, este formato utiliza um sistema de coordenadas próprio, chamado de *user space*. Este sistema de coordenadas possui sempre a mesma relação com a página que está sendo considerada, independentemente do dispositivo de saída o qual se está tentando imprimir. Em outras palavras, este sistema de coordenadas é independente da resolução do dispositivo o qual se deseja apresentar o documento, evitando diferente interpretações dos gráficos à medida que seja necessário apresentar o documento em dispositivos com resoluções distintas (Figura 13).

Para que seja possível, então, realizar a conversão do sistema de coordenadas do documento para o do dispositivo, é utilizada uma matriz de transformação, conhecida como *Current Transformation Matrix* (CTM). Esta matriz é mantida no estado dos gráficos, como uma entrada de tal dicionário. A partir desta matriz também é possível realizar transformações nos gráficos

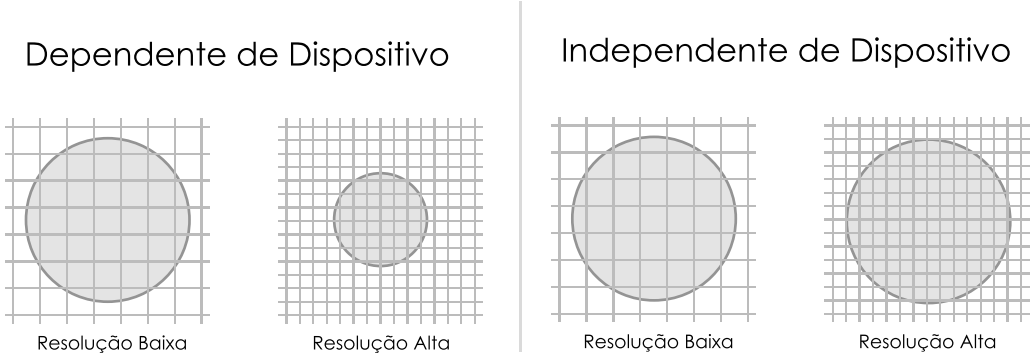

Figura 13 – Dependência e independência de dispositivo.

criados, como, por exemplo, rotações, escalas, distorções e translações. Através deste sistema de coordenadas próprio, versões mais recentes do formato PDF permitem a utilização de três dimensões para a descrição de gráficos mais complexos. Com isso, pode-se naturalmente descrever componentes gráficos sem ter que se preocupar em converter o sistema de coordenadas para duas dimensões, pois isto será feito automaticamente pelo interpretador do documento PDF, que irá utilizar a CTM correspondente para cada gráfico em questão. Cabe ressaltar que o valor padrão de uma CTM de um estado dos gráficos é sempre uma sem quaisquer transformações.

Outra característica existente para a obtenção de documentos com mais recursos visuais trata-se da utilização de transparência. Elementos gráficos em um documento PDF foram definidos até a sua quarta versão (versão 1.3) através da utilização de um modo de imagem totalmente opaco, onde cada gráfico seria desenhado em seqüência, sobrescrevendo qualquer porção de outro gráfico a qual viesse a interseccionar. A seguinte versão do PDF (versão 1.4) introduziu um modo de imagem transparente em que os elementos gráficos não necessariamente estariam totalmente opacos, permitindo que estes transparecessem. Desta maneira, através da utilização de tal funcionalidade, tornou-se possível estabelecer uma quantidade arbitrária de opacidade para cada gráfico existente no documento. A opacidade dos gráficos pode ser estabelecida através do estado dos gráficos, por meio de uma entrada do dicionário: ***alpha constant***. Esta entrada nada mais é do que um objeto númerico (real), cujo valor pode variar de 0.0 à 1.0. Neste contexto, o valor 1.0 representará objetos totalmente opacos, enquanto o valor 0.0, objetos totalmente transparentes. Para a mudança do valor de opacidade do estado dos gráficos corrente, utiliza-se o operador **gs**. Neste contexto, um estado dos gráficos inicial de uma página será definido como totalmente opaco, ou seja, com o *alpha constant = 1.0*.

Em termos de definição de cores para cada elemento gráfico, a versão atual do PDF provê diversos modos distintos a serem utilizados. Neste sentido, a grande vantagem oferecida pelo PDF é que estes modos são totalmente independentes do dispositivo. Para a apresentação do conteúdo descrito no documento, tais modos de cores são convertidos para o modo de cor presente no dispositivo em questão. Desta forma, o usuário poderá aplicar os modos de cores

que o convém, sem ter a preocupação de qual deve ser utilizado em um ou outro dispositivo específico. Entre alguns dos modos disponíveis no PDF, estão o RGB (*Red Green Blue*) e o CMYK (*Cyan Magenta Yellow Key*). Por outro lado, o formato PDF não restringe a utilização somente de modos de cor pré-definidos, permitindo ao usuário a criação de seus próprios, para então aplicá-los aos gráficos desejados. Os modos de cores devem ser especificados no estado dos gráficos, na entrada ***color space***, que, então, será aplicado para os gráficos subseqüentes. Vale a pena mencionar que o PDF permite o uso de diversos modos de cores distintos em um mesmo documento. Além disso, é no estado dos gráficos que estará a cor que deve ser aplicada, quando pintando *text objects* e *path objects*. No contexto de cores para o estado dos gráficos, o modo de cor padrão para um estado dos gráficos trata-se do RGB e a cor padrão é o preto.

4 Escalonamento

Durante o desenvolvimento de uma versão paralela de um programa diversas características devem ser levadas em consideração para maximizar o desempenho que se deseja adquirir. Neste contexto, uma das primeiras preocupações é a escolha do melhor tamanho de tarefa (grão) a ser transmitido e/ou processado por cada unidade ativa de processamento [14]. Em termos de unidades de processamento que possuam memória distribuída, a escolha do grão é um aspecto crítico para o bom funcionamento da aplicação. Caso se defina um grão ruim, o *overhead* existente para a transmissão ou quebra das tarefas pode não compensar o ganho de desempenho obtido através da divisão do trabalho. Neste sentido, pode-se afirmar que esta escolha deve ser baseada nas características específicas da aplicação e da arquitetura, buscando pela relação ótima entre a quebra das tarefas, em conjunto com o custo de comunicação, e o ganho de processamento obtido através da paralelização.

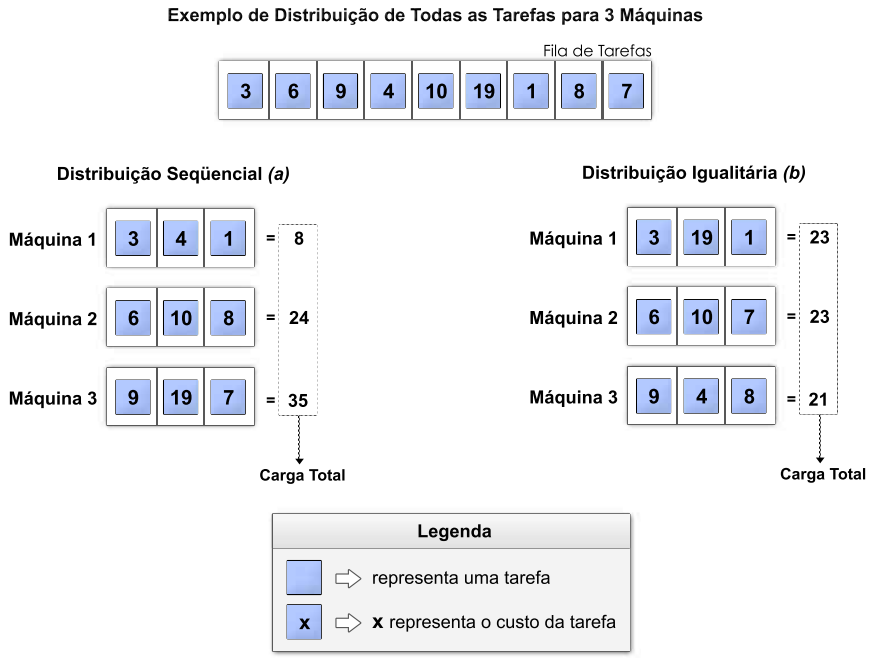

Figura 14 – Influência das estratégias de escalonamento.

Assim que o grão estiver definido é necessário estabelecer a melhor estratégia de distribuição das tarefas disponíveis entre os processos, de maneira que nenhum destes fique sobrecarregado ou sub-carregado. Caso a estratégia escolhida não represente uma boa escolha, a eficiência da paralelização pode ficar diretamente comprometida. A Figura 14 demonstra uma situação em como a distribuição das tarefas pode influenciar diretamente no ganho ou perda de desempenho

potencial que poderia ser encontrado. Nesta figura, pode-se notar que com uma distribuição seqüencial (Figura 14 *(a)*), na qual associam-se as tarefas uma a uma para cada máquina de forma circular (a primeira tarefa para a máquina 1, a segunda para a máquina 2, a terceira para a máquina 3 e assim por diante), tem-se uma carga total na máquina 3 de 35. Considerando que o processamento de toda a fila de tarefas será dito como finalizado, assim que todas as máquinas terminem suas tarefas, este valor (sendo o maior existente) representa tal momento. Por outro lado, buscando uma distribuição igualitária (Figura 14 *(b)*), onde a carga de cada tarefa é levada em consideração *a priori* a sua distribuição, tem-se uma carga total de 23, diminuindo então o tempo de finalização de todas as tarefas. Desta maneira, a procura pela divisão ideal de tarefas carateriza o problema de escalonamento [15]. Neste contexto, o objetivo final é uma configuração de distribuição de tarefas de forma que os recursos envolvidos sejam aproveitados da melhor maneira possível, buscando pela otimização de uma medida de desempenho arbitrária.

Neste capítulo serão abordados aspectos relacionados ao problema de escalonamento, fornecendo uma base teórica para a implementação de uma aplicação paralela capaz de superar os desempenhos obtidos com as estratégias de rasterização existentes. Neste contexto, são apresentadas: uma notação para o discernimento de situações em que se deve aplicar distintas estratégias de escalonamento, uma classificação para os problemas de escalonamento e alguns algoritmos de escalonamento existentes, assim como aspectos relacionados diretamente a suas respectivas eficiências.

4.1 Notação

Graham *et al.* [16] introduziram a notação $\alpha|\beta|\gamma$ com a finalidade de agrupar problemas de escalonamento com características em comum, permitindo, assim, uma análise mais objetiva de aspectos que influenciam cada grupo. Nesta notação, cada campo representa uma característica diferente quanto ao problema de escalonamento que deve ser considerada.

O campo α refere-se ao ambiente de execução da estratégia de escalonamento, relacionado diretamente ao *hardware* disponível. Alguns dos valores possíveis são apresentados a seguir:

- **Máquina única** (1): existe apenas uma máquina no sistema;
- **Máquinas paralelas e idênticas** (Pm): existem m máquinas idênticas em paralelo no sistema. Ao omitir-se o valor m se estabelece que o número de máquinas é arbitrário, ou seja, o grupo comporta qualquer número de máquinas idênticas (sendo este valor maior do que 1);
- **Máquinas não relacionadas** (Rm): existem m máquinas em paralelo no sistema, mas cada uma delas pode processar as tarefas em tempos distintos. Isto significa que as máquinas podem possuir *hardware* e velocidades distintas. Novamente, pode-se omitir o

valor m para se referenciar um número arbitrário qualquer de máquinas (maior do que 1).

O campo β diz respeito às peculiaridades de cada tarefa e restrições da estratégia de escalonamento que devem ser respeitadas. Entre os valores possíveis para este campo estão:

- **Preempção** ($pmtn$): tarefas podem ser preemptadas e resumidas mais tarde, possivelmente em uma máquina diferente. Se preempções são permitidas, este valor estará incluso no campo β, caso contrário tal operação não é permitida;
- **Restrições de precedência** ($prec$): este campo especifica que existem tarefas da fila que devem ser completadas antes que determinadas tarefas iniciem seu processamento. Caso *prec* não esteja especificado no campo β, as tarefas não estão sujeitas a restrições de precedência;
- **Prazos** (d_j): se este símbolo estiver presente, cada tarefa j deve ser completada até o seu prazo final d_j. Caso, contrário as tarefas não necessitam ser completadas até um determinado prazo.

Por fim, o campo γ se trata da medida de desempenho que se deseja otimizar com a aplicação da estratégia de escalonamento. Tais medidas estão sempre relacionadas ao tempo para completar as tarefas. Neste contexto, algumas definições são utilizadas para formalizar as subseqüentes medidas de desempenho:

- C_j: denota o tempo para completar a tarefa j, sendo que este tempo refere-se ao momento em que a tarefa j foi inserida na fila, até o seu processamento por completo;
- L_j: diz respeito ao atraso da tarefa j, onde $L_j = C_j - d_j$;
- T_j: trata-se do atraso encontrado para a tarefa j, no qual $T_j = max(L_j, 0)$.

Com o uso destas definições, então, algumas medidas de desempenho que podem ser descritas no campo em questão são:

- ***Makespan*** (C_{max}): se refere ao tempo para finalizar todas as tarefas de uma determinada fila. Neste sentido, ele pode ser definido como o $max(C_1, ..., C_n)$;
- ***Maximum lateness*** (L_{max}): trata-se do maior atraso para as tarefas da fila. Assim, esta medida será calculada como $max(L_1, ..., L_n)$;
- ***Total tardiness*** ($\sum T_j$): é o atraso encontrado nas em todas as tarefas da fila, sendo computado através do somatório dos atrasos de todas as tarefas.

4.2 Classificação

O escalonamento pode ser classificado quanto a duas situações distintas [15], dependendo da disponibilidade de informações sobre as tarefas: determinístico e não determinístico. A seguir são apresentadas particularidades sobre cada uma destas categorias, juntamente com alguns tipos de algoritmo a elas associados.

4.2.1 Escalonamento Determinístico

Técnicas de escalonamento pertinentes a esta categoria são aquelas em que todas as características das tarefas e as relações entre cada uma delas são conhecidas previamente à execução da aplicação. Neste contexto, destacam-se dois tipos de algoritmos: ótimos e sub-ótimos.

Algoritmos ótimos preocupam-se em encontrar a melhor solução para o problema de escalonamento, baseando-se em um conjunto de informações sobre as tarefas da aplicação. Entretanto, na maioria dos casos para que tal solução seja de fato descoberta, necessita-se de uma quantidade muito grande de informações. Além disso, sabe-se que o caso geral e diversos particulares dos problemas de escalonamento são NP-completos [17, 18]. Contudo, para alguns poucos casos existem algoritmos com a capacidade de resolver o problema de escalonamento em tempo polinomial, os quais conseguem apresentar soluções ótimas.

O caso ideal seria o emprego de algoritmos ótimos para todas as situações, desta maneira se teria a garantia que sempre a melhor configuração, em termos de divisão de tarefas, seria utilizada. No entanto, este tipo de algoritmos pode ser muito custoso em termos de desempenho e quantidade de informações necessárias, além de apenas existir para um pequeno conjunto das situações. Devido a este fato, os algoritmos sub-ótimos buscam uma solução através do uso de técnicas específicas, como a heurística, sem nenhuma garantia que é a melhor para o dado problema, mas sabe-se que a solução encontrada é ao menos próxima da ótima. Apesar disso, com o uso destes podem-se obter respostas em um tempo polinomial, o que não seria possível em diversos casos com o emprego de um algoritmo ótimo. Neste sentido, algoritmos sub-ótimos tem a capacidade de englobar uma gama muito maior de resoluções para problemas do que os ótimos, além de conseguirem solucionar os problemas em um tempo significativamente menor.

4.2.2 Escalonamento Não Determinístico

O escalonamento é dito não determinístico quando não se possui todas as informações necessárias sobre as tarefas antes da execução do programa paralelo. Desta maneira, este tipo de escalonamento provê uma decisão sobre como distribuir as tarefas durante a execução do pro-

grama e por isso acarreta um *overhead* na própria aplicação que deve ser minimizado. Neste contexto, dois tipos de algoritmos devem ser destacados: dinâmicos e estáticos.

Algoritmos estáticos são aqueles realizam uma análise sobre todas as tarefas antes de distribuir qualquer uma, possibilitando assim, a obtenção da melhor maneira de realizar o escalonamento para o todo o conjunto em questão. É importante ressaltar que esta análise é realizada durante a execução do programa e por isso se enquadra na categoria de escalonamento não determinístico. Esta abordagem torna-se vantajosa, caso o processamento desta análise consiga ser realizado em um espaço de tempo que seja compensado pelo ganho de desempenho obtido através da estratégia de escalonamento estabelecida.

Contudo, existem casos em que não é possível realizar a análise de todas as tarefas em conjunto, pelo fato de que todas elas não estão disponíveis ou porque o custo desta análise é muito grande. Desta maneira, algoritmos dinâmicos propõem a intercalação do processo de análise com a distribuição das tarefas. Em outras palavras, se realiza a análise apenas de um conjunto de tarefas, estas são distribuídas para as unidades ativas de processamento, que iniciam a computação, para então se partir para a análise de outro conjunto. Desta forma, pode-se afirmar que a partir de um certo ponto, o processo de escalonamento passa a ser concorrente com o cálculo do custo computacional de cada tarefa.

4.3 O Problema $Pm||C_{max}$

Considerando a notação $\alpha|\beta|\gamma$, dentre os problemas existentes na área de escalonamento, um dos fundamentais é denominado $Pm||C_{max}$ [19], que diz respeito ao escalonamento de um conjunto de n tarefas $\delta = T_1, T_2, ..., T_n$ em m máquinas idênticas, objetivando a redução do *makespan*, ou seja, do tempo para completar a última tarefa (C_{max}). Este problema mostrou-se como NP-difícil [20], provocando a impossibilidade de criação de algoritmos ótimos para resolver este problema.

Esta seção apresenta características relacionadas a este problema de escalonamento, descrevendo alguns algoritmos empregados nesta situação para a resolução de tal problema.

4.3.1 Análise de Competitividade

Como mencionado anteriormente, o problema de escalonamento $Pm||C_{max}$ é NP-difícil e portanto diversos algoritmos foram desenvolvidos a partir de heurísticas para a obtenção de um resultado o mais perto do ótimo possível. Neste sentido, uma abordagem comumente aplicada para a avaliação de algoritmos sub-ótimos no contexto de problemas NP-difíceis é empregada para analisar suas respectivas eficiências. Tal método trata da distância do resultado do algoritmo em questão, considerando o seu pior caso, em relação ao ótimo. No cenário de escalona-

mento esta abordagem é chamada de análise de competitividade [21, 22].

Para estabelecer a competitividade de um algoritmo de escalonamento A qualquer, algumas formalizações são necessárias. Denota-se que $f(A, I)$ é o resultado do escalonamento produzido pelo algoritmo A sobre a fila de tarefas de entrada I. Com isso, f representa a medida de desempenho que se deseja otimizar. Neste trabalho considera-se que $f = C_{max}$. Além disso, estabelece-se que a solução ótima para o escalonamento é obtida quando $A = OPT$. Então, partindo destas constatações, pode-se dizer que o algoritmo A é *c-competitive* caso $f(A, I) \leq c * f(OPT, I)$, para qualquer I. Em outras palavras, um algoritmo é *c-competitive* se no seu pior caso, este é capaz de gerar um resultado que se afasta até c vezes do ótimo. Por outro lado, é importante ressaltar que um algoritmo com uma competitividade melhor do que outro não necessariamente gerará um melhor resultado, mas apenas será garantido que o resultado, no seu pior caso, não ultrapassará a competitividade especificada.

4.3.2 Algoritmos de Escalonamento

Atualmente, diversos algoritmos podem ser encontrados para resolver o problema $Pm||C_{max}$ [16, 23–27]. Entre estes, os algoritmos mais difundidos são: *List Scheduling* (LS) [23], *Largest Processing Time first* (LPT) [16] e *Multifit* [24]. Esta seção discorre sobre o funcionamento de cada um destes algoritmos, com o objetivo de apresentar suas vantagens e desvantagens, ressaltando suas competitividades.

List Scheduling

O algoritmo LS foi introduzido por Graham na década de 60 e tornou-se a base para o desenvolvimento de diversos outros algoritmos de escalonamento. Além disso, a partir deste algoritmo, descobriram-se outros problemas na área de escalonamento, como a consideração de restrições de precedência para as tarefas da fila. Neste algoritmo, determina-se que para uma fila δ de tarefas organizadas em qualquer ordem, deve-se sempre transmitir a primeira tarefa para uma máquina ociosa qualquer. A Figura 15 apresenta o funcionamento desta abordagem sobre uma fila de tarefas arbitrária.

No exemplo apresentado na Figura 15, deve-se ressaltar que uma tarefa é associada a uma máquina somente quando tal máquina estiver ociosa, ou seja, assim que esta terminar por completo sua tarefa atual. Devido a tal fato, pode-se notar, por exemplo, que a máquina 1, recebe uma nova tarefa somente após terminar sua primeira, que ocorre no tempo 9. Sendo assim, a nova tarefa recebida pela máquina 1 será a 7, pois as anteriores foram distribuídas para as outras máquinas à medida que estas terminaram suas respectivas tarefas.

Considerando a análise competitiva para m máquinas, este algoritmo é $(2 - \frac{1}{m})$-*competitive* [16], significando que sua competitividade piora à medida que mais máquinas são utilizadas. Neste sentido, o pior caso do algoritmo LS ocorre quando a última tarefa é aquela com o maior

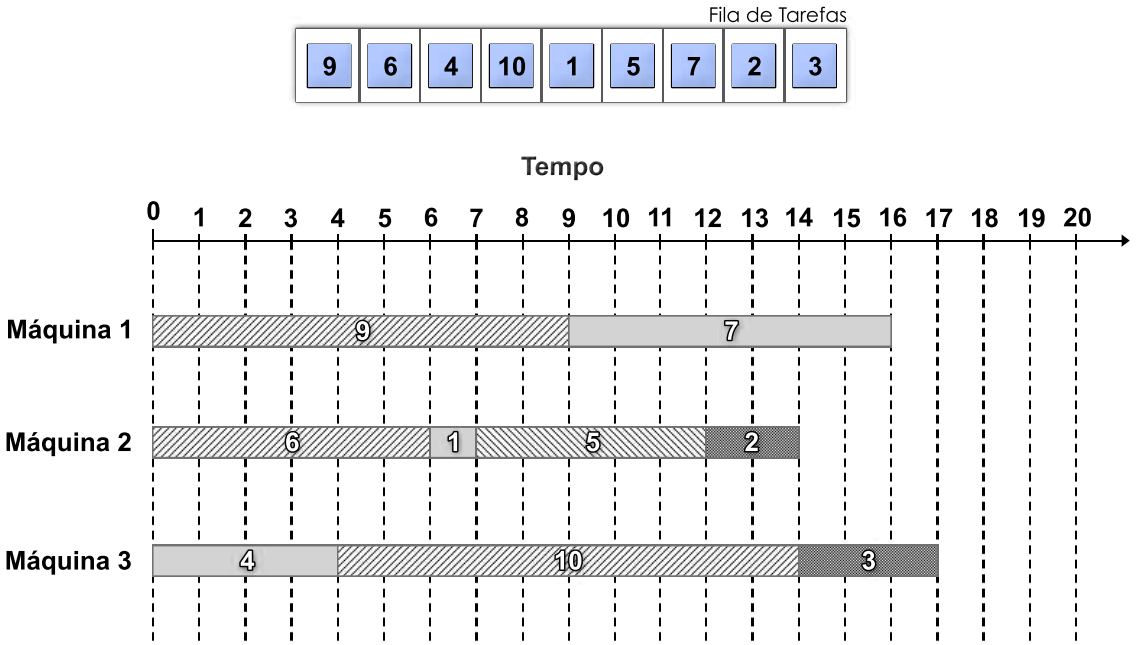

Figura 15 – Algoritmo *LS*.

tempo de processamento da fila, pois provocará com que apenas uma máquina fique sobrecarregada com esta tarefa grande, enquanto todas as outras máquinas já tenham terminado suas respectivas tarefas menores (Figura 16).

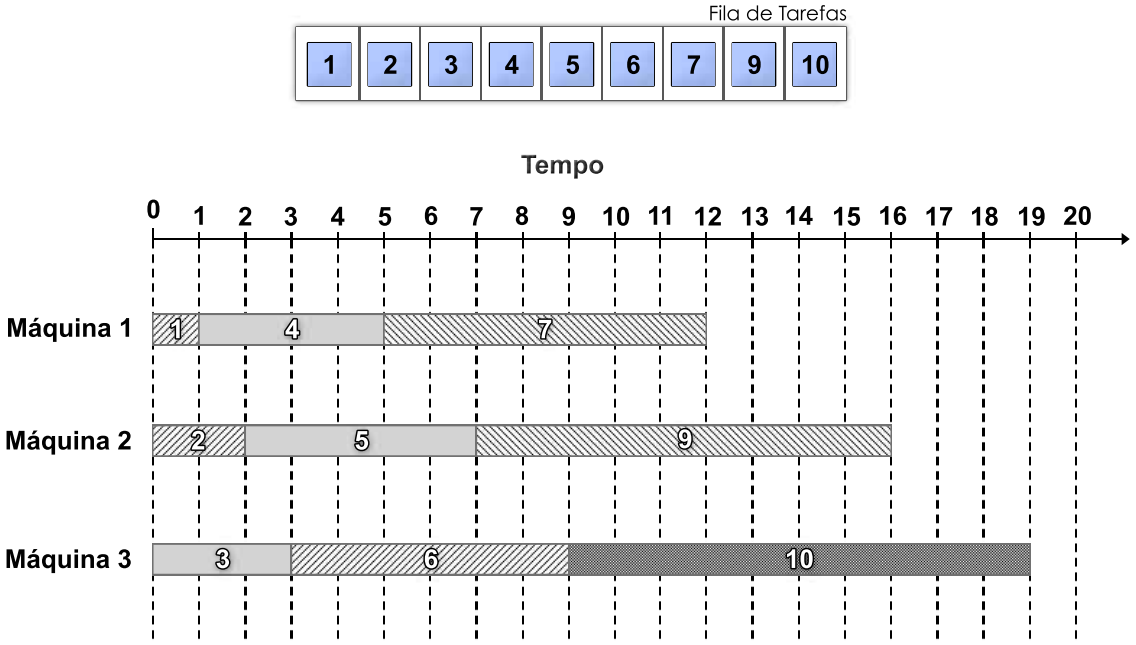

Figura 16 – Pior caso do algoritmo *LS*.

Largest Processing Time First

Com o intuito de melhorar a competitividade do algoritmo LS, Graham propôs a construção de um novo algoritmo conhecido como LPT. Para tanto, Graham focou na tentativa de prevenção do pior caso do algoritmo LS, que ocorre quando a última tarefa é aquela com a maior tempo de processamento de δ. Neste sentido, o algoritmo LPT introduz a ordenação das tarefas de forma decrescente, fazendo com que aquelas com maiores tempos de processamento estejam entre as primeiras posições de δ. Feito isso, aplica-se o algoritmo LS sem nenhuma outra alteração. Desta maneira, evita-se o processamento de tarefas grandes ao final do escalonamento, amenizando o efeito de perda de desempenho provocada pelo não balanceamento de cargas grandes entre os processos (Figura 17).

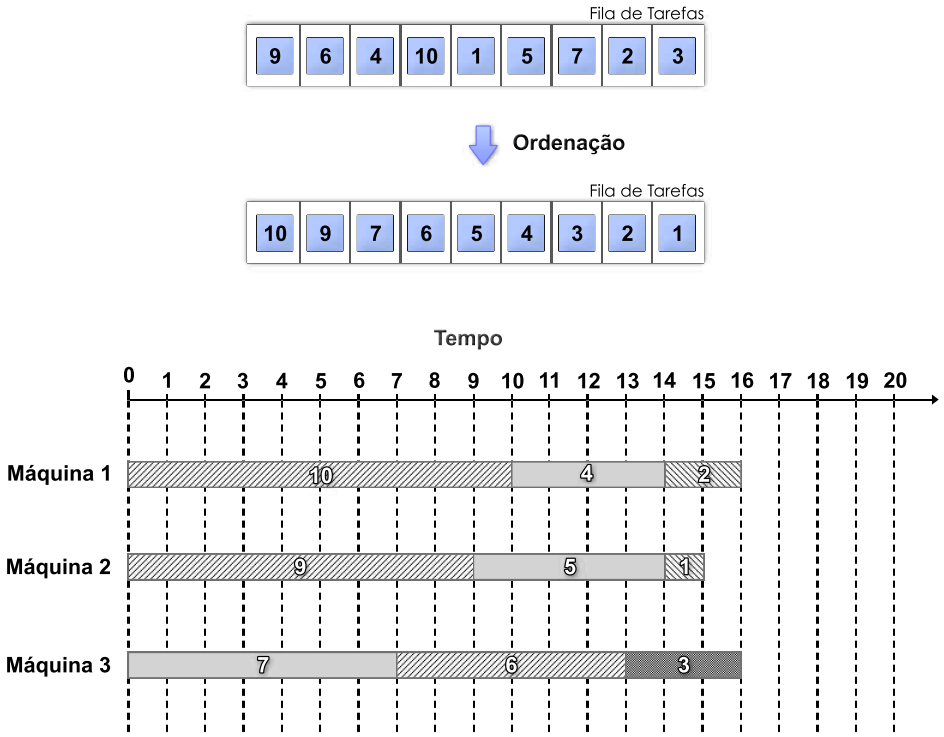

Figura 17 – Algoritmo LPT.

Com esta simples mudança, o algoritmo LPT representou um grande avanço em relação ao LS, apresentando uma eficiência $(\frac{4}{3} - \frac{1}{3m})$-*competitive* [23]. Como se pode notar, a competitividade deste algoritmo fica pior à medida que mais máquinas são introduzidas no escalonamento.

Multifit

A partir do estabelecimento do LPT, um dos primeiros algoritmos a apresentar um ganho realmente significativo foi o *Multifit*. Este algoritmo foi baseado em técnicas de *bin-packing* [28], que tratam do empacotamento de n tarefas, com cargas quaisquer, em um número finito de *bins* (pacotes) de forma que o número de *bins* utilizado seja o menor possível. O problema de

bin-packing é NP-completo [29] e portanto diversas heurísticas foram estabelecidas para resolver este problema. Neste sentido, o algoritmo *Multifit* utiliza a heurística *First-Fit Decreasing* (FFD) para agrupar as tarefas da fila em até m *bins*, considerando que a carga de um *bin* trata-se do tempo de processamento das tarefas nele contidas. A estratégia FFD funciona da seguinte maneira:

1. organiza-se as tarefas conforme seus tempos de processamento de forma decrescente em uma sequência;

2. cada tarefa da sequência é adicionada no *bin* com a menor carga dentre todos, de forma que a capacidade do *bin* não ultrapasse um limite pré-definido C;

3. os passos 1 e 2 são executados até que nenhuma todas as tarefas estejam em um *bin*.

Neste contexto, para encontrar o *bin-packing* ótimo, o limite C teria que ser o *makespan* ótimo para melhor configuração de *bins* possível, ou seja, $C = max(l(B_i))$, onde $l(B_i)$ denota a carga do *bin* B_i, para $1 \leq i \leq m$. Entretanto, para a situação mencionada descobrir $max(l(B_i))$ é tão difícil quanto descobrir a menor divisão das tarefas em até m *bins* [24].

Para que um bom (sub-ótimo) limite C seja encontrado em tempo polinomial, métodos de busca iterativa são empregados, estipulando um C inicial e refinando este valor ao longo da computação. Desta maneira, a cada iteração da busca será executada a técnica FFD, considerando o C atual. A busca será encerrada caso k iterações sejam atingidas. Dentre os métodos de busca existente, utiliza-se a busca binária no algoritmo *Multifit*.

O limite C é obtido através da média de um limite inferior, C_l, e outro superior C_{up}, que respectivamente referem-se ao melhor caso e pior caso possíveis para o empacotamento das tarefas da fila. O limite inferior é inicializado com o *makespan* ótimo para δ, calculado através do máximo entre carga total da fila de tarefas δ dividido por m e a maior carga das tarefas $T_k \in \delta$. Por outro lado, o limite superior é obtido através do máximo entre o dobro da carga total da fila de tarefas δ dividido por m e a maior carga das tarefas em δ.

Definido este valor inicial de C, passa-se para a busca binária do menor C possível até que k iterações sejam atingidas. Neste sentido, uma iteração corresponde a execução do algoritmo FFD considerando o C atual. Assim, no contexto de uma iteração, caso o resultado do algoritmo FFD, sejam mais de m *bins*, C_l receberá o C atual e C_{up} será mantido. Caso o resultado do FFD sejam até m *bins*, refina-se o valor do limite superior, onde C_{up} receberá o C atual e se manterá o limite inferior. Com isso, passa-se para a próxima iteração, onde C receberá novamente uma média entre C_l e C_{up}. No momento, em que k iterações sejam atingidas, C_{up} conterá o menor valor para C encontrado nesta busca binária. Caso na busca binária não seja encontrado nenhum C, tal que o FFD correspondente gere até m *bins* diz-se que C será o limite superior inicial. Em ambos os caso através da aplicação do FFD mais uma vez, com a capacidade C pré-determinada, poderão ser gerados até m *bins*. Um exemplo da aplicação do algoritmo *Multifit* é ilustrado na Figura 18.

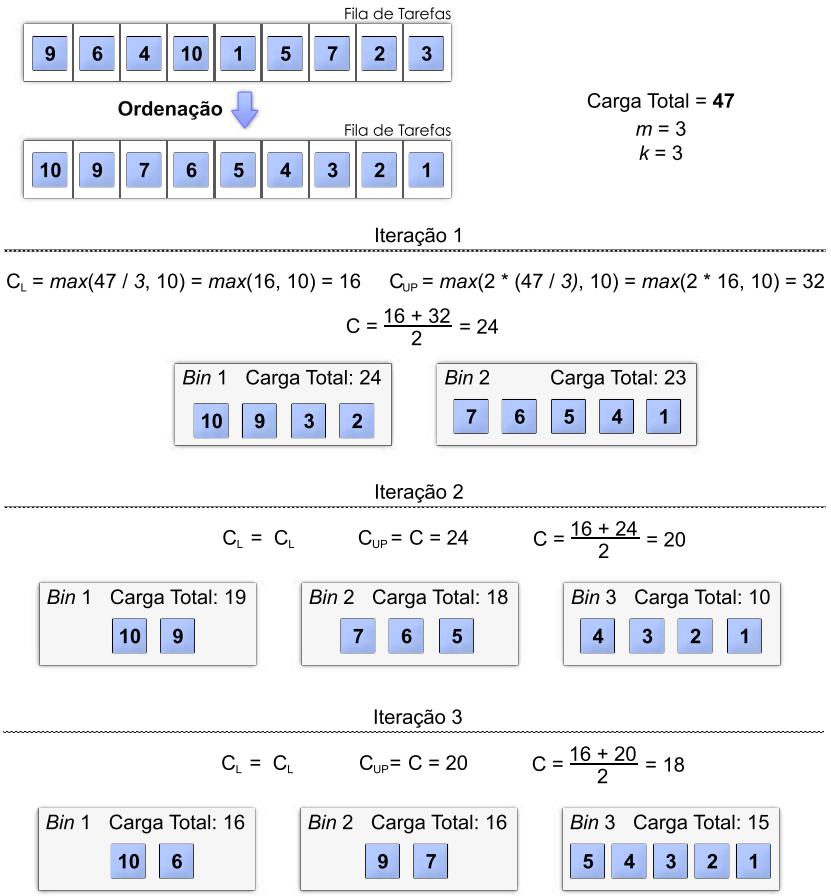

Figura 18 – Algoritmo *Multifit*.

Como resultado final do algoritmo *Multifit*, então, serão obtidos até m *bins* com cargas similares, que poderão ser escalonados em até m máquinas. A eficiência deste algoritmo mostrou-se $(\rho + 2^{-k})$-*competitive* [24], para $1.176 \leq \rho \leq 1.22$.

5 Abordagem Proposta

As estratégias existentes no âmbito da rasterização de documentos enquadram-se na classificação de problemas de escalonamento do cenário $Pm||C_{max}$, aplicando diferentes formas de quebrar os *jobs* para distribuí-los entre os RIPs disponíveis. Neste contexto, pode-se perceber que todas estas estratégias utilizam o algoritmo LS para a distribuição de tarefas. Como descrito anteriormente, sabe-se que tal algoritmo foi um dos primeiros estabelecidos na área de escalonamento, não possuindo a melhor competitividade dentre os algoritmos existentes. Além disso, pode-se afirmar que a natureza dos *jobs* das PSPs é diversificada, provocando a existência de um ambiente com uma ampla variação de cargas. Assim, devido a nenhuma consideração quanto ao custo computacional das tarefas por parte do algoritmo LS, é provável a ocorrência de anomalias no desempenho de tais estratégias, observando-se comportamentos estranhos, como picos e quedas bruscas em curvas de tempo de execução. Este fato, compromete as estratégias existentes, que não apresentam um comportamento previsível e confiável.

Com o intuito de melhorar a situação existente, pode-se aplicar algoritmos de escalonamento que utilizam heurísticas baseadas no custo computacional das tarefas, na tentativa de obter um balanceamento de cargas mais justo. Neste cenário, pode-se dizer que se está lidando com o escalonamento não-determinístico dinâmico, pois nenhuma informação sobre as tarefas é conhecida previamente à execução do escalonamento e que tais tarefas podem ser inseridas na fila a qualquer momento. Assim, propõe-se o emprego dos algoritmos LPT e *Multifit*, que são indicados para lidar com problemas $Pm||C_{max}$, por superarem a eficiência do algoritmo LS em termos de competitividade, além de poderem ser empregados no problema de escalonamento não-determinístico e dinâmico. Para satisfazer este objetivo, torna-se necessária uma maneira de estimar os custos computacionais de cada tarefa. Portanto, é preciso definir métricas para analisar o perfil dos *jobs*, que poderão ser aplicadas através do uso de uma ferramenta capaz de extrair as informações necessárias. Além disto, para a aplicação dos algoritmos é interessante a distribuição de cargas não muito grandes, com a finalidade de evitar que alguns RIPs fiquem com *jobs* muito grandes enquanto outros processem *jobs* pequenos. Assim, outra ferramenta auxiliar deve ser utilizada para quebrar os *jobs* em tarefas, de forma a diminuir o grão de trabalho.

Neste capítulo, então, é apresentado o trabalho desenvolvido para melhorar o desempenho das estratégias existentes. Desta maneira, em um primeiro momento são demonstradas as métricas obtidas para estimar o esforço que será realizado a fim de rasterizar um determinado *job* ou parte deste. A seguir, é descrita a ferramenta *PDF Profiler*, responsável por extrair as informações necessárias dos documentos, possibilitando a aplicação das métricas. Logo após,

descreve-se a aplicação *PDF Splitter* que possibilita a quebra dos *jobs* em porções menores, com a finalidade de diminuir o seu grão. Por fim, explica-se os escalonadores implementados, que utilizam as ferramentas e as métricas desenvolvidas. Estes escalonadores aplicam os algoritmos LPT e *Multifit* na tentativa de melhorar o desempenho da fase de rasterização, onde para o LPT propõem-se uma otimização relativa a análise das tarefas realizada sobre os *jobs*, na tentativa de possibilitar ao escalonador uma distribuição mais imediata das tarefas aos RIPs ociosos.

5.1 Métricas

Diversas métricas foram analisadas com o intuito de estimar o custo computacional de um *job*. O primeiro passo para a definição destas foi a decisão de que tipos de objetos e características analisar. Para tanto, é necessário conhecer um pouco mais sobre o funcionamento dos RIPs e sobre as características principais dos *jobs* das PSPs.

Como explicitado anteriormente (Seção 2.2), os RIPs são os responsáveis pela rasterização de um *job* PDF. Em suma, a função do RIP é a de gerar uma imagem *bitmap* correspondente ao PDF de entrada. O RIP executa esta função com base em páginas, o que significa que uma imagem é criada para cada página do PDF de entrada. Ao final de sua execução, um conjunto de imagens será obtido como resultado.

Para formar as imagens de saída, o RIP deve interpretar cada um dos objetos no PDF para pintá-los na imagem correspondente. Dentre estes objetos, aqueles sobre os quais será realizado o processo de conversão são os gráficos. Os gráficos mais comuns nos *jobs* são dois: os textos e as imagens. Em função destas características, **estes objetos gráficos**, juntamente com as **páginas**, foram os utilizados para compor as métricas para avaliar o custo de um PDF.

A seguir, são apresentados os experimentos que indicam a validade das métricas propostas. Estes resultados foram obtidos através da média de 20 execuções com o uso de um RIP *open-source*, o *ImageMagick converter* [30], e normalizados (para uma mesma escala entre 0 e 1), obtendo assim um **fator de relevância** para cada um destes. Nestes experimentos, os PDFs foram rasterizados utilizando-se 300 DPI (*Dots Per Inch*) de resolução. Através dos experimentos realizados será possível estabelecer uma maneira para calcular o custo computacional associado a cada uma das métricas avaliadas. É importante ressaltar que as métricas apresentam uma **aproximação** destes custos e não servem para prever valores exatos. Além disso, através da aplicação de tais métricas será possível ter uma idéia do custo de uma página / objeto.

5.1.1 Páginas

O número de páginas de um documento PDF está diretamente relacionado ao número de imagens geradas no final da rasterização. Desta forma, quanto maior for o número de páginas existentes, maior será a quantidade de operações de E/S (Entrada e Saída) que deverão ser realizadas pelos RIPs. Nesta seção é apresentado o custo de páginas em branco (sem quaisquer objetos gráficos) com o intuito de verificar o seu impacto no processo de rasterização como um todo. Para a realização dos experimentos a seguir, tomou-se como base diferentes tamanhos de páginas (com seus respectivos tamanhos denotados como *"largura x altura"* em *pixels*): *Postcard* (283 x 416), *Note* (540 x 720), A4 (595 x 842) e *Tabloid* (792 x 1224). A Figura 19 apresenta os fatores de relevância obtidos a partir da adição de mais páginas em um documento PDF.

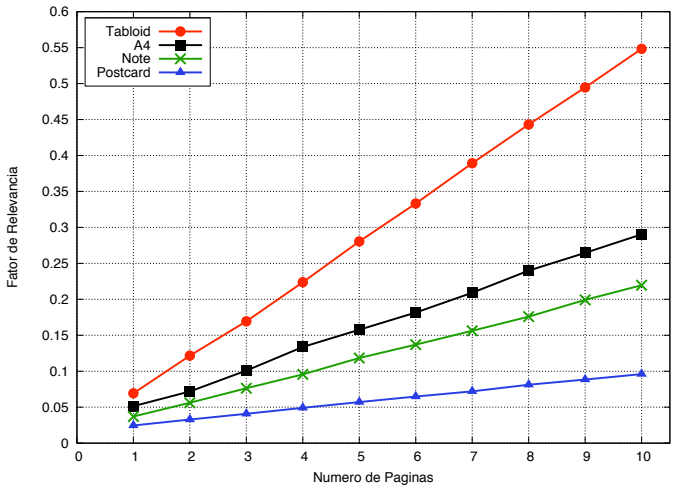

Figura 19 – Páginas em branco.

Com a finalidade de estabelecer o custo computacional para a rasterização de uma página, observou-se o incremento do fator de relevância existente à medida que mais páginas foram adicionadas. Neste sentido, diz-se que o custo de uma página será estabelecido como o incremento observado. Está claro que quanto maior as dimensões ou área da página em questão, maior é o incremento no fator de relevância. Esta situação está diretamente relacionada ao fato de que para páginas com áreas maiores, mais operações de E/S (Entrada e Saída) deverão ser realizadas, resultando assim em uma imagem final maior. Assim, definiu-se que o custo de uma página será denotado por $cPag_{tPag}$, onde $tPag$ representa as dimensões da página correspondente.

Os custos aproximados obtidos para os experimentos em questão foram: $cPag_{283x416} = 0.007$, $cPag_{540x720} = 0.019$, $cPag_{595x842} = 0.026$, $cPag_{792x1224} = 0.053$. Objetivando, então, estabelecer o custo $cPag_{tPag}$ para um $tPag$ qualquer, tomou-se como base o menor custo obtido nos experimentos: $cPag_{283x416}$. Neste sentido, diz-se que $area_{tPag}$ denota a área de um retângulo com as dimensões definidas por $tPag$. Com isso, se estabeleceu que o custo de uma

página de tamanho $tPag$ será obtido através da proporção de $\frac{area_{tPag}}{area_{283x416}}$, que resultará em um fator que será multiplicado pelo custo base. Esta estratégia é apresentada na Equação 5.1.

$$cPag_{tPag} = \frac{area_{tPag}}{area_{283x416}} * cPag_{283x416} \tag{5.1}$$

Devido à relação direta da área da página em questão com a grandeza do incremento, a equação apresentada é capaz de aproximar o fator de relevância. Assim, o custo de todas as páginas de um dado documento pode ser calculado através da soma do custo de cada página. Considerando os custos individuais de duas páginas, é possível perceber que o custo de ambas pode ser obtido através do custo da soma de seus tamanhos. Tal constatação pode ser generalizada, portanto, como demonstrado na Equação 5.2, diz-se que o custo de todas as páginas de um documento PDF é obtido através do custo da soma dos tamanhos das páginas (denotado como $tPagTot$).

$$cPag_{tPagTot} = \frac{area_{tPagTot}}{area_{283x416}} * cPag_{283x416} \tag{5.2}$$

5.1.2 Imagens

Imagens são objetos fundamentais na criação de *jobs* para clientes de uma PSP, sendo utilizadas para inúmeros fins, entre eles a apresentação de logotipos, propagandas, produtos, idéias, projetos, etc. Devido ao fato de que estas imagens são definidas através de uma matriz de pontos, uma grande preocupação por parte das PSPs é a de evitar a perda de qualidade, durante a impressão dos documentos. Assim, é comum a utilização de imagens com dimensões muito maiores do que o tamanho da página a qual estão inseridas, sendo redimensionadas para o tamanho desejado. Desta forma, mais pontos por polegada (*Dots Per Inch* - DPI) poderão ser aplicados para formar a imagem em questão. Este aspecto impossibilita o uso de *inline image objects* (Seção 3.2.2), devido às limitações apresentadas, que impedem a definição de imagens com um DPI desejável. Neste sentido, para definir imagens são aplicados somente os *image XObjects* (Seção 3.2.2).

Os experimentos apresentados a seguir levam em consideração um documento PDF contendo apenas uma página de tamanho A4 e imagens de tamanhos distintos, definidas através da utilização dos *image XObjects*. Os tamanhos de imagem selecionados foram quatro (descritos da seguinte forma *"largura x altura"* em *pixels*): *1190x1684*, *1785x2526*, *2380x3368* e *2975x4210*. O primeiro tamanho de imagem escolhido se refere ao dobro das dimensões de uma página A4 (de *595x842*), o segundo ao triplo e assim consecutivamente. Além disso, para compor estes experimentos, imagens distintas foram consideradas, como fotos (de alta resolução) e imagens artísticas (com degradês e efeitos gráficos), assim como o emprego da reusabilidade ou não para os objetos de imagens que as descrevem. Diz-se que uma imagem não re-utilizável é uma primeira instância (referência) a um dado *image XObject*, enquanto imagens

re-utilizáveis são aquelas instâncias subseqüentes as suas respectivas primeiras referências. Os resultados apresentados a seguir se referem a uma média dos resultados individuais de cada uma das imagens consideradas (fotos e imagens artísticas).

Sem Re-usabilidade

Para os experimentos apresentados nesta seção, cada objeto de imagem presente nos casos de teste é utilizado uma única vez, de forma a evitar a re-usabilidade.

O primeiro aspecto considerado na avaliação do custo computacional de uma imagem foi a sua cobertura (área de ocupação) na página. Isto se deve ao fato de que a mesma imagem deverá ser pintada no arquivo *bitmap* de saída, portanto este experimento avalia se quanto maior a imagem desenhada na página do documento PDF, maior será o esforço computacional aplicado para gerar o *bitmap* de saída. Para tanto, foram criados casos de teste sobre imagens, variando seu percentual de cobertura de 1% a 100%. A Figura 20 ilustra os resultados obtidos.

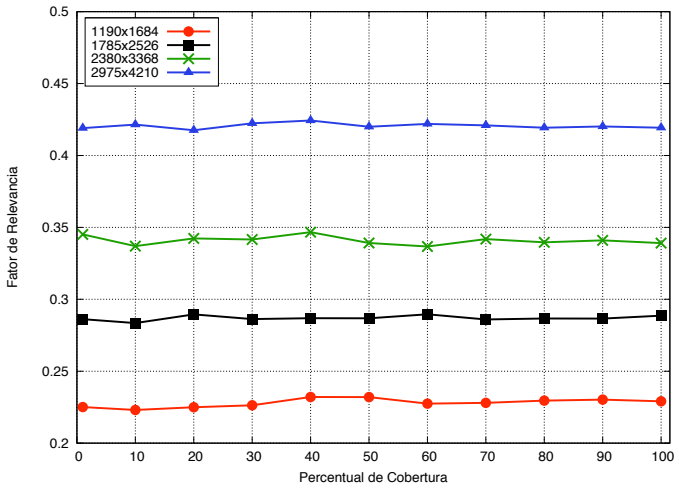

Figura 20 – Cobertura de imagens.

Analisando o gráfico apresentado, pode-se perceber que a cobertura da imagem não é um fator relevante para a rasterização do PDF e por outro lado, pode-se constatar que as dimensões das imagens são relevantes para tal processo. Isto ocorre devido ao fato de que o RIP deve interpretar o conteúdo da imagem, convertê-lo para o formato de saída selecionado e, então, redimensioná-lo para que então a imagem seja pintada. Com isso, quanto maior as dimensões da imagem em questão, mais processamento deverá ser realizado pelo RIP, impactando no tempo de rasterização. Neste sentido, para um mesmo tamanho de imagem, considerando-se o redimensionamento para 1% ou 100% da área da página, o RIP deverá interpretar e converter a mesma quantidade de conteúdo.

Para complementar o experimento supracitado, casos de teste foram gerados variando o número de objetos na página de 1 a 8. Desta forma, tornou-se possível descobrir o custo associado a um objeto de imagem à medida que mais objetos são adicionados na página. A seguir, a Figura 21 apresenta os resultados obtidos.

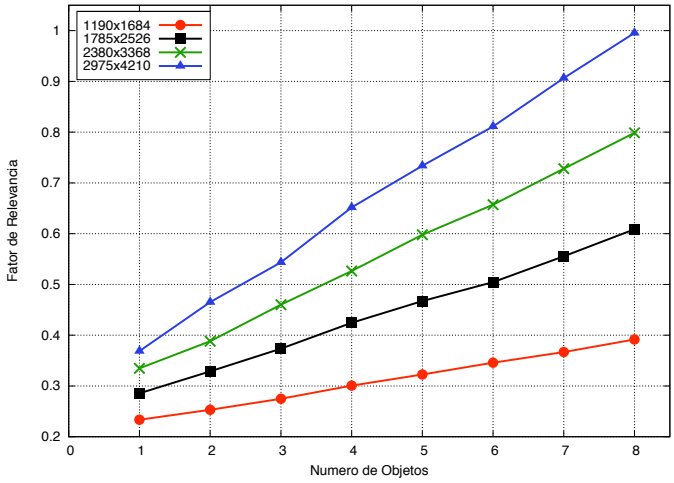

Figura 21 – Número de objetos de imagens.

O primeiro aspecto que pode ser analisado trata-se do fato de que quanto mais objetos de imagem, maior se torna o fator de relevância. Seguindo as constatações prévias, esta situação deveria ocorrer, pois isto representa mais conteúdo a ser processado pelo RIP. Outro fator que comprova esta afirmação está diretamente relacionado a que, quanto maior é a área do objeto inserido, maior é o incremento do fator de relevância.

A partir das constatações obtidas, deve-se quantificar o custo de um ou mais objetos de imagem. Neste sentido, diz-se que o custo de um objeto de imagem de tamanho tIm é representado como cIm_{tIm}. Para calcular o custo cIm_{tIm} dos objetos imagens, tomou-se como base $cIm_{1190x1684}$, que se trata do custo da menor imagem considerada nos experimentos.

Analisando o gráfico ilustrado na Figura 21, pode-se notar que $cIm_{1190x1684} = 0.019$. Observando o comportamento dos demais tamanhos de imagens, pode-se dizer que o incremento a cada objeto adicionado aumenta de acordo com o tamanho deste objeto, ou seja, quanto maior a área do objeto adicionado, maior será o incremento. Neste contexto, o custo de uma imagem com um tamanho tIm qualquer pode ser calculado da mesma forma que o custo de uma página, ou seja, através de uma proporção da área base com a área representada por tIm resultando em um fator que multiplicará o custo base. Seguindo este raciocínio, os seguintes custos são obtidos: $cIm_{1785x2526} = 0.042$, $cIm_{2380x3368} = 0.076$ e $cIm_{2975x4210} = 0.119$. Tais custos se aproximam da realidade observada nos experimentos e portanto o custo de uma imagem é definido como apresentado na Equação 5.3.

$$cIm_{tIm} = \frac{area_{tIm}}{area_{1190x1684}} * cIm_{1190x1684} \tag{5.3}$$

Definido o custo de um objeto de imagem, o custo de todos os objetos de imagem de um documento PDF pode ser calculado a partir da soma dos custos individuais de cada objeto. Por outro lado, está claro que o tamanho de dois objetos é igual a soma dos respectivos tamanhos e, portanto, o custo destes objetos pode ser obtido através do custo da soma de seus tamanhos. A partir desta constatação o custo total poderá ser obtido, então, através do custo da soma dos

tamanhos tIm dos objetos (representado aqui como $tImTot$). Esta equação é apresentada a seguir (Equação 5.4).

$$cIm_{tImTot} = \frac{area_{tImTot}}{area_{1190x1684}} * cIm_{1190x1684} \tag{5.4}$$

Partindo dos custos de objetos sem re-usabilidade então, foram realizados experimentos com a aplicação da transparência sobre estes, para verificar a importância desta característica. A Figura 22 apresenta os resultados obtidos, considerando a variação de transparência sobre tais objetos.

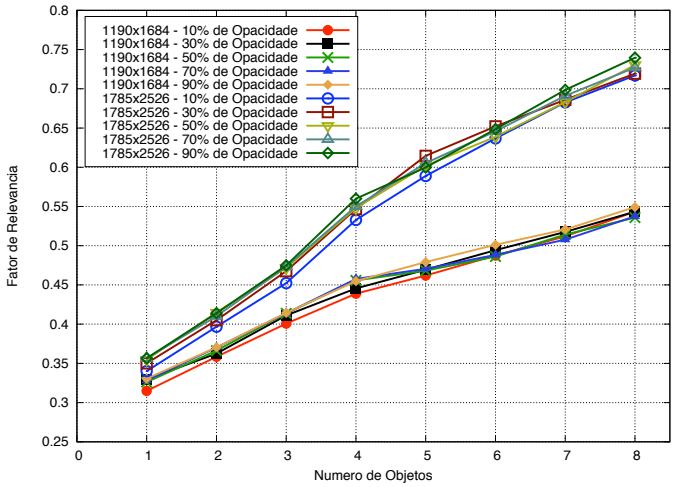

Figura 22 – Objetos de imagem não-reutilizáveis transparentes. Nesta figura as curvas mais acima representam o tamanho de imagem de *1785x2526*, enquanto as curvas inferiores demonstram o comportamento das imagens com tamanho *1190x1684*.

Como pode-se notar, devido à aplicação de transparência existe um grande crescimento no fator de relevância. Isto ocorre devido à necessidade do RIP de compor as cores dos objetos transparentes baseado na sobreposição das cores destes com as cores das páginas e objetos sobre os quais estão posicionados. Além disso, no gráfico apresentado pode-se notar que quanto maior a área dos objetos transparentes em questão, maior será é o incremento no fator de relevância associado. Tal fato decorre de que quanto maior a área do objeto transparente em questão, mais cores deverão ser computadas. Então, para descobrir a grandeza do custo de um objeto transparente, definido como $cImT_{tImT}$, pode-se adotar a estratégia de análise utilizada anteriormente: observar o incremento no fator de relevância.

Analisando o custo médio para os objetos com o menor tamanho dos experimentos pode-se concluir que $cImT_{1190x1684} = 0.026$. Sabendo que o custo de um objeto transparente está relacionado diretamente a sua área, pode-se facilmente verificar que $cImT_{1785x2526}$ pode ser calculado através da multiplicação de um fator pelo custo do menor objeto. Assim, da mesma forma que para objetos sem transparência, este fator pode ser calculado como $\frac{area_{tImT}}{area_{1190x1684}}$. De forma análoga ao custo de todos os objetos não re-utilizáveis opacos, o custo destas imagens transparentes podem ser calculadas através da aplicação do tamanho de todos estes objetos

($tImTTot$) na Equação 5.5.

$$cImT_{tImTTot} = \frac{area_{tImTTot}}{area_{1190x1684}} * cImT_{1190x1684} \tag{5.5}$$

Com Re-usabilidade

Os casos de teste apresentados nesta seção, possuem a definição de uma única imagem no contexto do documento PDF e esta é referenciada quantas vezes necessário. Assim, desejou-se avaliar o impacto da re-usabilidade no custo computacional presente na rasterização de imagens. Para tanto, casos de teste similares aos apresentados na Seção 5.1.2 foram criados, porém as imagens presentes nos documentos representam referências a uma instância de *image XObject* definida uma única vez. Por exemplo, na situação em que o caso de teste contenha 8 objetos, 7 destes serão apenas referências ao primeiro. Com isso, não são apresentados experimentos com um único objeto de imagem, pois nesta situação não há o emprego da re-usabilidade. Os resultados obtidos são apresentados na Figura 23.

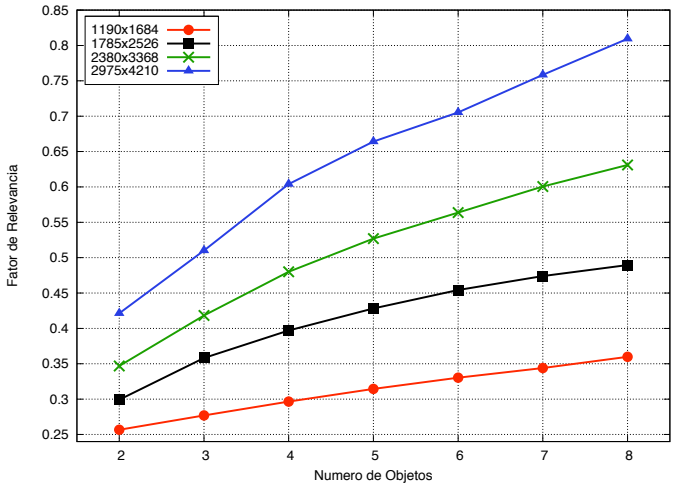

Figura 23 – Objetos de imagem re-utilizáveis.

Como se pode perceber, os resultados apresentados seguem o mesmo comportamento daqueles sem a re-usabilidade, entretanto o custo adicionado à medida que novos objetos são inseridos é menor. Cabe ressaltar que existe um custo ao adicionar mais objetos, mesmo que re-utilizáveis, pois os RIPs têm de buscar o resultado da rasterização dos mesmos em sua *cache*, aplicando-o na imagem de saída. Tal computação provoca a adição de um custo ao processamento total. Neste contexto, diz-se que o custo de um objeto reutilizável será denotado através de cRe_{tRe}. Tomando como base o custo do menor tamanho de imagem considerado, tem-se que $cRe_{1190x1684} = 0.011$. Os demais custos de objetos para os tamanhos testados são aproximadamente: $cRe_{1785x2526} = 0.030$, $cRe_{2380x3368} = 0.042$ e $cRe_{2975x4210} = 0.065$. Neste sentido, pode-se perceber que novamente existe uma relação entre a área do objeto adicionado com o seu custo associado. Mais uma vez, esta relação pode ser aproximada através da divisão entre a

área base, $area_{1190x1684}$, pela área do objeto de tamanho tRe que se está analisando ($area_{tRe}$). Entretanto, pode-se afirmar que este crescimento pode ser aplicado no contexto da existência de re-usabilidade, portanto se está considerando que tRe diz respeito ao tamanho de um objeto re-utilizado, excluindo, assim, a primeira instância do mesmo. Como visto anteriormente, o custo total dos objetos pode ser calculado na forma do custo da soma dos tamanhos. Neste sentido, pode ser estabelecido que $tReTot$ refere-se ao tamanho total dos objetos re-utilizados. A Equação 5.6 demonstra o custo para estes objetos.

$$cRe_{tReTot} = \frac{area_{tReTot}}{area_{1190x1684}} * cRe_{1190x1684} \tag{5.6}$$

Da mesma forma que para os objetos sem re-usabilidade, alguns experimentos foram realizados considerando a aplicação de transparência para os objetos re-utilizáveis. A Figura 24 ilustra os resultados obtidos.

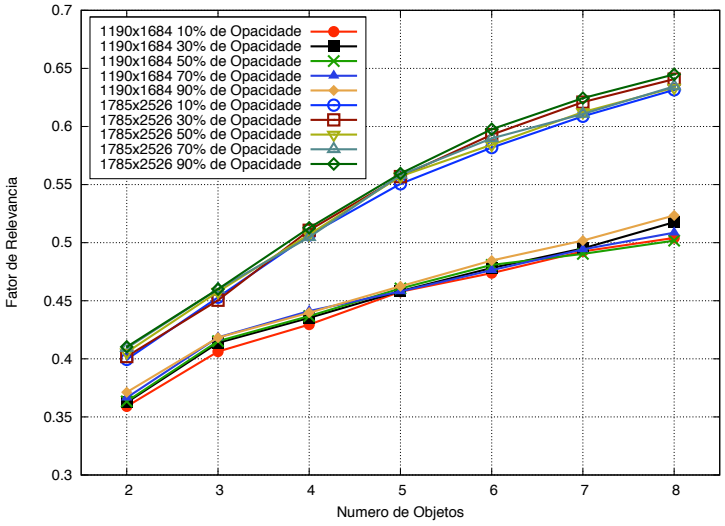

Figura 24 – Objetos de imagem re-utilizáveis transparentes. Nesta figura as curvas mais acima representam o tamanho de imagem de *1785x2526*, enquanto as curvas inferiores demonstram o comportamento das imagens com tamanho *1190x1684*.

O gráfico apresentado demonstra que para objetos re-utilizáveis transparentes, o mesmo aumento no fator de relevância pode ser observado. Considerando-se os tamanhos de imagem apresentados, os seguintes custos aproximados foram obtidos: $cReT_{1190x1684} = 0.019$ e $cReT_{1785x2526} = 0.042$. Tomando como base o menor custo, mais uma vez o outro custo pode ser aproximado através da proporção entre suas respectivas área, multiplicando-se o resultado pelo custo base. Portanto, o custo total de objetos de imagens re-utilizáveis transparentes pode ser computado pelo custo da soma de seus tamanhos ($tReTTot$), como apresentado na Equação 5.7.

$$cReT_{tReTTot} = \frac{area_{tReTTot}}{area_{1190x1684}} * cReT_{1190x1684} \tag{5.7}$$

5.1.3 Textos

Os textos são normalmente aqueles sobre os quais a personalização dos documentos é aplicada. Com isso, nestes elementos estarão contidas as mensagens direcionadas para cada recipiente das PSPs. No formato PDF, estes elementos são representados pelos *text objects*, que foram instanciados para a criação dos experimentos desta seção. A variabilidade considerada para estes objetos é representada pela quantidade de texto, tamanho de fonte e aplicação de transparência. Além disso, todos os testes apresentados foram criados com tamanho(s) de página(s) A4 e com 70 fontes distintas, com a aplicação de variações de cada uma delas, como negrito, itálico ou ambos. Neste sentido, foram gerados 188 casos de teste para cada característica analisada. Para fins de ilustração e visualização, 10 fontes foram selecionadas, que representam o comportamento observado em todos os casos de teste, mas uma listagem de todas as fontes empregadas estão disponíveis no Apêndice A.

O primeiro aspecto considerado foi impacto no custo de rasterização ao aumentar a quantidade de texto em um documento PDF. Para tanto, define-se que um objeto de texto corresponde a uma frase no documento. A Figura 25 apresenta os resultados obtidos.

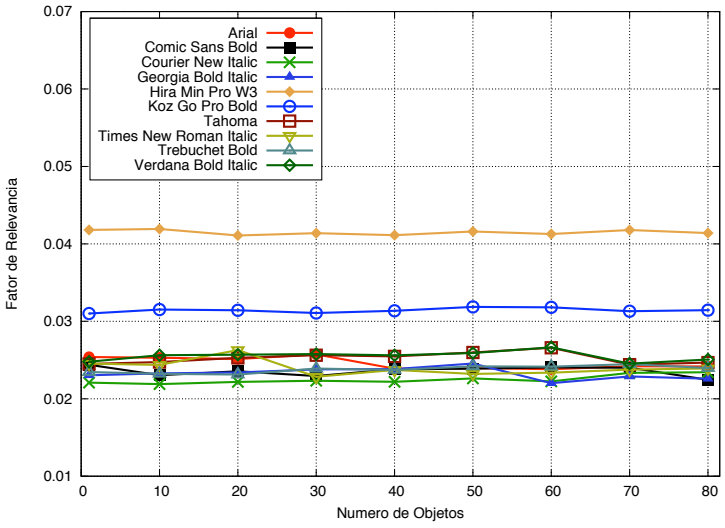

Figura 25 – Quantidade de texto.

Pode-se notar que existe uma flutuação nos valores obtidos à medida que mais objetos são adicionados. Entretanto, estes resultados não apresentam nenhuma tendência de crescimento ou decréscimo. Os mesmo valores foram obtidos quando variou-se o tamanho das fontes empregadas (de 10 a 100 pontos). Portanto, pode-se dizer que a quantidade de texto e o tamanho da fonte não é impactante no custo da rasterização do texto. Por outro lado, a presença de texto por si só influencia no tempo de rasterização, pois se comparado ao testes de páginas em branco (Seção 5.1.1), é possível perceber que o fator de relevância obtido é maior. Além disso, conforme a fonte aplicada, nota-se diferentes influências. Analisando o gráfico apresentado, pode ser visto que destacam-se três grupos distintos em termos de custo computacional. Neste sentido, define-

se que estes grupos serão referenciados, da sua maior influência para a menor respectivamente, como: **Pesado**, **Médio** e **Leve**. No primeiro grupo está presente a maior influência no fator de relevância (como a fonte *Hira Min Pro W3*), representando 5% das 188 variações de fontes consideradas. O segundo grupo está relacionado às fontes que possuem uma influência média no fator de relevância (como a fonte *Koz Go Pro Bold*) e denotam 9% das variações testadas. Finalmente, o terceiro grupo representa as fontes que apresentam a menor influência no fator de relevância (como as 8 fontes restantes apresentadas no gráfico acima). Considerando-se as variações testadas, este grupo representa o comportamento de 86% delas.

A diferença na influência entre o tipo de fontes com o fator de relevância ocorre devido a peculiaridades de cada fonte, como o seu próprio *design* e a quantidade de detalhes, que afetam o custo de processamento por parte dos RIPs. Entretanto, não é o objetivo deste trabalho a realização de uma análise da complexidade das fontes, que pode ser muito custoso em termos de esforço de processamento e poderia comprometer ou invalidar a aplicação da métrica em PDFs que contenham diversas variações de fontes. Portanto, será assumido que a presença de texto representa um custo constante para cada página, já que o número de objetos de texto não influenciam nesta questão. Este custo será calculado como uma média ponderada dos custos individuais de cada um dos três grupos apresentados. Assim, diz-se que um custo $cTxt = txt$ deverá ser computado caso exista a presença de na página do documento PDF. Com o intuito de encontrar o valor aproximado de txt no contexto dos diferentes grupos, experimentos usando um número fixo de *text objects* e um número variado de páginas (de 1 a 5) foram gerados. Desta maneira, tornou-se possível avaliar os custos obtidos à medida que mais páginas com texto foram adicionadas. Estes resultados são ilustrados na Figura 26.

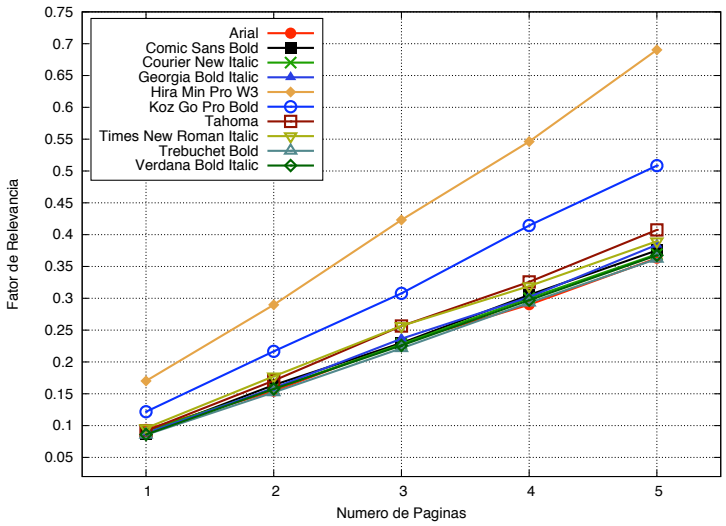

Figura 26 – Textos e páginas.

Descartando o custo das páginas para os casos apresentados e analisando o custo restante, pode-se notar que este aumenta à medida que mais páginas com texto estão envolvidas. Assim, é possível dizer que o texto realmente influencia no processo de rasterização de um PDF. Por-

tanto, através da divisão do custo restante (dos textos) pelo número de páginas envolvidas no experimento, os seguintes valores foram obtidos:

- Grupo **leve**: 0.042;
- Grupo **médio**: 0.061;
- Grupo **pesado**: 0.088.

Utilizando estas constatações, é possível aplicar uma média ponderada sobre tais resultados, obtendo um único custo constante para a presença de texto: $cTxt = 0.046$. Com isso, diz-se que para np páginas com texto, o custo total das porções de texto em um documento PDF (denotado como $cTxtTot$) é calculado como apresentado na Equação 5.8.

$$cTxtTot = \sum_{i=1}^{np} txt \tag{5.8}$$

Outro aspecto analisado sobre os objetos de texto é a presença de transparência e seu impacto no custo de rasterização. Com o intuito de verificar o impacto deste recurso, testes foram realizados com diferentes valores de opacidade (cuja uma média é apresentada a seguir) e um número fixo de objetos (80 objetos de texto). Os resultados obtidos são apresentados na Figura 27.

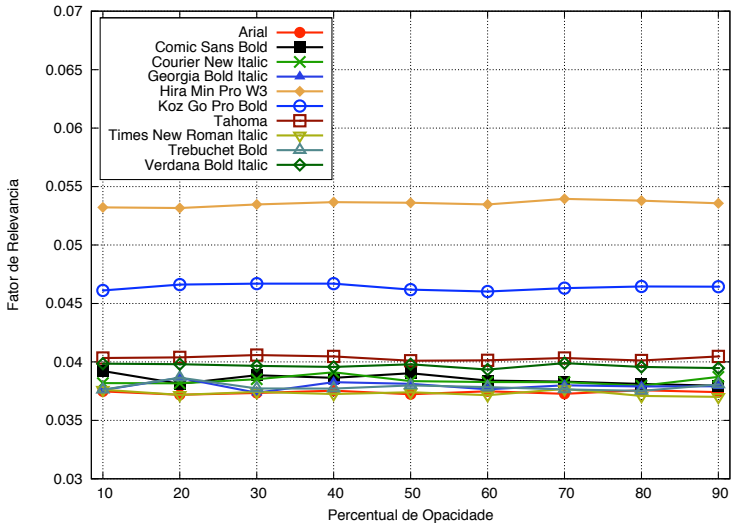

Figura 27 – Transparência de texto.

Da mesma maneira do que para os objetos de imagem, alguma flutuação foi encontrada, mas nada que indicasse uma relação entre o fator de relevância e o valor de opacidade empregado. Por outro lado, como pode ser visto no gráfico apresentado, o fator de relevância aumentou devido ao fato dos objetos de texto serem transparentes. Para verificar o valor dos textos transparentes, desta vez variou-se o número de páginas com texto transparente, analisando o custo à medida que mais páginas foram adicionadas. Estes resultados são demonstrados na Figura 28.

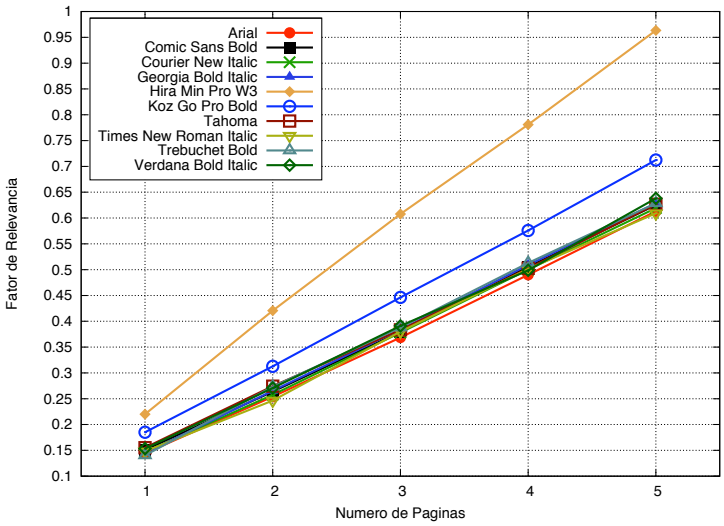

Figura 28 – Transparência de texto e páginas.

Para os três grupos de texto estabelecidos, os seguintes custos computacionais foram obtidos:

- Grupo **leve**: 0.092;
- Grupo **médio**: 0.127;
- Grupo **pesado**: 0.154.

Comparando estes resultados com aqueles dos textos sem transparência, pode-se notar que o efeito de transparência provoca um aumento de aproximadamente $incTxt = 2$ vezes no fator de relevância. Desta forma, a fórmula para calcular o custo dos textos com e sem transparência pode ser unificada em uma única, como apresentado na Equação 5.9.

$$cTxtTot = \sum_{i=1}^{np} txt + (tr_i * (incTxt - 1) * txt) \tag{5.9}$$

Na equação apresentada, a variável tr_i representa a existência ou não de transparência nos textos na página i. Assim, pode-se afirmar que tr_i pode assumir os valores 0 ou 1, provocando ou não um aumento no custo dos textos. Neste cenário, o valor 0 é assumido caso não exista transparência na página i e, caso contrário, o valor 1 é assumido.

5.1.4 Custo Computacional Total

Até o momento foram definidos os custos analisando os objetos e características individualmente. Entretanto, deseja-se saber o custo computacional de um documento PDF, fornecendo assim uma estimativa de qual o esforço que será realizado para rasterizá-lo. Para tanto, diz-se que o custo computacional de um documento trata-se da soma dos custos individuais das

características analisadas. Desta maneira, a Equação 5.10 apresenta a fórmula que deverá ser utilizada para obter o custo de um documento, denotado como $cDoc$.

$$cDoc = cPag_{tPagTot} + cIm_{tImTot} + cImT_{tImTTot} + cRe_{tReTot} + cReT_{tReTTot} + cTxtTot \quad (5.10)$$

5.2 PDF Profiler

A ferramenta *PDF Profiler* foi desenvolvida com o intuito de analisar um documento PDF, fornecendo informações sobre o documento em si. Esta ferramenta foi implementada com a utilização da biblioteca Java PDFBox [31], que provê métodos para navegar e encontrar os elementos necessários no documento PDF. Para a execução do *PDF Profiler* deve ser fornecido um documento XML que define as informações que serão analisadas, juntamente com o documento PDF do qual deseja-se extrair as informações. O funcionamento geral da ferramenta é composto por três passos principais (Figura 29): interpreta-se o arquivo XML de entrada, configurando-se quais são os dados desejados; buscam-se as informações requisitadas no documento PDF recebido como entrada, realizando a interpretação dos objetos necessários; gera-se um documento XML de saída com os resultados obtidos. Entre os possíveis resultados fornecidos pela ferramenta estão aqueles necessários para a aplicação das métricas. Entretanto, o *PDF Profiler* não se limita a apenas tal conhecimento, sendo capaz de prover dois conjuntos de dados importantes, que serão utilizados futuramente (para mais detalhes ver Capítulo 7). É importante ressaltar que a ferramenta *PDF Profiler* suporta a análise de perfil de um documento inteiro e de vários fragmentos deste separadamente. Na última situação, um único *job* será recebido como entrada, mas cada fragmento deste será analisado como se fosse um documento distinto. Esta funcionalidade será interessante para as estratégias de escalonamento que serão aplicadas (Seção 5.4). Assim, as informações extraídas pelo *PDF Profiler* são:

1. **Páginas**: área total das mesmas;
2. **Textos**: presença destes elementos em quantas páginas e presença de transparência para estes em quantas páginas.
3. **Imagens**: área total ocupada com e sem re-usabilidade, assim como área total de imagens com transparência, com e sem re-usabilidade.
4. **Escopo de *image XObjects***: para cada *image external object*, quais são as páginas em que eles são aplicados.
5. **Transparência de páginas**: quais páginas contêm elementos transparentes e quais contêm apenas objetos opacos.

Considerando-se as informações especificadas acima, pode-se afirmar que o processamento do *PDF Profiler* é focado na interpretação do *content stream* de uma página, analisando os objetos de texto, imagem e o estado dos gráficos.

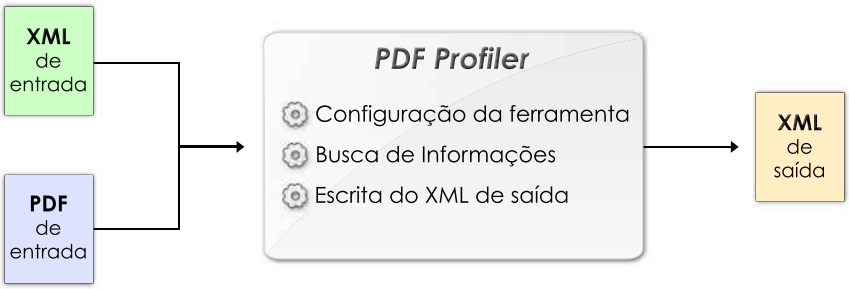

Figura 29 – Funcionamento geral da ferramenta *PDF Profiler*.

Pode-se perceber que a principal função da ferramenta é a busca e interpretação dos objetos do PDF. Assim, com o intuito de descrever tal processamento é necessário expor algumas estruturas que são utilizadas para a obtenção das informações de interesse:

- *estadoGraficosAtual*: o estado dos gráficos corrente. Nesta estrutura estará apenas a informação se deve-se utilizar transparência ou não;
- *pilhaEstadoGraficos*: a pilha para o controle dos estados dos gráficos;
- *mapaImagens*: mapa para o controle das imagens, possibilitando a verificação se estas estão sendo re-utilizadas ou não. Este mapa é acessado através do identificador único da imagem retornando um objeto que conterá informações sobre tal objeto. Assim, nestas informações estarão os escopos de cada imagem.

A ferramenta *PDF Profiler* inicia o processamento do PDF com a interpretação (*parsing*) do documento, utilizando a biblioteca PDFBox. São disponibilizados, assim, elementos de alto nível de abstração permitindo a navegação nos objetos do documento de maneira mais intuitiva. Com isso, uma estrutura hierárquica é obtida, onde no topo desta hierarquia está o *document catalog* do PDF. Esta estrutura fornecerá acesso a todas as páginas e atributos das mesmas, assim como a seus respectivos *content streams* e *resources*.

Realizada a interpretação do documento por completo, com o uso da estrutura obtida como resultado, são recuperadas todas as páginas do documento, que serão analisadas uma a uma em um laço. Desta forma, uma iteração do laço em questão corresponderá a análise de uma página e de seus objetos correspondentes. Ao início de uma iteração, o estado dos gráficos atual é inicializado, estabelecendo-se a ausência de transparência. A seguir, a altura e largura da página são recuperadas e multiplicadas para a obtenção da área desta, acumulando-se o resultado. Com isso, parte-se para a recuperação dos atributos de interesse para os objetos de texto e imagem da página. Para tanto, se torna necessário interpretar cada operação realizada no conteúdo da

página que influencie sobre os objetos de interesse. Neste sentido, apenas os operadores que dizem respeito à pintura de texto, imagens e grupos nas páginas, assim como ao estabelecimento de transparência sobre o estado dos gráficos, além do salvamento e da restauração destes elementos, são de interesse para a análise do *PDF Profiler*. Como exposto anteriormente (Seção 3.2.2), estes operadores tratam-se de: **gs**, **q**, **Q**, **Tj** e **Do**.

Para o estado dos gráficos os operadores **gs**, **q** e **Q** são analisados. No momento em que se encontre o primeiro destes operadores, uma análise sobre os seus respectivos operandos é realizada, para verificar se está se aplicando transparência. Em caso afirmativo, a variável *estadoGraficosAtual* é marcada com o valor de utilização de transparência, senão a ela é atribuído o valor de ausência de transparência. Nota-se, como exposto anteriormente, que esta variável apenas armazenará informação quanto a transparência, já que é o único parâmetro do estado dos gráficos o qual deseja-se conhecer. Os demais operadores do estado dos gráficos, provocarão alterações na estrutura *pilhaEstadoGraficos*. Assim, à medida que o operador **q** for encontrado, o valor de transparência do *estadoGraficosAtual* será colocado no topo da pilha simulando uma operação de salvamento. Quando o operador **Q** estiver presente, a variável *estadoGraficosAtual* receberá o valor daquele elemento no topo da estrutura *pilhaEstadoGraficos*, sendo que este elemento será removido da mesma.

Considerando os objetos de texto avalia-se apenas o operador **Tj**. Assim que tal operador for encontrado, sabe-se que deseja-se pintar um texto na página. Desta forma, verifica-se o estado dos gráficos atual avaliando a existência de transparência. Caso esteja sendo empregado a transparência, marca-se que a página corrente contém texto transparente e diz-se que não há a necessidade de analisar mais a presença de textos para tal página. Isto decorre do fato de que a quantidade de texto não é interessante para as métricas, portanto apenas é desejado saber se a página possui texto e se este texto é transparente ou não. No cenário em que não se esteja aplicando transparência, verifica-se se a página ainda deve ser analisada. Em caso afirmativo, marca-se a página atual com a presença de texto opaco. Caso contrário, nenhuma ação é tomada. Ao final de cada iteração, então, será verificado se a página foi marcada com presença de texto ou texto transparente, somando um ao respectivo contador (de páginas com texto ou com texto transparente).

O operador **Do** se refere a pintura de *XObjects* na página. Neste sentido, deseja-se apenas considerar os objetos deste tipo que sejam imagens ou grupos (para verificar os objetos nele contidos). Para tanto, assim que um operador **Do** for encontrado, através do operando que trata-se nome do objeto, procura-se a referência para o mesmo no dicionário *resources* da página. Desta maneira, obtém-se o identificador único do *XObject*, possibilitando a procura da definição de tal objeto no documento PDF, através de uma consulta a *cross-reference table*. Com a definição do *XObject* em mãos, verifica-se seu tipo (através da entrada do dicionário **Subtype**), para estabelecer se é uma imagem ou um grupo. Caso o objeto seja um grupo, recupera-se o seu conteúdo, tratando-o como se fosse o conteúdo de uma página. Isto quer dizer que analisa-se todos operadores para o conteúdo do grupo, tomando as mesmas ações definidas para a avali-

ação do conteúdo de uma página. Tal atitude é tomada, pois como mencionado anteriormente um *group XObject* trata-se de uma unidade auto-contida, onde os objetos pertinentes a ele estão definidos no seu próprio *stream*. Por outro lado, caso o objeto seja uma imagem, procede-se para o processamento de seus atributos.

O processamento de uma imagem consiste em três passos: na verificação se está imagem é uma primeira instância, ou trata-se da re-utilização da mesma, atualizando o mapa de imagem; na avaliação caso aplicou-se transparência e, por fim, na obtenção da área da própria imagem, acumulando-se o resultado. Com isso, primeiramente a ferramenta *PDF Profiler* realiza uma consulta na estrutura *mapaImagens* com o identificador único da imagem. Neste contexto, diz-se que se o mapa não conter o identificador pesquisado, a imagem trata-se de uma primeira instância. Assim, insere-se o identificador único da imagem no mapa, juntamente com a informação da página em que está sendo utilizada (página atual). Por outro lado, se o mapa de imagens já conter o identificador da imagem em questão, ela trata-se de uma re-utilização. Desta forma, apenas atualiza-se a informação do objeto de imagem do mapa, adicionando a página atual, se esta já não estiver presente em tal informação. A seguir, a ferramenta verifica no estado dos gráficos atual a existência de transparência, reconhecendo se a imagem será transparente ou não. Finalmente, a ferramenta consulta o dicionário da imagem em questão, recuperando sua altura e largura, que são multiplicadas para adquirir a sua área. Com as informações obtidas nos três passos descritos, então, soma-se a área da imagem no acumulador correspondente, aquele de imagens não re-utilizáveis opacas, não re-reutilizáveis transparentes, re-utilizáveis opacas ou re-utilizáveis transparentes.

Através da análise descrita acima para os operadores de interesse, ao final da análise de todas as páginas obter-se-ão as informações requisitadas. Cabe ressaltar que no arquivo XML de configuração define-se quais são as informações de saída desejadas, evitando-se, assim o processamento por parte do *PDF Profiler* de análises irrelevantes para o usuário final. Assim, a quantidade de informações requisitadas no arquivo de configuração terá um impacto direto no tempo processamento da ferramenta.

5.3 PDF Splitter

A ferramenta *PDF Splitter* foi desenvolvida com o intuito de quebrar um documento PDF em diversos fragmentos. Como mencionado anteriormente, o PDF se trata de um formato baseado em referências, o que significa que um objeto qualquer (não somente objetos gráficos) utilizado em uma página, pode ser definido no contexto de outra página. Portanto, o *PDF Splitter* preocupa-se em resolver referências para cada fragmento do PDF em questão. Com esta finalidade, para cada página do fragmento gerado, a ferramenta irá procurar pela definição de objetos no documento PDF original que não se encontrem no contexto da própria página e das demais páginas do fragmento. Para tanto, utiliza-se uma funcionalidade fornecida pela biblio-

teca PDFBox, que é capaz de resolver todas as referências de uma página que estão faltando para um documento em questão.

Em suma, a funcionalidade do *PDF Splitter* é a criação de um novo documento PDF, para cada fragmento especificado, com suas páginas correspondentes. Neste sentido, a ferramenta gera um novo arquivo para cada um dos fragmentos e insere as páginas correspondentes em cada um destes. Esta inserção, consiste na cópia dos objetos de cada página para o seu arquivo, **caso estes objetos já não existiram no documento em questão**. Assim, alguns objetos terão de ser re-definidos para cada documento novo, podendo-se perder parte da vantagem fornecida pelo recurso da re-usabilidade. A Figura 30 exemplifica esta situação.

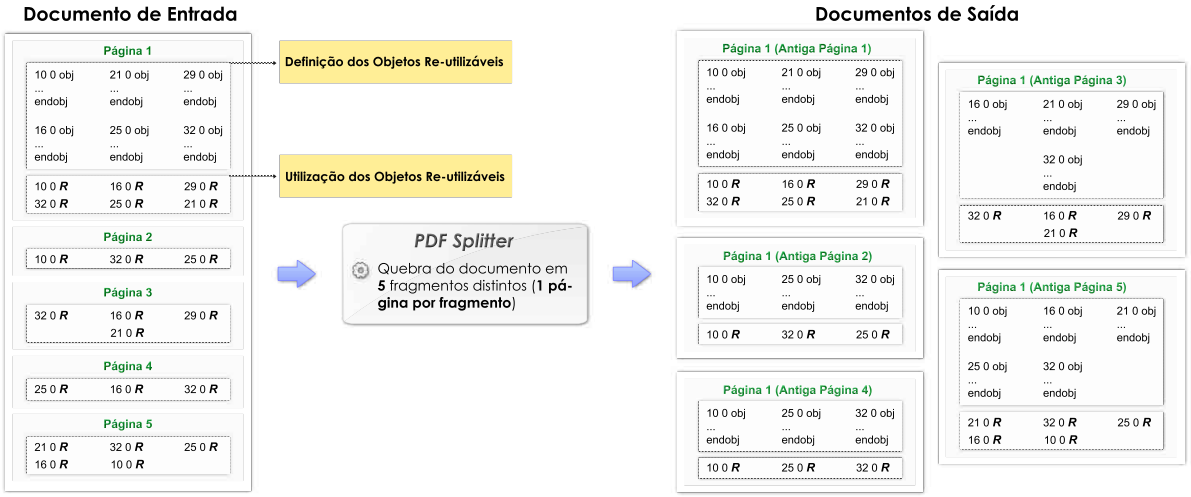

Figura 30 – Quebra do recurso da re-usabilidade.

Os modos de quebra suportados pela ferramenta tratam-se da quebra por intervalos e da quebra através de páginas específicas. Através da utilização do primeiro, é possível obter fragmentos que representem uma quebra do documento original a cada p páginas, onde p representa o intervalo desejado. O segundo trata-se de um modo de quebra mais flexível, onde é possível especificar exatamente as páginas desejadas do documento original que estejam contidas em cada um dos fragmentos. Esta segunda situação é interessante para aproveitar melhor a estrutura do documento, como por exemplo, a re-usabilidade. Nesta situação, poderia-se agrupar as páginas que contêm os mesmos objetos re-utilizáveis, mantendo assim o recurso de re-usabilidade entre os diversos fragmentos. Além disso, através dos modos de quebra é possível ignorar certas páginas do documento PDF, ou seja, pode-se obter fragmentos do documento PDF original que não necessariamente contemplem o documento inteiro. Esta característica otimiza o desempenho do processo de quebra, pois não será necessário analisar todas as páginas do documento original para obter-se os fragmentos desejados.

5.4 Escalonadores

Como mencionado anteriormente, as estratégias existentes aplicam apenas o algoritmo LS para ganhar desempenho no âmbito da rasterização dos *jobs* da fila. Neste sentido, dois algoritmos de escalonamento foram aplicados, objetivando a obtenção de um desempenho melhor: LPT e *Multifit*. Além destes, propôs-se um algoritmo, chamado aqui de LPT Otimizado, onde realizou-se uma pequena modificação sobre o algoritmo LPT para contornar um problema que poderia ocorrer no escalonamento dos *jobs*. Neste contexto, estes três algoritmos foram aplicados tomando como base a estratégia existente que possui o menor número de desvantagens: **todos os RIPs por *job***. Esta estratégia propõe a quebra dos *jobs*, obtendo assim um grão mais fino se comparado a situação em que *jobs* inteiros são escalonados para cada máquina. Desta forma, evita-se uma maior sobrecarga e sub-carga dos processos. Além disso, esta estratégia é claramente mais eficiente do que o caso em que são utilizados grupos de RIPs, pois como explicitado anteriormente diversos RIPs podem ficar ociosos enquanto outros do mesmo grupo não finalizaram seu trabalho. Cabe ressaltar que as estratégias propostas irão utilizar a mesma maneira para a leitura dos *jobs* de entrada, conforme as estratégias existentes: os *jobs* serão disponibilizados em um diretório visível por todos os RIPs, e serão lidos por estes à medida que necessário. Desta maneira, será possível avaliar apenas a forma como será realizada a distribuição das tarefas, além de se manter compatível com a maneira de disponibilização de novos *jobs* para os RIPs, conforme já aplicado no cenário das PSPs.

Em um primeiro momento, será apresentada uma abordagem genérica adotada para o funcionamento dos escalonadores desenvolvidos. A seguir serão descritas as peculiaridades de cada algoritmo de escalonamento empregado em tal abordagem. Neste contexto, as ferramentas *PDF Profiler* e *PDF Splitter*, em conjunto com as métricas estipuladas, foram utilizadas para a construção das estratégias propostas. Desta maneira, considerando-se a existência de m unidades de processamento disponíveis, o algoritmo de escalonamento A e a fila de *jobs* ψ, contendo n *jobs* iniciais, tal que $\psi = \{job_1, job_2, ..., job_n\}$, os passos realizados pelos escalonadores são:

1. Para cada job_i da fila ψ, onde $1 \leq i \leq n$, é executado o passo 2.

2. Considerando-se um job_j com um número total de páginas tp_j, são estabelecidos m limites de fragmentos (página inicial e páginal final) caso tp_j seja maior que m, denotados por $lim_{j,l}$. Nesta situação, cada um destes conterá $\frac{tp_j}{m}$ páginas. Por outro lado, caso tp_j seja menor que m, são gerados apenas tp_j limites, cada um com uma página.

3. Os limites gerados são inseridos seqüencialmente na fila de tarefas $\delta = \{T_1, T_2, ..., T_k\}$, onde cada tarefa T_l, para $1 \leq l \leq k$, corresponde a um limite gerado $lim_{j,l}$.

4. Através do emprego do *PDF Profiler* são obtidas informações para as novas tarefas inseridas na fila δ.

5. A métrica é aplicada sobre as informações obtidas, resultando no custo $cDoc_l$ associado a cada tarefa T_l analisada.

6. São aplicadas as diretivas de escalonamento definidas em A para organizar as tarefas da fila δ.

7. As tarefas são transmitidas para cada máquina ociosa de acordo com o definido em A.

8. A cada novo *job* obtido pelo escalonador uma regra é aplicada conforme o algoritmo A empregado.

9. Toda vez que uma máquina ficar ociosa é executado novamente o passo 7.

Dentre os passos estabelecidos, é importante ressaltar que cada tarefa corresponde ao limite de páginas de cada fragmento, ou seja, a partição dos *jobs* **não** é realizada pelo escalonador. Estes limites são considerados como tarefas, pois serão transmitidos a cada máquina que, então, fará a partição do job_j associada ao limite $lim_{j,l}$ para cada tarefa T_l recebida. Desta maneira, o escalonador não ficará sobrecarregado com a computação de cada fragmento, pois tal processamento será delegado para os RIPs disponíveis que realizarão tal tarefa em paralelo. Assim, para a aplicação desta estratégia, se está considerando um ambiente onde todas as máquinas tenham acesso aos *jobs* da fila. Além disso, vale a pena destacar que as informações de cada tarefa T_l são obtidas através da execução do *PDF Profiler* sobre o job_j associado ao limite $lim_{j,l}$. O resultado da execução do *PDF Profiler* sobre um job_j será informações sobre todos os limites (fragmentos) associados a este de uma única vez.

Considerando a abordagem genérica apresentada acima, cada algoritmo irá realizar diferentes ações, consistentes com o funcionamento do próprio algoritmo. A seguir, serão estabelecidas as ações realizadas sobre a abordagem genérica para o cenário dos três algoritmos de escalonamento descritos.

LPT

A base do algoritmo LPT está centrada na execução de tarefas mais pesadas antes daquelas mais leves. Assim, com a utilização deste algoritmo, uma ordenação é realizada sobre o conjunto de tarefas δ (passo 6), tal que estas tarefas estejam organizadas de forma decrescente, conforme $cDoc_l$, para cada tarefa T_l da fila. Desta forma, as tarefas com o custo computacional maior, conforme as métricas estipuladas, estarão por primeiro na fila. Feito isso, no passo 7, serão transmitidas as primeiras tarefas da fila, uma para cada máquina disponível, de forma que todas as máquinas recebam uma tarefa ou que não hajam mais tarefas na fila para as máquinas que não receberam nenhuma tarefa. À medida que uma máquina ficar ociosa ela receberá a primeira tarefa da fila, caso esta existir. Considerando o passo 8, a cada novo *job* obtido, os passos 2, 3 (este com uma pequena modificação), 4 e 5 serão realizados. Em outras palavras, este novo *job* será quebrado em tarefas e cada tarefa será inserida na fila δ. Entretanto, esta

inserção será realizada de forma a manter a ordenação da decrescente da fila, conforme o custo de cada tarefa. Com isso, será evitada a necessidade de re-ordenar a fila cada vez que novas tarefas tenham que ser inseridas em δ.

LPT Otimizado

A maior desvantagem do algoritmo LPT é a necessidade da avaliação do custo computacional das tarefas para, então, inserí-las na fila, disponibilizando-as para os RIPs. Esta avaliação, realizada através do *PDF Profiler*, apesar de poder não representar um grande custo computacional, pode impossibilitar a transmissão de tarefas imediatamente para os RIPs ociosos. Isto decorre do fato de que o escalonador pode estar ocupado realizando a execução do *PDF Profiler* no momento em que os RIPs estejam livres para receber mais tarefas. Tal situação irá comprometer o possível desempenho adquirido com o uso do LPT, já que os RIPs ociosos estarão esperando por tarefas à medida que o escalonador está realizando a análise do custo computacional das tarefas. Assim, propôs-se uma pequena modificação quanto a esta análise, para amenizar o problema descrito. Todos os passos além da alteração descrita a seguir, serão os mesmos para o LPT e o LPT Otimizado.

Para melhorar o tempo de resposta do escalonador, estabeleceu-se que a tarefa de aplicação da métrica sobre as tarefas será realizada de forma concorrente com o escalonamento das tarefas para os RIPs livres. Para tanto, assim que o escalonador precisar aplicar a métrica sobre um determinado *job*, será lançada uma nova *thread* responsável por tal função. Entretanto, anteriormente à criação desta *thread*, as tarefas deste *job* serão inseridas no final da fila estando, desta maneira, disponíveis para serem transmitidas. À medida que as *threads* terminem sua função, o custo computacional das tarefas correspondentes será atualizado, removendo-as da fila e inserido-as novamente de forma ordenada. Através desta estratégia, caso um RIP fique livre e não existam mais tarefas na fila a não ser aquelas para as quais o custo computacional não tenha sido estabelecido, uma destas tarefas será enviada assim mesmo. Caso uma *thread* finalize sua computação e as suas tarefas já tenham sido enviadas, nenhuma ação será tomada.

Multifit

No caso da utilização do algoritmo *Multifit*, as tarefas da fila são empacotadas em *bins* (durante o passo 6), através do emprego do algoritmo FFD, resultando na geração de b *bins*, sendo $1 \leq b \leq m$. Com isso, estes b *bins* são transmitidos para b máquinas (passo 7), onde cada máquina ficará com até 1 *bin*.

À medida que novos *jobs* são inseridos na fila (passo 8), são executados os passos 2, 3, 4 e 5. Como descrito anteriormente, através da aplicação destes passos são obtidas as tarefas com os seus respectivos custos computacionais. Assim que estes sejam executados, caso não existam *bins* no momento, aplica-se a técnica FFD, gerando assim os novos *bins* que são mantidos em uma fila de *bins*. Para esta fila será associado o C que foi utilizado para gerar os *bins*. Por outro

lado, no caso em que novos *jobs* sejam obtidos e já existam *bins* na fila, a estratégia adotada para o empacotamento das novas tarefas geradas será a tentativa da inserção de cada tarefa no *bin* com a menor carga existente, considerando o C atual da fila. Se todas as tarefas forem acomodadas nos *bins* atuais, nenhum outro processamento será necessário. Com o uso desta estratégia, ameniza-se o esforço computacional para obter os *bins*, disponibilizando-os mais rapidamente para os RIPs. Entretanto, se não for possível acomodá-los considerando o C atual, os *bins* existentes serão quebrados e o FFD será re-executado considerando as tarefas antigas e atuais.

6 Resultados Obtidos

Com o intuito de avaliar aspectos relativo à abordagem proposta neste trabalho, utilizaram-se diversos *jobs* que representam casos reais, juntamente com três distintas configurações de filas. Entre os componentes analisados, estão a validação das métricas nos *jobs* reais, além da verificação de desempenho para ferramenta de análise do perfil dos *jobs* e para as estratégias de escalonamento definidas.

Neste sentido, este capítulo discorre sobre o ambiente de teste, em termos de *hardware* e *software*, utilizado para executar os componentes da abordagem proposta, apresentando os *jobs* e configurações de filas utilizadas. Logo a seguir, demonstra-se uma avaliação de desempenho da ferramenta *PDF Profiler* a fim de justificar e validar seu uso para a análise do perfil dos *jobs*. Finalmente, ilustra-se os resultados obtidos com as estratégias de escalonamento desenvolvidas, comparando-as com aquelas já existentes.

6.1 Ambiente de Teste

Nesta seção será descrito o ambiente de *hardware* e *software* empregado para implementar e executar os escalonadores novos e já existentes, como também as ferramentas auxiliares desenvolvidas. Além disso serão apresentados os *jobs* utilizados no contexto dos testes realizados, para a avaliação do comportamento e desempenho dos novos escalonadores em comparação com os já existentes.

6.1.1 Ambiente de Hardware e Software

Com o intuito de se aproximar da realidade existente nas PSPs, o ambiente de *hardware* utilizado nos testes assemelha-se ao empregado no ambiente de produção de tais empresas. Este trata-se de um agregado de *blades*, que é uma arquitetura multicomputador, podendo ser classificado como um COW (*Cluster Of Workstations*). Neste sentido, utilizou-se um agregado com 5 *blades*, onde cada uma delas contém um processador *Intel Xeon 3.0GHz* (de 64 *bits*) ***quad-core***, com 8GB de memória RAM e 400GB de disco. Neste sentido, considerou-se cada *core* como sendo uma unidade de processamento ativa da arquitetura, suportando assim até 20 processos simultaneamente (quatro por *blade*). Além disso, estas *blades* estão conectadas através de uma

rede de alta velocidade, que é a Gigabit Ethernet. O sistema operacional aplicado nas máquinas é o *Windows Server 2003 R2 Standard x64 Edition*, com o *Service Pack 2*, um sistema comumente aplicado em ambientes de impressão, devido à compatibilidade com diversas versões de RIPs industriais.

Em termos de implementação, utilizou-se a linguagem Java para implementar as estratégias de escalonamento previamente descritas. Apesar da utilização de Java não ser usual na área de alto desempenho, tal linguagem vem oferecendo ao longo do tempo melhorias que permitem sua utilização, fazendo com que pesquisadores da área adotem-na como escolha para a implementação. Exemplos desta situação podem ser vistos em [**?**, 32]. Além disso, esta linguagem oferece vantagens quanto à portabilidade, extensibilidade e alto nível de abstração para as soluções propostas, além de oferecer compatibilidade com a biblioteca Java PDFBox, utilizada no *PDF Profiler*. Neste sentido, empregou-se a versão 1.5 do Java, mais especificamente a versão *JDK 1.5_16 amd64*. Assim, devido à arquitetura mencionada acima ser um multicomputador, a comunicação entre os processos deve ser realizada através da troca de mensagens. Para tanto, utilizou-se a biblioteca *MPJ-Express* [33], que é uma implementação do padrão MPI (*Message Passing Interface*) [34] puramente com o uso da linguagem Java. Devido à utilização da linguagem Java, esta biblioteca fornece conceitos de alto nível de abstração, além da portabilidade oferecida pelo linguagem, que é uma característica muito importante para as PSPs. Em termos de desempenho, a *MPJ-Express* consegue obter resultados similares quando comparada a bibliotecas como o MPICH (*MPI CHameleon*) [35] e o LAM/MPI (*Local Area Multicomputer / MPI*) [36], mostrando-se como uma boa escolha para a troca de mensagens entre processos que utilizam Java (para mais detalhes ver [33]).

6.1.2 Casos de Teste

Com o objetivo de avaliar a validade das métricas propostas sobre exemplos reais de *jobs* e para verificar o desempenho das ferramentas e estratégias implementadas, distintos *jobs* foram selecionados. Neste sentido, estabeleceu-se um conjunto fixo de 20 *jobs* com diferentes características, juntamente com três distintas configurações de fila através do uso destes. Estes componentes são apresentados nesta seção.

Jobs

Os *jobs* selecionados estão distribuídos entre 8 tipos comumente presentes no âmbito das PSPs. Dentre estes tipos, cabe ressaltar que diversos se assemelham em termos da presença de certos elementos para descrevê-los, mas podem se distinguir devido ao tipo físico de papel sobre o qual serão impressos ou ao *layout* utilizado em cada um deles. Um exemplo de cada documento é ilustrado na Figura 31. Além disso, a seguir algumas características de cada um dos 8 tipos, em conjunto com uma breve descrição sobre eles, serão apresentados, ressaltando

suas peculiaridades quanto a presença de texto, imagens e ao número de páginas:

1. **Carta**: uma carta contém uma ou mais páginas com uma grande quantidade de texto e poucas imagens. Estas imagens normalmente são logotipos, carimbos ou assinaturas.

2. **Carta de notícias**: uma carta de notícias difere de uma carta normal, contendo diversas imagens para a apresentação de propagandas, além de notícias sobre produtos e serviços. O número de páginas deste tipo de documento pode variar, conforme a idéia que se deseja passar ao cliente.

3. **Cartão**: cartões de visitas, contendo imagens de fundo e pouco texto. Em uma página deste tipo de documento podem existir diversos cartões, assim como uma única instância dos mesmos.

4. **Cartão postal**: cartões postais são utilizados para a apresentação de um lugar, produto ou serviço para o cliente. Neste sentido, são empregados textos e imagens para compor cada página. De um modo geral, cartões postais contêm duas páginas (frente e verso), podendo ser colocados diversos em uma única página.

5. ***Flyer***: estes documentos são utilizados para a divulgação de produtos, idéias, eventos, entre outros. Geralmente cada documento contêm duas páginas, que são normalmente impressas no modo frente e verso, onde cada documento estará contido em uma folha de papel. Em termos de conteúdo tanto imagens quanto textos são utilizados.

6. **Folheto**: documento composto geralmente por um grande número de páginas, sendo rico em imagens e texto. Ao final da impressão, as páginas de um documento em folheto serão dispostas de maneira a formar uma espécie de livreto.

7. **Jornal**: documento informativo contendo manchetes e notícias. Neste tipo de documento estão presentes uma ou mais páginas, cada uma com a presença de textos e imagens.

8. **Pôster**: documento com normalmente apenas uma página de tamanho grande. Neste sentido, um pôster pode conter apenas imagens, textos ou uma mistura de imagens com texto.

Levando em consideração os tipos explicitados acima, os 20 *jobs* escolhidos distribuem-se entre eles, representando exemplos reais de documentos utilizados nas PSPs. Assim sendo, estes *jobs* não possuem apenas textos e imagens, mas também contêm a presença de objetos gráficos não analisados para as métricas estipuladas, como *paths* e *shading patterns*. Estes *jobs* são apresentados na Tabela 1, que explicita suas características, juntamente com seu custo computacional estimado através da aplicação das métricas. Vale a pena ressaltar que o custo computacional estimado é obtido através da soma dos fatores de relevância obtidos pelas fórmulas.

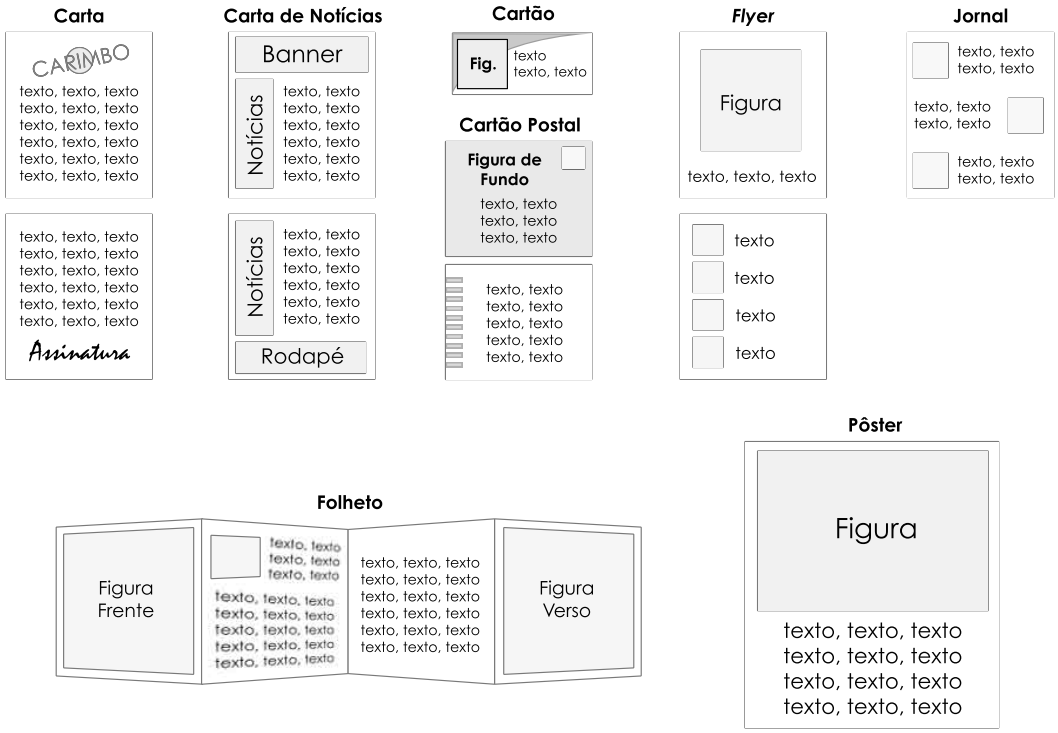

Figura 31 – Exemplos de tipos de documentos.

Tabela 1 – Características dos *jobs*.

Job	**Documentos**	**Páginas**	**Imagens**	**Páginas com texto**	**Custo computacional**
Carta 1	130	390	130	390	36.43
Carta 2	400	800	800	800	80.98
Carta 3	90	450	0	450	35.00
Cartão 1	400	200	800	200	36.36
Cartão 2	40	40	40	40	2.15
Carta de notícias 1	220	440	3740	440	71.97
Carta de notícias 2	170	340	680	340	35.21
Cartão postal 1	500	250	4000	250	40.02
Cartão postal 2	20	40	120	40	2.38
Flyer 1	80	160	320	160	16.38
Flyer 2	100	200	400	200	17.09
Flyer 3	250	500	750	500	39.52
Flyer 4	75	150	150	150	25.55
Folheto 1	9	108	162	108	6.76
Folheto 2	200	1000	1800	1000	90.07
Folheto 3	300	600	2700	600	51.34
Jornal 1	400	400	2000	400	49.61
Jornal 2	150	900	2700	900	89.73
Pôster 1	500	500	2000	500	39.44
Pôster 2	70	70	140	70	6.81

Como se pode notar, os documentos selecionados para a realização dos testes possuem uma grande variabilidade de quantidade de páginas, textos e imagens, possuindo assim uma grande abrangência de custos (de 2.15 a 90.07). Esta alta variabilidade retrata a realidade da maioria das PSPs, nas quais os *jobs* provêem de diversos cenários, acabando por acarretar tal diversificação de custos computacionais. Desta maneira, com o uso dos *jobs* estipulados será possível, então, avaliar o comportamento das estratégias novas e já existentes quanto ao balanceamento de carga

entre as unidades de processamento disponíveis.

Filas de Jobs

Para realizar a rasterização de determinados *jobs* de uma fila é importante notar que a ordem dos *jobs* da fila poderá afetar o ganho de desempenho quando empregando uma determinada estratégia de escalonamento. Neste sentido, torna-se interessante a execução das estratégias sobre diferentes configurações de fila, avaliando assim como será realizado o escalonamento para as unidades de processamento, assim como o impacto de tal escalonamento sobre o possível ganho de desempenho.

Sabendo que os *jobs* da fila podem ser disponibilizados a qualquer momento, nas configurações de fila testadas estabeleceu-se que alguns *jobs* estão disponíveis inicialmente na fila (chamados de *jobs* iniciais) e outros serão inseridos nesta depois de x segundos (*jobs* tardios). Definiu-se, então, que $10 \leq x \leq 60$. Com o uso deste intervalo pequeno, novas tarefas serão disponibilizadas rapidamente, tornando possível visualizar como cada estratégia trata da análise do perfil, organização e distribuição dos *jobs* de forma concorrente. Neste contexto, para a simulação do ambiente das PSPs, os *jobs* tardios são inseridos seqüencialmente na fila e o tempo para inserir um destes apenas começará a ser contabilizado à medida que seu predecessor já tenha sido inserido na fila. Com isso, definiu-se três distintas configurações de fila, através da utilização dos 20 tipos de *jobs* descritos na Seção 6.1.2, que estão apresentadas na Tabela 2.

As configurações de fila estipuladas foram direcionadas para tratar de três casos distintos no cenário dos algoritmos escolhidos. Estes cenários dizem respeito à distribuição de carga dos *jobs* no contexto da fila, provocando assim diferentes conseqüências para as estratégias empregadas. Estes cenários são descritos a seguir:

1. fila de *jobs* com uma variação de cargas do primeiro *job* ao último (Fila 1);
2. fila de *jobs* ordenados respectivamente da maior carga a menor (Fila 2);
3. fila de *jobs* ordenados da menor carga a maior (Fila 3).

O primeiro caso foi definido para a avaliação do comportamento das novas estratégias quanto a variações de cargas dos *jobs*, verificando assim se as estratégias são capazes de lidar com este tipo de irregularidade, balanceando as cargas dentre os RIPs disponíveis. Além disso, tornou-se possível avaliar o impacto desta variação nas estratégias de escalonamento já existentes.

O segundo caso representa a melhor situação para o algoritmo LS, pois as tarefas serão inseridas na fila de forma decrescente em termos de custo computacional, prevenindo o pior caso do algoritmo. Portanto, através desta configuração tornou-se possível avaliar qual a influência das computações adicionais necessárias para a aplicação dos algoritmos utilizados em comparação com o melhor caso para o algoritmo das estratégias de escalonamento existentes.

Tabela 2 – Configurações de filas de *jobs*.

(a) Fila 1

Posição na Fila	*Job*	Tipo	Atraso (segundos)
1	Folheto 1	Inicial	-
2	Folheto 2	Inicial	-
3	Folheto 3	Inicial	-
4	Cartão 1	Inicial	-
5	Cartão 2	Tardio	60
6	*Flyer* 1	Tardio	50
7	*Flyer* 2	Tardio	40
8	*Flyer* 3	Tardio	30
9	*Flyer* 4	Tardio	20
10	Carta 1	Tardio	10
11	Carta 2	Tardio	20
12	Carta 3	Tardio	30
13	Carta de notícias 1	Tardio	40
14	Carta de notícias 2	Tardio	50
15	Jornal 1	Tardio	60
16	Jornal 2	Tardio	30
17	Cartão postal 1	Tardio	40
18	Cartão postal 2	Tardio	20
19	Pôster 1	Tardio	50
20	Pôster 2	Tardio	60

(b) Fila 2

Posição na Fila	*Job*	Tipo	Atraso (segundos)
1	Folheto 2	Inicial	-
2	Jornal 2	Inicial	-
3	Carta 2	Inicial	-
4	Carta de notícias 1	Tardio	40
5	Folheto 3	Tardio	10
6	Jornal 1	Tardio	60
7	Cartão postal 1	Tardio	40
8	*Flyer* 3	Tardio	30
9	Pôster 1	Tardio	60
10	Carta 1	Tardio	10
11	Cartão 1	Tardio	30
12	Carta de notícias 2	Tardio	50
13	Carta 3	Tardio	30
14	*Flyer* 4	Tardio	20
15	*Flyer* 2	Tardio	40
16	*Flyer* 1	Tardio	50
17	Pôster 2	Tardio	50
18	Folheto 1	Tardio	10
19	Cartão postal 2	Tardio	20
20	Cartão 2	Tardio	60

(c) Fila 3

Posição da Fila	*Jobs*	Tipo	Atraso (segundos)
1	Cartão 2	Inicial	-
2	Cartão postal 2	Inicial	-
3	Folheto 1	Inicial	-
4	Pôster 2	Inicial	-
5	*Flyer* 1	Inicial	-
6	*Flyer* 2	Inicial	-
7	*Flyer* 4	Inicial	-
8	Carta 3	Tardio	30
9	Carta de notícias 2	Tardio	50
10	Cartão 1	Tardio	30
11	Carta 1	Tardio	10
12	Pôster 1	Tardio	60
13	*Flyer* 3	Tardio	30
14	Cartão postal 1	Tardio	40
15	Jornal 1	Tardio	60
16	Folheto 3	Tardio	10
17	Carta de notícias 1	Tardio	40
18	Carta 2	Tardio	20
19	Jornal 2	Tardio	50
20	Folheto 2	Tardio	10

Por fim, em contraste com o segundo caso, o terceiro representa o pior caso para o algoritmo LS. Assim, avaliou-se qual o possível ganho de desempenho com a utilização dos novos algoritmos empregados no cenário da rasterização em comparação com o algoritmo LS em tal situação.

6.2 Validade das Métricas

Até o momento definiram-se diversas métricas para medir o custo computacional de um *job* através de casos de teste experimentais, totalmente direcionados para seus respectivos propósitos. Entretanto, tais métricas devem corresponder ao custo computacional de *jobs* reais, para

que sua aplicação se justifique. Para tanto, os *jobs* selecionados para a realização dos testes, que tratam-se de exemplos reais de mercado, foram executados (rasterizados) seqüencialmente obtendo assim um tempo de execução para cada um destes. Desta maneira, tornou-se possível comparar o custo computacional estimado com este tempo de execução, verificando assim se a métrica condiz com os custos reais. Estes resultados são demonstrados na Tabela 3, na qual os *jobs* estão ordenados de forma decrescente através do custo computacional, apresentando o valor da média de 20 execuções para cada *job*:

Tabela 3 – Tempo de rasterização e métricas.

Job	**Tempo de rasterização (segundos)**	**Custo computacional estimado**
Folheto 2	1907.12	90.07
Jornal 2	1830.28	89.73
Carta 2	1703.20	80.98
Carta de notícias 1	1614.08	71.97
Folheto 3	1399.97	51.34
Jornal 1	1358.73	49.61
Cartão postal 1	959.20	40.02
Flyer 3	862.21	39.52
Pôster 1	877.05	39.44
Carta 1	841.30	36.43
Cartão 1	830.23	36.36
Carta de notícias 2	879.18	35.21
Carta 3	874.15	35.00
Flyer 4	572.58	25.55
Flyer 2	276.67	17.09
Flyer 1	375.92	16.38
Pôster 2	50.004	6.81
Folheto 1	41.844	6.76
Cartão postal 2	16.47	2.38
Cartão 2	13.402	2.15

É possível notar que a métrica utilizada provê uma boa estimativa para os custos computacionais dos *jobs* apresentados. Neste contexto, percebe-se que através do custo estimado pode-se estabelecer faixas de tempo de execução que são retratados pelos diferentes valores de métrica. Desta maneira, com a métrica pode-se facilmente diferenciar *jobs* custosos daqueles mais leves, fornecendo assim uma boa base para o propósito de balanceamento de carga que deseja-se atingir neste trabalho.

Por outro lado, à medida que os custos ficam próximos, verifica-se que existe um erro envolvido. Tal situação pode ser visualizada, analisando o custos dos exemplos *Flyer* 1, Carta de notícias 2 e Carta 3, onde o tempo de execução é maior do que alguns exemplos com custos computacionais estimados menores. Acredita-se que este erro é resultado da existência de objetos PDF nos *jobs*, os quais não foram considerados para a formulação das métricas, como, por exemplo, *paths* e *shading pattern objects*. Assim, estes objetos não considerados podem adicionar um determinado tempo de rasterização à computação, o qual não está sendo contabilizado para a estimativa proposta. Entretanto, a métrica ainda é capaz de dar uma boa estimativa dos custos dos *jobs*, mesmo com este erro associado, podendo assim ser aplicada para o objetivo de balanceamento de carga.

6.3 Custo Associado ao PDF Profiler

Como visto anteriormente, as estratégias utilizadas para melhorar o desempenho da fase de rasterização enquadram-se no cenário de escalonamento não-determinístico. Devido a tal fato, nenhuma informação sobre as tarefas é conhecida previamente, provocando assim a necessidade da análise de cada uma delas para a aplicação dos algortimos empregados. Entretanto, está claro que se esta análise se mostrar muito custosa, o desempenho das estratégias não será satisfatório, pois, assim, tal fase irá impedir a distribuição de tarefas em um tempo hábil para manter os RIPs trabalhando continuamente. Desta maneira, nesta seção apresenta-se uma avaliação sobre o custo da análise das tarefas, a qual é realizada pela ferramenta *PDF Profiler*, verificando assim a viabilidade deste pré-processamento. A Tabela 4 demonstra os tempos obtidos (em segundos) para a análise do perfil dos diferentes *jobs* utilizados no trabalho, juntamente com o tempo de rasterização correspondente a cada um deles e uma proporção percentual, indicando qual a grandeza de tal análise quando comparada ao respectivo tempo total de rasterização.

Tabela 4 – Custo da análise do perfil dos *jobs*.

Job	**Tempo de rasterização**	**Tempo para a análise do perfil**	**Percentual da análise**
Carta 1	841.30	2.26	0.27
Carta 2	1703.20	10.63	0.62
Carta 3	874.15	2.46	0.28
Cartão 1	830.23	18.89	2.28
Cartão 2	13.402	0.92	6.86
Carta de notícias 1	1614.08	14.49	0.90
Carta de notícias 2	879.18	7.42	0.84
Cartão postal 1	959.20	1.56	0.16
Cartão postal 2	16.47	0.96	5.83
Flyer 1	375.92	2.05	0.55
Flyer 2	276.67	0.69	0.25
Flyer 3	862.21	5.11	0.59
Flyer 4	572.58	1.41	0.25
Folheto 1	41.844	4.08	9.75
Folheto 2	1907.12	5.86	0.31
Folheto 3	1399.97	4.64	0.33
Jornal 1	1358.73	3.04	0.22
Jornal 2	1830.28	3.3	0.18
Pôster 1	877.05	10.6	1.21
Pôster 2	50.004	1.54	3.08

Através da tabela de resultados pode-se notar que em alguns *jobs* o tempo de análise se mostra muito maior do que em outros. Isto ocorre devido à quantidade de objetos do próprio PDF (não apenas os gráficos), que influenciam o tempo de interpretação do arquivo para que este seja convertido para a representação de objetos Java empregada na biblioteca PDFBox. Além disso, outros fatores impactantes em tal análise são o tamanho em disco do arquivo PDF, pois este fator está diretamente relacionado ao número de operações de leitura do disco que devem ser realizadas, em conjunto com a combinação de quantidade de imagens, textos e páginas que cada *job* possui, os quais são os objetos de interesse e demandam processamento por parte da ferramenta. Apesar de existir estas variações, os tempos de análises para os arquivos não ultrapassam os 20 segundos, com uma média de aproximadamente 5 segundos de análise.

Por outro lado, levando-se em consideração o percentual da análise do perfil dos *jobs* em relação ao tempo total de rasterização, pode-se notar que a grande maioria é ínfima, permanecendo em torno de 0.20% a 2.00%. Assim, o uso da ferramenta pode ser realizado facilmente, proporcionando um desempenho satisfatório em relação ao esforço total que será aplicado sobre os *jobs* posteriormente. Entretanto, percebe-se um percentual maior de aproximadamente 3%, 6%, 7% e 10% para os *jobs* Pôster 2, Cartão Postal 2, Cartão 2 e Folheto 1 respectivamente. Esta situação decorre do fato de que estes *jobs* são pequenos em termos de tempo de rasterização, sofrendo assim uma maior influência na sua análise de perfil por parte do tempo de operações de E/S e para a conversão de seus objetos para a representação da biblioteca PDFBox. Neste sentido, acredita-se que quanto menor o custo computacional para a rasterização do *job* e quanto maior o número de objetos nele contidos, maior será a influência destas duas fases iniciais. Vale a pena ressaltar que o número de objetos diz respeito a todos os objetos do *job*, não apenas os gráficos, pois todos estes devem ser interpretados e convertidos para a representação de dados fornecida pela biblioteca. Apesar disso, estas operações são de vital importância para a execução da ferramenta *PDF Profiler* e devem ser realizadas por completo, representando assim uma barreira que não pode ser quebrada. Todavia, a aplicação da análise se justifica, pois mesmo com tal barreira ela representa, no pior caso, um valor em segundos baixo, 10 vezes menor do que o tempo de rasterização do correspondente *job*.

6.4 Análise de Desempenho dos Escalonadores

Esta seção discorre sobre medidas de desempenho para as estratégias propostas, comparando-as com as anteriores já empregadas e com o tempo sequencial de rasterização da fila de *jobs* como um todo (17283.60 segundos). Neste sentido, 10 execuções foram realizadas sobre as 6 estratégias consideradas neste trabalho, para cada uma das três configurações de fila descritas. Destes resultados, então, foram descartados aqueles de maior e menor tempo para cada bateria de teste, obtendo uma média dos outros 8 valores restantes. Vale a pena ressaltar que para executar qualquer uma das estratégias descritas, 1 processo sempre será o escalonador, que apenas realizará a distribuição de tarefas sem rasterizar nenhum dos *jobs*. Assim, para a existência de RIPs em paralelo o número mínimo de processos relacionados deve ser 3, contando com 2 RIPs e um escalonador. Desta maneira, para a execução dos testes variou-se o número de processos de 3 a 20, com ressalva da estratégia existente de *N RIPs por job*. Nesta estratégia, foram considerados 4 tamanhos de grupos distintos: 2 RIPs por *job*, 3 RIPs por *job*, 4 RIPs por *job* e 5 RIPs por *job*. Para cada tamanho de grupo, o número de processos foi direcionado para comportar sempre 1 ou mais grupos inteiros, até que este número chegasse o mais perto de 20 (que corresponde ao total de unidades de processamento disponíveis).

Para analisar o desempenho das estratégias, então, primeiramente são apresentados os resultados de *speed-up* (fator de aceleração) obtidos com cada estratégia através da primeira fila

(Fila 1), onde as cargas dos *jobs* foram distribuídas aleatoriamente na mesma. Para uma melhor visualização destes resultados, a Tabela 5 descreve os fatores de aceleração obtidos para a estratégia de *N RIPs por job* para os determinados números de processos envolvidos, enquanto a Figura 32 ilustra um gráfico com as demais estratégias, variando o número de processos de 3 a 20.

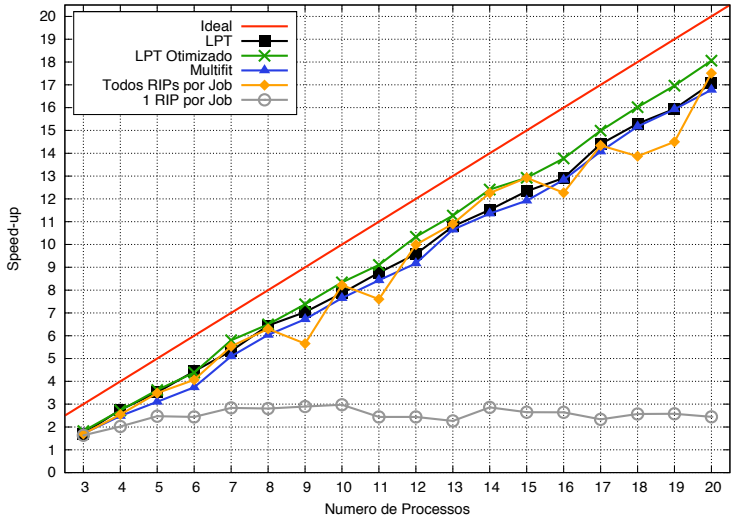

Figura 32 – *Speed-up* para a Fila 1 - 1 RIP por *job*, Todos RIPs por *job*, LPT, *Multifit* e LPT Otimizado.

Tabela 5 – *Speed-up* para a Fila 1 - N RIPs por *job*

Número de RIPs por *job*	**Número de Processos**												
	3	4	5	6	7	9	10	11	13	15	16	17	19
2	1.42	-	2.32	-	2.57	2.77	-	3.17	2.84	2.97	-	2.77	2.98
3	-	2.66	-	-	4.72	-	4.96	-	4.99	-	5.20	-	4.99
4	-	-	3.74	-	-	4.15	-	-	3.91	-	-	3.47	-
5	-	-	-	4.26	-	-	-	3.72	-	-	3.54	-	-

Como é possível verificar nos resultados apresentados a estratégia *1 RIP por job* manteve seu fator de aceleração para a variação de processos utilizada por volta de 3, sem qualquer tendência de crescimento. Isto ocorre, pois no cenário em que são aplicados um número baixo de RIPs, os *jobs* iniciais são distribuídos imediatamente para eles, fazendo-os trabalhar. Entretanto, considerando que o número de *jobs* iniciais é pequeno e é escalonado imediatamente para os primeiros RIPs ociosos, à medida que o número de RIPs disponíveis aumenta, diversos ficarão sem tarefas para computar, devido ao fato de que os *jobs* tardios são inseridos incrementalmente na fila e possuem um atraso para o mesmo. Desta maneira, a adição de mais processos nesta estratégia não irá representar um ganho efetivo, pois várias unidades de processamento não serão empregadas para a rasterização.

Os *speed-ups* obtidos para a estratégia *N RIPs por job*, mostraram-se insatisfatórios também. Como pode-se visualizar na Tabela 32 (b), existe uma flutuação dos fatores obtidos, com aumentos e diminuições nas medidas. Além disso, nota-se que o melhor resultado atinge um máximo de 5.2 com o uso de 16 processos (casualmente, *3 RIPs por job*). Estas situações podem

ser explicadas devido ao fato de que tal estratégia acaba provocando o mau uso dos RIPs, além de um balanceamento de carga pobre, pois RIPs de um mesmo grupo poderão receber tarefas de tamanhos muito diferentes, fazendo com que aqueles que não terminaram o seu processamento acabem por impedir os demais do grupo de receberem novas tarefas. Assim, a paralelização fica comprometida, atrasando a rasterização dos *jobs* remanescentes na fila.

Dentre as estratégias existentes, a *Todos RIPs por job* foi na qual pode-se visualizar os melhores resultados. Neste contexto, o melhor *speed-up* obtido foi de 17.51 com o uso de 20 processos. Entretanto, verifica-se que o comportamento de sua curva de desempenho possui flutuações, gerando situações imprevisíveis e fatores de aceleração insatisfatórios para determinados números de processos. Para compreender estas anomalias, deve-se analisar o funcionamento desta estratégia mais a fundo. Neste contexto, sabe-se que tal estratégia quebra os *jobs* em fragmentos, diminuindo o grão a ser computado por cada RIP. Cada tarefa, então, é transmitida para os RIPs ociosos, assim que elas estiverem disponíveis. Porém, nenhuma consideração quanto a carga da tarefa que está sendo transmitida é considerada. Desta maneira, esta distribuição torna suscetível a ocorrência da sobrecarga e sub-carga de RIPs, gerando o comportamento visualizado.

Partindo, então, para a análise de desempenho das estratégias desenvolvidas, pode-se verificar que as grandes flutuações nos resultados foram eliminadas. Isto decorre do fato de que estas estratégias consideram a carga das tarefas para distribuí-las, a fim de balancear o esforço computacional entre as unidades de processamento disponíveis. Levando-se em consideração as estratégias LPT e *Multifit* percebe-se que o *speed-up* obtido, apesar de constante, mostra-se em diversos pontos abaixo daquele visto com a *Todos RIPs por job*. Acredita-se que esta situação acontece, devido à necessidade por parte destas estratégias de sempre realizar a análise do perfil dos *jobs* previamente a disponibilização destes na fila de tarefas. Assim, caso existam RIPs ociosos durante a análise de um ou mais *jobs*, estes terão de esperar até que este processamento seja realizado por completo para receberem uma nova tarefa. Isto acaba por acarretar um *overhead* que se reflete diretamente na curva de *speed-up*. Por outro lado, verifica-se que o *speed-up* da estratégia *Multifit* está, na maioria das vezes, abaixo da LPT, pois com o uso do *Multifit* adiciona-se ainda mais *overhead* para o empacotamento das tarefas em *bins*. Por fim, os melhores resultados foram apresentados pela estratégia *LPT Otimizado*, que além de amenizar as flutuações atingiu *speed-ups* que superam todas as demais estratégias. A grande vantagem oferecida por esta estratégia é o fato dela conseguir transmitir imediatamente as tarefas para os RIPs ociosos, concorrentemente a análise do perfil dos *jobs*. Com isso, ameniza-se o *overhead* do LPT para sempre analisar as tarefas previamente ao seu envio.

Os resultados apresentados até então refletem-se diretamente no percentual de utilização de cada unidade de processamento. Para verificar esta medida, então, apresenta-se a eficiência obtida para cada estratégia na Tabela 6.

Com a análise de eficiência para cada estratégia pode-se visualizar o efeito causado com os fatores de aceleração obtidos, quando varia-se o número de processos. Para a estratégia *1*

Tabela 6 – Medida de eficiência com a execução da Fila 1.

Estratégia	**Número de Processos**																	
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1 RIP por *job*	0.81	0.67	0.61	0.48	0.47	0.40	0.36	0.33	0.24	0.22	0.18	0.21	0.18	0.17	0.14	0.15	0.14	0.12
2 RIPs por *job*	0.71	-	0.58	-	0.42	-	0.34	-	0.31	-	0.23	-	0.21	-	0.17	-	0.16	-
3 RIPs por *job*	-	0.88	-	-	0.78	-	-	0.55	-	-	0.41	-	-	0.34	-	-	0.26	-
4 RIPs por *job*	-	-	0.93	-	-	-	0.51	-	-	-	0.32	-	-	-	0.21	-	-	-
5 RIPs por *job*	-	-	-	0.85	-	-	-	-	0.37	-	-	-	-	0.23	-	-	-	-
Todos RIPs por *job*	0.84	0.84	0.87	0.81	0.92	0.90	0.70	0.91	0.76	0.90	0.90	0.94	0.92	0.81	0.89	0.81	0.80	0.92
LPT	0.85	0.92	0.88	0.89	0.89	0.92	0.88	0.87	0.88	0.87	0.90	0.89	0.88	0.86	0.90	0.90	0.89	0.90
Multifit	0.86	0.83	0.78	0.75	0.85	0.86	0.84	0.85	0.84	0.83	0.89	0.87	0.85	0.86	0.88	0.89	0.88	0.88
LPT Otimizado	0.91	0.91	0.91	0.88	0.97	0.93	0.92	0.93	0.91	0.94	0.94	0.95	0.92	0.92	0.94	0.94	0.94	0.95

RIP por job verifica-se claramente que à medida que mais processos são inseridos a eficiência cai abruptamente, devido a existência de diversos RIPs ociosos durante a execução. O mesmo percebe-se para os grupos considerados (*N RIPs por job*), onde a adição de mais processos não representa um ganho efetivo de desempenho, fazendo com que a eficiência permaneça sempre baixa. No caso do uso de *Todos os RIPs por job* pode-se visualizar eficiências altas em torno de 90% para os casos em que o algoritmo LS obtém um desempenho satisfatório, mas também notam-se eficiências menores de 70% a 85% para aqueles cenários em que existe a queda de desempenho, provocando assim a existência de flutuações nesta medida também. As estratégias desenvolvidas mantiveram eficiências altas, mesmo com a adição de mais processos. Considerando-se uma média dos valores de eficiência para estas estrátegias observa-se um valor de 84% para a estratégia *Multifit*, de 88% para o LPT e de 92% para o LPT Otimizado.

Partindo, então, para a análise de desempenho das aplicações consideradas com a segunda configuração de filas (Fila 2), na qual as cargas dos *jobs* estão ordenadas da maior para a menor, os resultados de *speed-up* são apresentados na Figura 33 e na Tabela 7. Cabe ressaltar que esta configuração visava a organização das tarefas de maneira a tentar prevenir o pior caso do algoritmo LS, enquanto as maiores tarefas serão executadas por primeiro.

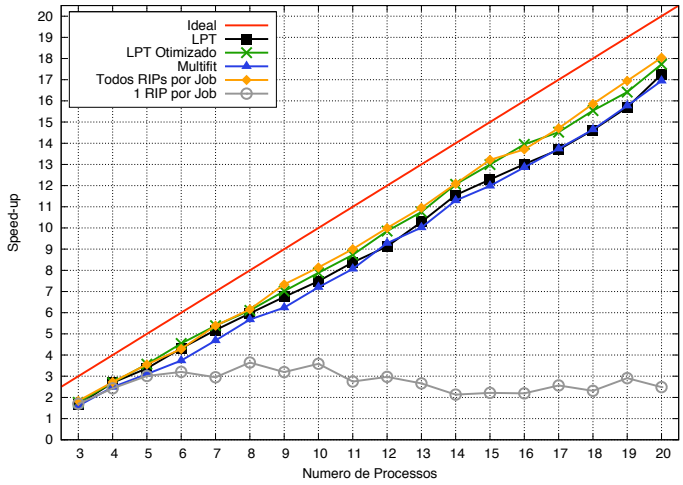

Figura 33 – *Speed-up* para a Fila 2 - 1 RIP por *job*, Todos RIPs por *job*, LPT, *Multifit* e LPT Otimizado.

Os resultados ilustrados para a estratégia *1 RIP por job* demonstram que houve um pequeno aumento no fator de aceleração, se comparados aos resultados da Fila 1, mas estes se mantém

Tabela 7 – *Speed-up* para a Fila 2 - N RIPs por *job*.

Número de RIPs por *job*	**Número de Processos**												
	3	4	5	6	7	9	10	11	13	15	16	17	19
2	1.38	-	2.66	-	3.07	3.05	-	2.98	3.40	3.13	-	3.04	3.04
3	-	2.64	-	-	4.50	-	5.12	-	5.01	-	3.82	-	4.29
4	-	-	3.54	-	-	3.66	-	-	3.59	-	-	3.63	-
5	-	-	-	4.38	-	-	-	3.98	-	-	3.73	-	-

numa faixa de 3, sem tendências de crescimento. Pode-se concluir, então, que o fato da existência de RIPs ociosos continua por atrapalhar o ganho de desempenho da estratégia, que não apresenta uma escalabilidade boa. Além disso, nota-se que existe novamente uma flutuação dos resultados. Acredita-se que esta situação ocorre devido a tamanho do grão ser muito grande, acabando por gerar a sobrecarga e sub-carga de RIPs.

Para a estratégia de *N RIPs por job* não é possível notar quaisquer mudanças significativas no ganho de desempenho. O problema da existência de RIPs ociosos permanece como um fator agravante para o ganho de desempenho, resultando em flutuações nos valores obtidos e em fatores de aceleração baixos. Este comportamento pode ser observado para todas as configurações de grupos analisadas.

Observando o *speed-up* da estratégia *Todos os RIPs por job* pode-se visualizar que houve uma linearização dos resultados, sem a presença de variações significativas, como picos e vales. Com a configuração de filas considerada, foi possível distribuir primeiramente as tarefas mais pesadas, amenizando o fator da sobrecarga de RIPs no final do processamento, que representam o pior caso para o algoritmo LS. Com isso, pode-se dizer que esta configuração de fila simula o funcionamento do algoritmo LPT para a estratégia em questão, sem a necessidade de analisar o custo computacional das tarefas e ordenar a fila. Desta maneira, esta estratégia consegue apresentar os melhores resultados dentre todas as demais.

Considerando as estratégias desenvolvidas, pode-se notar que estas mantiveram resultados sem comportamentos estranhos, mas com fatores de aceleração menores do que a estratégia de *Todos os RIPs por job*. Isto pode ser devidamente explicado por causa da realização da análise das tarefas que acaba gerando um *overhead*, onde para a dada configuração de filas não traz nenhum benefício, já que os *jobs* já estão ordenados. Por outro lado, analisando a estratégia LPT Otimizado pode-se dizer que os resultados apresentados não afastam-se muito daqueles da estratégia *Todos os RIPs por job*, devido à capacidade de transmitir as tarefas imediatamente aos RIPs ociosos. Entretanto, existe ainda um pouco do *overhead* gerado pelas *threads* que analisam o perfil dos *jobs*, provocando uma perda de desempenho.

No contexto da Fila 2, as medidas de eficiência para as estratégias são apresentadas na Tabela 8. Como esperado, a eficiência das estratégias *1 RIP por job* e *N RIPs por job* apresentou-se pequena à medida que mais unidades de processamento foram adicionadas, devido à baixa escalabilidade de tais aplicações. Por outro lado, a eficiência da estratégia de *Todos os RIPs por job* manteve-se alta, com uma média de 92%. Para as estratégias desenvolvidas, obteve-se os valores médios de 82%, 86% e 90% para o LPT, *Multifit* e LPT Otimizado respectivamente.

Neste sentido, nota-se que os dois primeiros valores afastam-se bastante do obtido com o *Todos os RIPs por job*, mas o último consegue apresentar uma eficiência bastante similar, pois este não sofre tanto com o impacto da sobrecarga desnecessária gerada com a análise dos *jobs*.

Tabela 8 – Medida de eficiência com a execução da Fila 2.

Estratégia	**Número de Processos**																	
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1 RIP por *job*	0.85	0.81	0.75	0.64	0.49	0.52	0.40	0.40	0.27	0.27	0.22	0.16	0.16	0.15	0.16	0.14	0.16	0.13
2 RIPs por *job*	0.69	-	0.66	-	0.51	-	0.38	-	0.29	-	0.28	-	0.22	-	0.19	-	0.16	-
3 RIPs por *job*	-	0.88	-	-	0.75	-	-	0.56	-	-	0.41	-	-	0.25	-	-	0.23	-
4 RIPs por *job*	-	-	0.88	-	-	-	0.45	-	-	-	0.29	-	-	-	0.22	-	-	-
5 RIPs por *job*	-	-	-	0.87	-	-	-	-	0.39	-	-	-	-	0.24	-	-	-	-
Todos RIPs por *job*	0.91	0.90	0.88	0.85	0.89	0.87	0.91	0.90	0.90	0.90	0.91	0.92	0.94	0.91	0.91	0.93	0.94	0.94
LPT	0.83	0.90	0.84	0.86	0.86	0.85	0.84	0.83	0.84	0.83	0.86	0.89	0.88	0.87	0.86	0.86	0.87	0.91
Multifit	0.80	0.83	0.78	0.75	0.78	0.81	0.78	0.80	0.81	0.84	0.83	0.87	0.86	0.86	0.86	0.86	0.88	0.89
LPT Otimizado	0.88	0.90	0.89	0.91	0.90	0.87	0.88	0.88	0.87	0.90	0.90	0.93	0.93	0.93	0.91	0.91	0.91	0.93

Por fim, executou-se as estratégias com o uso da terceira fila proposta, onde os *jobs* foram ordenados de maneira que os menores fossem disponibilizados primeiramente aqueles com cargas mais pesadas. Os resultados obtidos são demonstrados na Figura 34 e na Tabela 9.

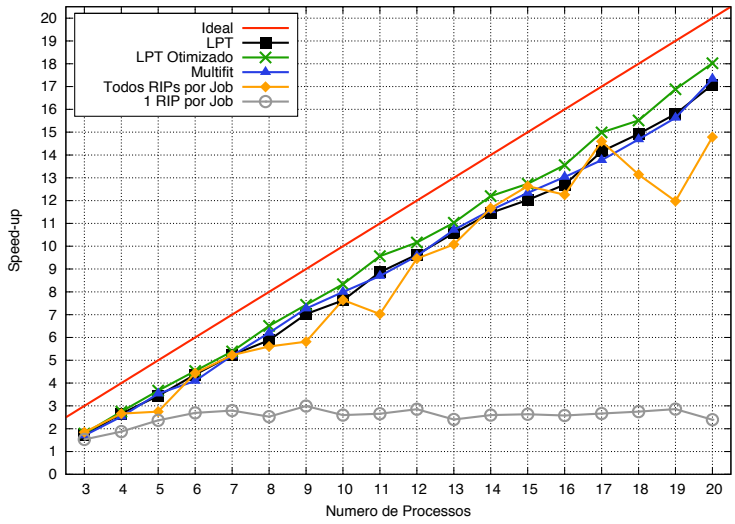

Figura 34 – *Speed-up* para a Fila 3 - 1 RIP por *job*, Todos RIPs por *job*, LPT, *Multifit* e LPT Otimizado.

Tabela 9 – *Speed-up* para a Fila 3 - N RIPs por *job*

Número de RIPs por *job*	**Número de Processos**												
	3	4	5	6	7	9	10	11	13	15	16	17	19
2	1.61	-	2.54	-	3.09	3.36	-	3.04	2.96	3.04	-	2.78	2.99
3	-	2.57	-	-	4.43	-	4.51	-	5.04	-	4.60	-	4.56
4	-	-	3.47	-	-	3.82	-	-	3.44	-	-	4.25	-
5	-	-	-	4.21	-	-	-	3.95	-	-	3.50	-	-

Os *speed-ups* para as estratégias desenvolvidas mostraram-se novamente com um comportamento previsível. Estes mantiveram-se perto do ideal, nos quais deve-se destacar aquele obtido pela estratégia LPT Otimizado, que superou os fatores de aceleração de todas as demais estratégias. As estratégias LPT e *Multifit* obtiveram seus correspondentes desempenhos influenciados

pelo *overhead* da análise do perfil dos *jobs* (e empacotamento no caso do *Multifit*), que atrasa a distribuição das tarefas. Apesar disto, estas estratégias conseguiram superar ou igualar, na maioria dos casos, os desempenhos obtidos pelas estratégias existentes.

Considerando as estratégias *1 RIP por job* e *N RIPs por job* nenhuma alteração do comportamento previamente descrito foi observado. Neste sentido, estas estratégias apresentaram uma baixa escalabilidade, sem nenhum ganho significativo à medida que mais processos foram adicionados. Por outro lado, a estratégia *Todos os RIPs por job* apresentou um comportamento particular com aumentos e quedas bruscas de desempenho. Neste contexto, acredita-se que o desbalanceamento de carga provocado pela execução das maiores tarefas por último, acaba gerando sobrecargas e sub-cargas de RIPs que são diretamente impactantes nesta estratégia. Isto se deve ao fato de que no âmbito desta estratégia em alguns momentos os RIPs sobrecarregados no final estarão com as maiores cargas da fila, provocando quedas abruptas de desempenho. Por outro lado, em alguns casos esta situação não ocorrerá, pois diversos RIPs estarão processando cargas pesadas ao final do processamento, terminando sua computação em tempos não muito distantes uns dos outros.

Por fim, a Tabela 10 apresenta as medidas de eficiência para a terceira configuração de fila.

Tabela 10 – Medida de eficiência com a execução da Fila 3.

Estratégia	**Número de Processos**																	
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1 RIP por *job*	0.76	0.63	0.59	0.54	0.47	0.36	0.37	0.29	0.27	0.26	0.20	0.20	0.19	0.17	0.17	0.16	0.16	0.13
2 RIPs por *job*	0.80	-	0.63	-	0.51	-	0.42	-	0.30	-	0.24	-	0.21	-	0.17	-	0.16	-
3 RIPs por *job*	-	0.85	-	-	0.73	-	-	0.50	-	-	0.42	-	-	0.30	-	-	0.25	-
4 RIPs por *job*	-	-	0.86	-	-	-	0.47	-	-	-	0.28	-	-	-	0.26	-	-	-
5 RIPs por *job*	-	-	-	0.84	-	-	-	-	0.39	-	-	-	-	0.23	-	-	-	-
Todos RIPs por *job*	0.92	0.89	0.69	0.89	0.87	0.80	0.73	0.85	0.70	0.86	0.84	0.90	0.90	0.82	0.91	0.77	0.67	0.78
LPT	0.86	0.89	0.86	0.87	0.87	0.84	0.88	0.85	0.89	0.88	0.88	0.88	0.86	0.85	0.88	0.88	0.88	0.90
Multifit	0.85	0.84	0.88	0.82	0.87	0.89	0.91	0.89	0.87	0.87	0.89	0.89	0.88	0.87	0.86	0.86	0.87	0.91
LPT Otimizado	0.89	0.92	0.92	0.91	0.90	0.93	0.93	0.93	0.96	0.92	0.92	0.94	0.91	0.90	0.94	0.91	0.94	0.95

Como já se sabe, o comportamento observado no *speed-up* está diretamente relacionado às eficiências obtidas. Desta forma, as estratégias *1 RIP por job* e *N RIPs por job* apresentam eficiências menores com a inserção de mais processos rasterizadores. Devido às flutuações bruscas no desempenho da estratégia *Todos RIPs por job*, existe uma grande variação nas eficiências observadas, partindo de 66% (com 19 processos) até 92% (com 3 processos). Devido à tal situação, a eficiência média desta estratégia ficou em torno de 82%. Para as estratégias desenvolvidas, novamente pode-se observar eficiências médias altas de 87% para o LPT e *Multifit*, e de 92% para o LPT Otimizado, que supera todas as demais estratégias.

7 Conclusão e Trabalhos Futuros

O trabalho apresentado mostrou-se como um estudo válido para melhores resultados na rasterização de *jobs* a partir da computação de alto desempenho. Através da definição de métricas, que se mostraram consistentes para o estabelecimento de um perfil de *jobs*, pôde-se aplicar estratégias de escalonamento com competitividades melhores do que as existentes, obtendo assim resultados mais satisfatórios. Tornou-se possível, então, a resolução de problemas de balanceamento de carga das estratégias existentes, fornecendo aplicações capazes de apresentar um comportamento previsível e com um desempenho que se mostrou satisfatório, atingindo *speed-ups* perto do ideal para a variação de unidades de processamento considerada. Com isso, os recursos disponíveis foram melhor utilizados, como demonstrado através das tabelas de eficiência descritas nos resultados obtidos. Neste contexto, o trabalho realizado englobou a aplicação de diversos estudos para o desenvolvimento das estratégias apresentadas. Entre estes destacam-se os problemas existentes na área de rasterização, a composição de um documento PDF, o enquadramento do problema em classificações de escalonamento existentes, modelos teóricos de algoritmos para o problema de escalonamento, granularidade, a seleção de possíveis elementos PDF que influenciam no custo do processamento de um documento, além do desenvolvimento de ferramentas auxiliares que permitiram o desenvolvimento das novas estratégias de escalonamento.

Cabe ressaltar que métricas similares às apresentadas neste documento, mas no cenário de RIPs industriais, também foram definidas durante o período do mestrado, dando início a geração de uma patente para a empresa *Hewlett Packard* (HP). No contexto das métricas discutidas neste trabalho, deseja-se realizar a submissão de um artigo para o *journal* MTAP (*Multimedia Tools and Applications*) no mês de Janeiro de 2009, demonstrando os resultados obtidos com tais métricas, que foram definidas a partir da utilização do RIP *open-source ImageMagick*.

Por outro lado, através dos resultados apresentados pode-se estabelecer um ponto de partida para trabalhos futuros, buscando uma maior otimização da fase de rasterização de documentos. Neste contexto, pretende-se focar na utilização de RIPs industriais, os quais possuem diversas maneiras de otimizar o processamento de um *job*, caso lhes seja passada uma devida configuração que condiz com o perfil do *job* em questão. Mais especificamente, pode-se citar dois tipos de otimizações disponibilizadas por tais RIPs industriais: as de re-usabilidade e as de transparência.

As primeiras dizem respeito a um controle maior quanto ao gerenciamento da *cache* de cada RIP. Sabe-se que objetos re-utilizáveis no formato PDF são definidos para prevenir o re-processamento de dados que podem ser armazenados na memória de um RIP, à medida que

foram computados pela primeira vez. Entretanto, diversos objetos re-utilizáveis podem ser aplicados em um documento PDF, fazendo com que o resultado da rasterização daqueles mais antigos possa ser apagado da memória do RIP, devido à necessidade de tal fatia de memória para outro objeto re-utilizável recém encontrado no documento. Esta situação pode ser boa, no sentido que o objeto re-utilizável descartado não apareça mais no documento, ou ruim se ele for re-aplicado diversas outras vezes. No primeiro caso, nenhuma perda efetiva será obtida, pois como tal objeto não será mais re-utilizado, este pode ser apagado. No último caso, porém, será necessário o re-processamento deste objeto, perdendo a vantagem do recurso da re-usabilidade na sua íntegra. Desta maneira, alguns RIPs industriais possuem uma opção de indicar que existem muitos objetos sendo re-utilizados no contexto de um documento PDF. Assim, cada RIP irá priorizar o armazenamento do resultado destes objetos em sua *cache* e em algum meio de armazenamento auxiliar (como um disco ou uma memória maior), garantindo a re-utilização dos resultados. Entretanto, se o PDF de entrada apresentar muitos objetos re-utilizáveis que são utilizados uma única vez, as operações de armazenamento tornam-se custosas comprometendo o desempenho do processo de rasterização.

Entre os *jobs* existentes nas PSPs, não é incomum a existência de PDFs com uma mistura de objetos re-utilizáveis que são aplicados diversas vezes no documento e que são utilizados uma única vez. Neste contexto, torna-se difícil saber quando ativar a opção de otimização ou não. Devido à tal situação, deseja-se realizar uma quebra inteligente dos *jobs* para processá-los em paralelo, no sentido de que as páginas com objetos re-utilizáveis em comum sejam processadas pelo mesmo RIP com o uso otimização de re-usabilidade, enquanto aquelas sem objetos que estão sendo re-utilizados sejam processadas sem a otimização em questão. Desta maneira, acredita-se será possível superar o desempenho existente no cenário de rasterização. Para a realização desta quebra, a ferramenta de análise de perfil dos *jobs* terá de prover uma lista contendo quais objetos são aplicados em quais páginas, para que um processamento adicional consiga ditar uma quebra do *job* que agrupe os objetos re-utilizáveis de uma maneira inteligente. Assim, vale a pena mencionar que a ferramenta *PDF Profiler* já é capaz de prover esta informação quanto ao escopo de cada objeto re-utilizável (Seção 5.2), além disso a quebra dos *jobs* já é suportada por meio de uma lista de páginas por parte da aplicação *PDF Splitter* (Seção 5.3).

A outra otimização existente no contexto dos RIPs industriais é relativa à transparência. Como explicitado anteriormente, para que um RIP seja capaz de compor as cores de um objeto transparente é necessário um processamento sobre todas as cores que o objeto em questão está sobreposto, como as de páginas e outros objetos. Desta forma, alguns RIPs industriais possuem a possibilidade de indicar que um documento PDF contém diversos objetos transparente, aplicando, então, um processo otimizado para a rasterização destes objetos. Novamente, caso o documento PDF possua diversos objetos opacos juntamente àqueles transparentes, o desempenho da rasterização será comprometido. Isto se deve ao fato de que o processamento de objetos opacos com esta otimização ativada acarreta um custo adicional ao processamento ao invés de aumentar o seu desempenho. Neste sentido, sabe-se que o *PDF Profiler* pode fornecer

uma lista de quais são as páginas que contêm objetos transparentes, assim como aquelas que não possuem nenhum. Frente a isso, poderia-se quebrar os *jobs* de uma maneira inteligente, agrupando as páginas que possuem transparência em um ou mais conjuntos para transmití-los a RIPs com a otimização de transparência ligada. Por outro lado, as páginas sem transparência seriam enviadas aos RIPs sem a otimização de transparência.

Por fim, um outro estudo que deverá ser realizado diz respeito ao estabelecimento de métricas para aqueles tipos de objetos gráficos não considerados no trabalho atual. Estes são: *shading pattern objects*, *inline image objects* e *path objects*. Apesar destes objetos não apresentarem-se comumente entre os *jobs*, com a análise destes teria-se um conjunto de dados completo para a análise de perfil de quaisquer tipos de *jobs*.

Referências

[1] PURVIS, L. et al. Creating Personalized Documents: an Optimization Approach. In: *DocEng '03: Proceedings of the 2003 ACM Symposium on Document Engineering*. Grenoble, France: ACM, 2003. p. 68–77.

[2] NUNES, T. et al. High Performance XSL-FO Rendering for Variable Data Printing. In: *SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing*. Dijon, France: ACM, 2006. p. 811–817.

[3] Adobe Systems. *PDF Reference*. 4th. ed. San Jose, USA, 2003.

[4] 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. p. 1–14.

[5] NUNES, T. et al. An Improved Parallel XSL-FO Rendering for Personalized Documents. In: *14th Euro PVM/MPI - European PVM/MPI Users Group Meeting*. Paris, France: Springer Berlin / Heidelberg, 2007. p. 56–63.

[6] GOODMAN, T. *Case Study: Digital Publishing at the Olympic Games*. Sydney, Australia: Open Publish, 2001.

[7] OPPEN, D. C. Prettyprinting. *ACM Transactions on Programming Languages and Systems*, ACM, New York, USA, v. 2, n. 4, p. 465–483, October 1980.

[8] GIANNETTI, F. A Multi-format Variable Data Template Wrapper Extending Podis PPML-T Standard. In: *DocEng '07: Proceedings of the 2007 ACM symposium on Document engineering*. Winnipeg, Canada: ACM, 2007. p. 37–43.

[9] Adobe Systems. *PostScript Language Reference Manual*. 2nd. ed. San Jose, USA, 1990.

[10] COMPANY, H. P. *PCL 5 Developer's Guide*. 3rd. ed. Palo Alto, USA, 1991.

[11] PENNEBAKER, W. B.; MITCHELL, J. L. *JPEG Still Image Data Compression Standard*. Norwell, USA: Kluwer Academic Publishers, 1992. 638 p.

[12] ZIV, J.; LEMPEL, A. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on Information Theory*, IEEE Computer Society Press, Los Alamitos, USA, v. 23, n. 3, p. 337–343, May 1977.

[13] WELCH, T. A. A Technique for High-Performance Data Compression. *Computer*, IEEE Computer Society Press, Los Alamitos, USA, v. 17, n. 6, p. 8–19, June 1984.

[14] KRUATRACHUE, B.; LEWIS, T. Grain Size Determination for Parallel Processing. *IEEE Software*, IEEE Computer Society Press, Los Alamitos, USA, v. 5, n. 1, p. 23–32, January 1988.

[15] ZOMAYA, A. Y. H. (Ed.). *Parallel and Distributed Computing Handbook*. New York, USA: McGraw-Hill, Incoraporated, 1996. 1199 p.

[16] GRAHAM, R. L. Bounds for Certain Multiprocessing Anomalies. *Bell System Technical Journal*, ACM, New York, USA, v. 45, n. 9, p. 1563–1581, November 1966.

[17] BRUCKER, P. *Scheduling Algorithms*. Secaucus, USA: Springer-Verlag New York, Inc., 2001. 371 p.

[18] PINEDO, M. *Scheduling: Theory, Algorithms and Systems*. New York, USA: Prentice Hall, 2008. 678 p.

[19] LEUNG, J. (Ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, USA: CRC Press, Inc., 2004. 1120 p.

[20] GAREY, M. R.; JOHNSON, D. S. Strong NP-Completeness Results: Motivation, Examples, and Implications. *J. ACM*, ACM, New York, USA, v. 25, n. 3, p. 499–508, July 1978.

[21] SLEATOR, D. D.; TARJAN, R. E. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, ACM, New York, USA, v. 28, n. 2, p. 202–208, February 1985.

[22] KARLIN, A. R. et al. Competitive Snoopy Caching. *Algorithmica*, Springer New York, New York, USA, v. 3, n. 3, p. 79–119, November 1988.

[23] GRAHAM, R. L. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, Society for Industrial and Applied Mathematics, Philadelphia, USA, v. 17, n. 2, p. 416–429, March 1969.

[24] COFFMAN, E. G.; GAREY, M. R.; JOHNSON, D. S. *SIAM Journal of Computing*, Philadelphia, USA, n. 7, p. 1–17, February.

[25] DELL'AMICO, M.; MARTELLO, S. Optimal Scheduling of Tasks on Identical Parallel Processors. *ORSA Journal on Computing*, Spring, Ottawa, Canada, v. 7, n. 2, p. 191–200, January 1995.

[26] FRIESEN, D. K. Tighter Bounds for the Multifit Processor Scheduling Algorithm. *SIAM Journal of Computing*, Society for Industrial and Applied Mathematics, Philadelphia, USA, v. 13, n. 1, p. 170–181, February 1984.

[27] HOCHBAUM, D. S.; SHMOYS, D. B. Using Dual Approximation Algorithms for Scheduling Problems Theoretical and Practical Results. *J. ACM*, ACM, New York, USA, v. 34, n. 1, p. 144–162, January 1987.

[28] JOHNSON, D. S. et al. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM Journal on Computing*, SIAM, New York, USA, v. 3, n. 4, p. 299–325, December 1974.

[29] BOOK, R. V. Reducibility Among Combinatorial Problems. *The Journal of Symbolic Logic*, Association for Symbolic Logic, Poughkeepsie, USA, v. 40, n. 4, p. 618–619, December 1975.

[30] ImageMagick Home Page. Disponível em: http://www.imagemagick.org. Acesso em: 25 jun. 2008.

[31] PDFBox Home Page. Disponível em: http://www.pdfbox.org. Acesso em: 20 abr. 2008.

[32] GETOV, V.; HUMMEL, S. F.; MINTCHEV, S. High-performance Parallel Programming in Java: Exploiting Native Libraries. *Concurrency: Practice and Experience*, ACM, New York, USA, v. 10, n. 11, p. 863–872, December 1998.

[33] MPJ-Express Home Page. Disponível em: http://mpj-express.org. Acesso em: 10 fev. 2008.

[34] SNIR, M. et al. *MPI: The Complete Reference*. Cambridge, USA: MIT Press, 1996. 450 p.

[35] MPICH Home Page. Disponível em: http://www-unix.mcs.anl.gov/mpi/mpich1. Acesso em: 15 jan. 2008.

[36] LAM/MPI Home Page. Disponível em: http://www.lam-mpi.org. Acesso em: 16 jan. 2008.

APÊNDICE A – Lista de fontes utilizadas para a definição das métricas de texto separadas por grupo

Tabela 11 – Lista de fontes testadas nas métricas propostas.

(a) **Grupo leve**

ACaslon Pro Bold	Arno Pro Smbd Display	Giddyup Std	Nueva Std Cond Italic
ACaslon Pro Bold Italic	Arno Pro Smbd Italic	Gujarati MT	OCRA std
ACaslon Pro Italic	Arno Pro Smbd Italic Caption	Gujarati MT Bold	Orator Std
ACaslon Regular	Arno Pro Smbd Italic Display	Gurmukhi MT	Plantagenet Cherokee
ACaslon Pro Semibold	Arno Pro Smbd Italic Subhead	Hobo Std	Poplar
ACaslon Pro Semibold Italic	Arno Pro Smbd Sm Text	Impact	Prestige Elite Std
AGaramond Pro Bold	Arno Pro Smbd Subhead	Kaliasa Regular	Rosewood Std Regular
AGaramondPro Bold Italic	Arno Pro Sm Text	Kokonor Regular	Stencil Std
AGramond Pro Italic	Arno Pro Subhead	Letter Gothic Std Bold	Tahoma
AGaramond Pro Regular	Bell Gothic Std Black	Letter Gothic Std Bold Slanted	Tahoma Bold
Andale Mono	Bell Gothic Std Bold	Letter Gothic Std Slanted	Tekton Pro Bold
Arial	Birch Std	Letter Gotchi Std	Tekton Pro Bold Cond
Arial Bold	Blackoak Std	Lithos Pro Black	Tekton Pro Bold Ext
Arial Bold Italic	Brush Script Std	Lithos Pro Regular	Tekton Pro Bold Oblique
Arial Italic	Chalkboard	Mesquite Std	Times New Roman
Arial Black	Chalkboard Bold	Microsoft Sans Serif	Times New Roman Bold
Arial Narrow	Chaparral Pro Bold	Minion Pro Bold	Times New Roman Bold Italic
Arial Narrow Bold	Chaparral Pro Bold Italic	Minion Pro Bold Cn	Times New Roman Italic
Arial Narrow Bold Italic	Chaparral Pro Italic	Minion Pro Bold Cn Italic	Trajan Pro Bold
Arial Narrow Italic	Charlemagn Std Bold	Minion Pro Bold Italic	Trajan Pro Regular
Arial Rounded MT Bold	Comic Sans MS	Minion Pro Italic	Trebuchet MS
Arial Unicode MS	Comic Sans MS Bold	Minion Pro Medium	Trebuchet MS Bold
Arno Pro Bold	Cooper Black Std Italic	Minion Pro Medium Italic	Trebuchet MS Bold Italic
Arno Pro Bold Caption	Cooper Black Std	Minion Pro BoldRegular	Trebuchet MS Italic
Arno Pro Bold Display	Courier New	Minion Pro Bold Semibold	Verdana
Arno Pro Bold Italic	Courier New Bold	Minion Pro Bold Semibold Italic	Verdana Bold
Arno Pro Bold Italic Caption	Courier New Bold Italic	Mshtakan	Verdana Bold Italic
Arno Pro Bold Italic Display	Courier New Italic	Mshtakan Bold	Verdana Italic
Arno Pro Bold Italic Sm Text	Deco Type Naskh	Mshtakan Bold Oblique	Webdings
Arno Pro Bold Italic Subhead	Devanagari MT	Mshtakan Oblique	Wingdings
Arno Pro Bold Sm Text	Devanagari MT Bold	Myriad Pro Bold	Wingdings 2
Arno Pro Bold Subhead	Eccentric Std	Myriad Pro Bold Cond	Wingdings 3
Arno Pro Caption	Euphemia UCAS	Myriad Pro Bold Cond Italic	
Arno Pro Display	Euphemia UCAS Bold	Myriad Pro Bold Italic	
Arno Pro Italic	Euphemia UCAS Italic	Myriad Pro Cond	
Arno Pro Italic Caption	Garamond Premiere Pro Italic	Myriad Pro Cond Italic	
Arno Pro Italic Display	Garamond Premiere Pro Smbd	Myriad Pro Italic	
Arno Pro Italic Sm Text	Garamond Premiere Pro Smbd Italic	Myriad Pro Regular	
Arno Pro Italic Subhead	Garamond Premiere Pro	Myriad Pro Semibold	
Arno Pro Light Display	Georgia	Myriad Pro Semibold Italic	
Arno Pro Regular	Georgia Bold	Nueva Std Bold Cond	
Arno Pro Smbd	Georgia Bold Italic	Nueva Std Bold Cond Italic	
Arno Pro Smbd Caption	Georgia Italic	Nueva Std Cond	

(b) **Grupo médio**

Hira Kaku Std W8	Koz Go Pro Regular
Hira Kaku Std N W8	Koz Min Pro Bold
Koz Go Pro Bold	Koz Min Pro Extra Light
Koz Go Pro Extra Light	Koz Min Pro Heavy
Koz Go Pro Heavy	Koz Min Pro Light
Koz Go Pro Light	Koz Min Pro Medium
Koz Go Pro Medium	Koz Min Pro Regular

(c) **Grupo pesado**

Bickham Script Pro Bold	Hira Min Pro W3
Bickham Script Pro Regular	Hira Min Pro W6
Bickham Script Pro Semibold	
Hira Kaku Pro W3	
Hira Kaku Pro W6	
Hira Maru Pro W4	
Hira Maru Pro N W4	