Comput Sci Res Dev (2012) 27:147–166
DOI 10.1007/s00450-010-0134-0

SPECIAL ISSUE PAPER

Job profiling and queue management in high performance printing

Luiz Gustavo Fernandes · Thiago Nunes · Mariana Kolberg · Fabio Giannetti · Rafael Nemetz · Alexis Cabeda

Published online: 29 September 2010

Abstract Digital presses have consistently improved their speed in the past ten years. Meanwhile, the need for document personalization and customization has increased. As a consequence of these two facts, the traditional RIP (Raster Image Processing) process has become a highly demanding computational step in the print workflow. Print Service Providers (PSP) are now using multiple RIP engines and parallelization strategies to speed up the whole ripping process which is currently based on a per-page basis. Nevertheless, these strategies are not optimized in terms of ensuring the best Return On Investment (ROI) for the RIP engines. Depending on the input document jobs characteristics, the ripping step may not achieve the print-engine speed creating a unwanted bottleneck. The aim of this paper is to present a way to improve the ROI of PSPs proposing a profiling strategy which enables the optimal usage of RIPs for specific jobs features ensuring that jobs are always consumed at least at engine speed. The profiling strategy is based on a per-page analysis of input Portable Document Format (PDF) jobs identifying their key components. This work introduces a PDF Profiler tool aimed at extracting information from jobs and some metrics to predict a job ripping cost based on its profile. This information is extremely useful to rasterize jobs in a clever way. The computational cost estimated using the information extracted by the PDF Profiler and the proposed metrics is useful for the print jobs queue management to improve the allocated RIPs load balance, resulting in a higher throughput for the ripping step. Experiments have been carried out in order to evaluate the PDF Profiler, the proposed metrics and their impact in the print jobs queue management.

Keywords Digital printing · Raster image processing · Job profiling · PDF · Load balance · Print queue · Parallel processing

L.G. Fernandes (✉)
GMAP–PPGCC–PUCRS, Av. Ipiranga, 6681 – Pr. 32, Porto Alegre, Brazil
e-mail: luiz.fernandes@pucrs.br

T. Nunes
ThoughtWorks, Porto Alegre, Brazil
e-mail: thiago.nunes@pucrs.br

M. Kolberg
GMAP–ULBRA, Porto Alegre, Brazil
e-mail: marianakolberg@gmail.com

F. Giannetti
HP Laboratories, Palo Alto, USA
e-mail: fabio.giannetti@hp.com

R. Nemetz
GMAP–PUCRS, Porto Alegre, Brazil
e-mail: rafael.nemetz@pucrs.br

A. Cabeda
HP Brazil R & D, Porto Alegre, Brazil
e-mail: alexis.cabeda@hp.com

1 Introduction

The introduction of Digital Presses has opened the publishing market to new grounds. The increased speed of printing and the removal of the plate-setting stage has enabled publishers to reduce the size of their runs. This is a first step towards an emerging trend of personalization and customization. In the past, documents were produced for a large number of recipients and once they were transformed to the plates, the presses could go at full speed. Personalization was not usual at this time. In the few cases it was applied, the personalized documents should be printed on separate devices.

With the advent of Digital Presses, automated procedures for document creation and transformation have become necessary, in order to fulfil the personalization demand. A new discipline—Variable Data Printing (VDP) [16]—has been introduced, providing several techniques, technologies, concepts and patterns to enable the creation of documents with dynamic content. At the end of the whole print process, documents need to be ripped to be transformed to a bitmap. In the past, this was done only once for the entire print run. With the introduction of VDP, it is required for each page within the job. To give an order of magnitude in a conventional printing environment, a 10 page job sent to 1,000 recipients would require the 10 pages to be ripped once. In a similar scenario, in a VDP context, the ripping required can be as demanding as 10,000 pages.

The "every page is different" paradigm introduces scalability issues. Moreover, approximately every four years, Digital Presses increase their speed at a constant rate of two to three times faster their engine speed. This dramatic increase in performance has repercussions in the printing workflow. The processing speed of the Raster Image Processing engines (RIPs), storage devices and Local Area Network (LAN) throughput are stretched, becoming unsuitable for the task. In order to deal with this problem, more sophisticated workflows were proposed. Distributed and parallel systems have been deployed to ensure appropriate ripping and optimized press utilization. Nevertheless, systems can only be tuned for the "average" job. This means that many jobs fail to be ripped at engine speed, delaying or even stopping production. This is more evident in small and mid size PSPs for two reasons: (i) their level of investment is lower: resulting in less RIP engines; (ii) the nature of the jobs they produce have a wider spectrum and consequently less predictability.

This paper is structured as follows: Sect. 2 presents the motivations and objectives of this paper. Section 3 describes a PSP case study scenario introducing several kinds of input documents that will be used to validate the Profiler and the proposed metrics. Section 4 describes the work carried out in implementing a tool capable of identifying complexity for PDF jobs. It also presents a detailed set of information that can be retrieved, how they are utilized and correlated. The proposed metrics are presented in Sect. 5. Section 6 evaluates the proposed metrics precision when estimating the computational cost to rip a PDF job. Section 7 presents a case study to illustrate the impact of the proposed metrics in the printing jobs queue management and, finally, Sect. 8 describes the achievements of this work and discusses themes for future researches.

2 Motivations and objectives

As introduced earlier, ripping is a crucial step in the Digital Printing workflow. The increase in personalization has led to the utilization of Page Description Languages (PDL), such as the Portable Document Format (PDF) [1] to increase the usage of reusable objects. Among the new formats that have been introduced, the most successful is the PPML (Personalized Print Markup Language) [2]. PPML describes jobs containing personalized content and reusable components that can have a scope which spans from a page to the entire document.

The usage of these formats reduces the potential amount of content to be ripped. The success of these formats lies in the ability of the Job Producer to capture the reusability and of the Job Consumer to exploit them. Even in the case when both producer and consumer are fully exploiting the format, the issue of highly variable documents remains a problem. In this case, the support of the PPML or PDF with reusable objects is of little help since all the variable parts need to be ripped anyway.

While these documents may appear very specific, in reality they are very common. All the photo printing applications are a very simple case of "every page is different" application on a per recipient basis.

The ripping time and throughput have repercussions for the entire job capacity that a PSP can sustain. In order to remain competitive, PSPs are forced to extract all the production capability available from their investment as profit margins decrease. This motivates investigating more efficient ways to consume jobs, but also to predict the resources required in order to rip these jobs at engine speed. This could potentially help PSPs to achieve a better capacity planning and cost estimates.

The PSPs task is to fully employ their Digital Presses and other equipments. This literally translates into the number of jobs a PSP can produce in an hour/day. In order to model the problem space, it is considered that the PSP manages a queue of jobs to be processed for a given day. It is imperative that all the activities related to a document's preparation must be finished at engine speed since the printing machine speed is the physical bottleneck that cannot be avoided. Moreover, PSPs usually employ presses in parallel in order to increase the jobs consumption, this reduces the risk of press failure and maintenance. This clearly implies that the performance of the document pre-processing phase must increase proportionally in order to fully satisfy the combined engine speed.

However, none of the current ripping strategies (see Sect. 7) can guarantee the best scheduling and load balancing. They often result in resources either being under or over loaded and this directly impacts the overall RIP performance. One possible strategy to solve these issues is to

predict the ripping computational cost for each job. This solution would make it possible to allocate only the necessary resources on a per job basis.

Strategies based on high performance computing techniques were employed on previous works [12–14] to significantly improve the document composition task performance. It is our objective to employ similar techniques to the ripping task, improving its performance through the optimization of the current parallel RIP strategies.

In order to achieve this objective, the work here presented describes the necessary intermediate steps. First of all, consistent metrics are defined in order to measure the computational effort to rip a job. Moreover, a PDF Profiler tool [11] is introduced and used to gather all the information needed to apply the given metrics. This tool analyzes the PDF internal structure and provides information at the page/fragment level. As mentioned before, this is important to optimize and to guide the splitting process before the pages/fragments are sent to the RIPs. After that, a study exemplifies the possible performance improvements that can be achieved by applying the metrics obtained through the PDF Profiler tool to organize the printing jobs queue. A further aspect is to design, around the Profiler information, an Adaptive Job Router tool capable of segmenting a job into portions that are optimized for the RIP, and not merely by order, taking full advantage of the distributed strategies.

3 PSP scenario—input documents

As mentioned in the introduction, small and mid sized PSPs are more sensitive to the situation in which jobs fail to be ripped at engine speed: they have less RIP engines and the jobs they produce are more diversified making unappropriated the adoption of an "average" job strategy.

This section introduces a set of different types of input documents these PSPs may deal with. These documents will be further used in this paper to help the evaluation of the profiler performance and to validate the proposed ripping cost estimation metrics.

The jobs selected are distributed among 8 types of input documents, that are commonly used within the PSPs. Among these types, it should be noted that many of them present similar elements in their content. An example of each document type is shown in Fig. 1. Characteristics of the 8 types of documents are:

1. *Letter*: a letter contains one or more pages with a large amount of text and few pictures. These images are typically logos, stamps or signatures;
2. *Newsletter*: a newsletter differs from a standard letter because it usually contains several images. These images are used for advertisements and news on products and services. The number of pages in this case can vary depending on the message to be delivered to the customer;
3. *Cards*: business cards, containing background images and little text. A page of such document may have several cards or a single instance of them;
4. *Postcard*: postcards are used for the exhibition of a place, product or service. In this sense, texts and images are used to compose each page. In general, postcards contains two pages (front and back) and several can be printed on one page;
5. *Flyer*: these documents are used to present and advertise products, ideas, events, among others. Generally each document contains two pages usually printed on the front and back sides of a sheet of paper. Both images and texts are used in such kind of document;
6. *Brochure*: this document usually consists of a large number of pages rich in images and text. At the end of printing, the pages of a brochure document will be assembled in order to form a kind of booklet;
7. *Newspaper*: document containing information and news headlines. This document type can have one or more pages, each one containing texts and images;
8. *Poster*: this type of document has typically only a large size page. A poster can contain only images, only texts or a mix of both.

For our experiments, 20 customers jobs used in PSPs were chosen among these 8 types as input documents. These jobs not only have documents with text and images, but also documents with graphical objects, such as paths and shading patterns (introduced later in Sect. 4.1). The full list of jobs is presented in Table 1, highlighting their characteristics.

The selected set of documents presents a large variability in number of pages, text and images. It represents a wide range of different rip computational costs. This high variability reflects the reality of most small to mid sized PSPs where jobs come from various customers.

4 PDF profiler

The PDF Profiler has been developed to analyze a PDF document and to extract the relevant information needed to optimize the ripping phase. The tool is implemented using the PDFBox Java library [15] to navigate and find PDF elements. Some studies were performed analyzing the PDF format [3, 17] to overcome the existing issues of the ripping phase but none employed a profiling and "cost" metrics. The main objective of the PDF Profiler is to gather the necessary information about the input PDF documents which will be used to estimate their computational costs (Sect. 5).

This section first introduces an analysis of the PDF format and its main objects which will be manipulated by the PDF Profiler. After that, the profiler architecture is described and each one of its modules is explained. Finally, an evaluation of the profiler additional computational cost against the ripping process is introduced.

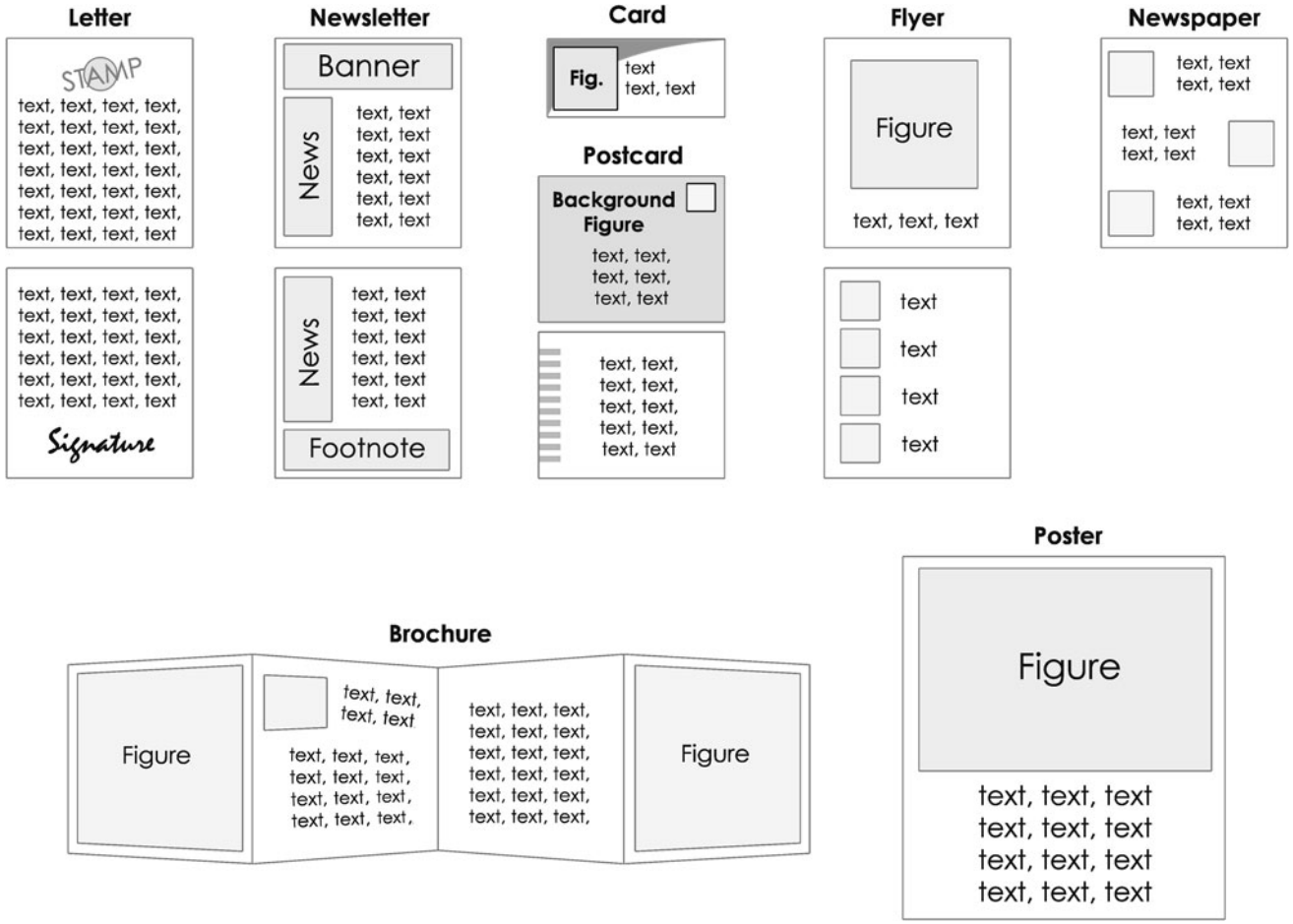


Fig. 1 Examples of the analyzed document types

4.1 PDF format objects

In an effort to identify metrics for predicting the computational cost of the PDF jobs, it is indispensable to better understand what are the PDF objects and how they influence the ripping of each job. Therefore, the PDF format was studied in detail in order to provide a better understanding of its objects and how they can be represented by such metrics.

The PDF format is a document description language that uses vector graphics to represent the content, providing several advantages to the PSPs, such as the document resolution independence. The PDF specification [1] has *specific commands* to define or use primitive types, for instance, integer and floating point numbers, strings, boolean values, etc. Furthermore, through the usage of these commands, it is also possible to define and to utilize the elements which will be painted on the pages of a document.

The objects can be classified into 5 distinct subtypes:

- *path objects*: represent arbitrary shapes, trajectories, regions and paths;
- *external objects* (*xObjects*): represent elements that can be reused. Three subtypes of xObjects are available:
  - *images xObjects* describe bitmap images, that use a pixel matrix in order to present the content;
  - *postscript xObjects* define the content based on PS commands. These objects are no longer used on the PDF format;
  - *groups* (*form*) *xObjects* define groups of graphic objects, used to set common properties to the objects within the group.
- *inline image objects*: define an image directly in the content stream of the PDF, which cannot be reused. This object has several limitations, of which the most relevant one refers to the image size;
- *shading pattern objects*: describe a geometrical shape that has its color defined by an arbitrary function;
- *text objects*: describe the text portions of a document. This includes text formatting and content characteristics, such as: font, size, space between characters, etc.

Moreover, the PDF format provides various ways to perform geometrical transformations (in order to scale, skew and rotate the objects) and the application of *transparency*

Table 1 Jobs characteristics

Job	Documents	Pages	Images	Pages with text
Brochure 1	9	108	162	108
Brochure 2	200	1000	1800	1000
Brochure 3	300	600	2700	600
Card 1	400	200	800	200
Card 2	40	40	40	40
Flyer 1	80	160	320	160
Flyer 2	100	200	400	200
Flyer 3	250	500	750	500
Flyer 4	75	150	150	150
Letter 1	130	390	130	390
Letter 2	400	800	800	800
Letter 3	90	450	0	450
Newsletter 1	220	440	3740	440
Newsletter 2	170	340	680	340
Newspaper 1	400	400	2000	400
Newspaper 2	150	900	2700	900
Postcard 1	500	250	4000	250
Postcard 2	20	40	120	40
Poster 1	500	500	2000	500
Poster 2	70	70	140	70

(opacity reduction). These features are obtained throughout the definition of the graphics state.

4.2 Profiler architecture

The PDF Profiler is capable of extracting a copious amount of information from the input document. This information comes from the number of objects and their related properties, the number of pages and their corresponding sizes. Therefore, this tool can provide an in-depth analysis of the PDF documents, if necessary. Clearly this has a computational cost which is directly proportional to the amount of information that will be extracted. Thus, it is important to find a compromise between the amount of information versus the speed of analysis. For such purpose, the PDF Profiler receives as input a XML (Extensible Markup Language) [4] file configuring the level of details of the profiling information returned. Figure 2 illustrates the architecture and components of the Profiler and the existing relationship between them.

The *GRestore* and *GSave* modules are directly related to the graphics state scope. In a PDF document, graphics states can be restored and saved (see Sect. 4). The last defined graphics state is the one that will be applied to the graphic objects. These two modules handle this control by using a stack, piling up the graphics states whenever they are saved and removing from the top the first one whenever there is a

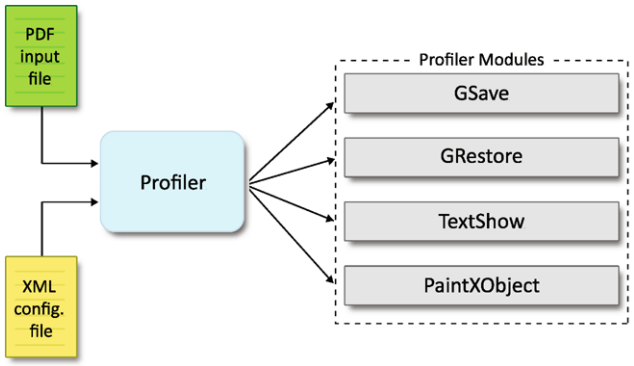


Fig. 2 PDF Profiler architecture

restore command. Thus, the top of the stack will always represent the current graphics state being applied to the graphics objects. With this kind of control, it is possible to check for every graphic object that it is being verified whether it is transparent or not.

The *TextShow* module evaluates if the current page includes text or transparent text. This module is instantiated whenever there is text being painted on the PDF document. Once created, it checks the current graphics status to verify if there is any transparency being applied. If it is the case, the current page is marked with the presence of transparent text.

The *Paint xObject* module deals with the image xObjects. Its main goals are (i) to check if the xObject is being reused or not, (ii) to verify if it is transparent or not and (iii) to retrieve the objects area. In order to accomplish the first function, this module contains a map with the already painted images. Thus, whenever a new image is painted, it first checks this map to verify if the image is being reused or not. In the case where it is the images first instance, that image is inserted on the map and the xObject is marked as non-reused. Otherwise, the object is marked as reused to verify if transparency is being applied or not, the graphics state is recovered from the stack and evaluated.

Finally, the PDF Profiler recovers all the properties of the image xObject, through the xObjects dictionary using the name of the object (which is mandatory in the painting command). Then, the tool picks the width and height of the image from the recovered properties in order to properly calculate the area. With all the tasks completed, this module adds the computed value to the right accumulator: transparent reusable images area, non-transparent reusable images area, transparent non-reusable images area or either non-transparent non-reusable images area.

4.3 Profiling cost

This section introduces an evaluation of the profiling time against the total ripping time. Table 2 shows the times

Table 2 Jobs profiling cost

Job	Ripping time (s)	Profiling time (s)	Percentage (%)
Brochure 1	41.844	4.08	9.75
Brochure 2	1907.12	5.86	0.31
Brochure 3	1399.97	4.64	0.33
Card 1	830.23	18.89	2.28
Card 2	13.402	0.92	6.86
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
Letter 1	841.30	2.26	0.27
Letter 2	1703.20	10.63	0.62
Letter 3	874.15	2.46	0.28
Newsletter 1	1614.08	14.49	0.90
Newsletter 2	879.18	7.42	0.84
Newspaper 1	1358.73	3.04	0.22
Newspaper 2	1830.28	3.3	0.18
Postcard 1	959.20	1.56	0.16
Postcard 2	16.47	0.96	5.83
Poster 1	877.05	10.6	1.21
Poster 2	50.004	1.54	3.08

(in seconds) obtained for the profile analysis of different jobs, as well as the ripping time for each of them. The last column of the table indicates the percentage the profiling time represents when compared to the total ripping time.

Looking closely at the figures presented in Table 2, it is possible to notice that the profiling time for some jobs is much higher than others. This fact is directly related to the existing number of objects in the PDF file itself (not just the graphics). The higher the number of occurrences of these objects, the larger is the interpretation time to convert the file into the PDFBox library format. Two other factors impacting the profiling time can be mentioned: (i) the size of the PDF file (ii) the amount of images, text and pages for each job.

In most of the cases, the job profile analysis time does not exceed 2.00% of the total ripping time. These results confirm the viability of using the Profiler tool. However, for some jobs like Poster 2, Postcard 2, Cards 2 and Brochure 1 the profiling time represents higher percentages of the total ripping time (approximately 3%, 6%, 7% and 10% respectively). The reason for this is the fact that these jobs are ripped in a short time, suffering proportionally much more the influence of I/O operations during the profiling time.

5 Metrics

To estimate the computational ripping cost of a PDF job, this section proposes several metrics. These metrics are related to a set of PDF objects and features which are relevant for the ripping process. A RIP is responsible for generating a *bitmap* image for each page in the input job. In order to compose the output images, the RIP must interpret each one of the objects within the PDF and, paint them on the corresponding image.

Several metrics are proposed and experiments were carried out to verify the hypotheses. These results were obtained though an average of 20 executions, with the usage of the open-source RIP converter from ImageMagick [6]. Furthermore, these results were normalized (to a scale between 0 and 1), generating a *relevance factor* for each test case. Through these experiments it is possible to establish a formula to calculate the computational cost associated with each of the analyzed feature. It is important to point out that the metrics represent an *approximation* of these costs and *not* the exact values. The estimated cost obtained by the application of these metrics to the information extracted from a PDF job by the PDF Profiler gives an approximation of the *total computational cost* for each job. The most representative elements to formulate metrics to evaluate a PDF job are now presented.

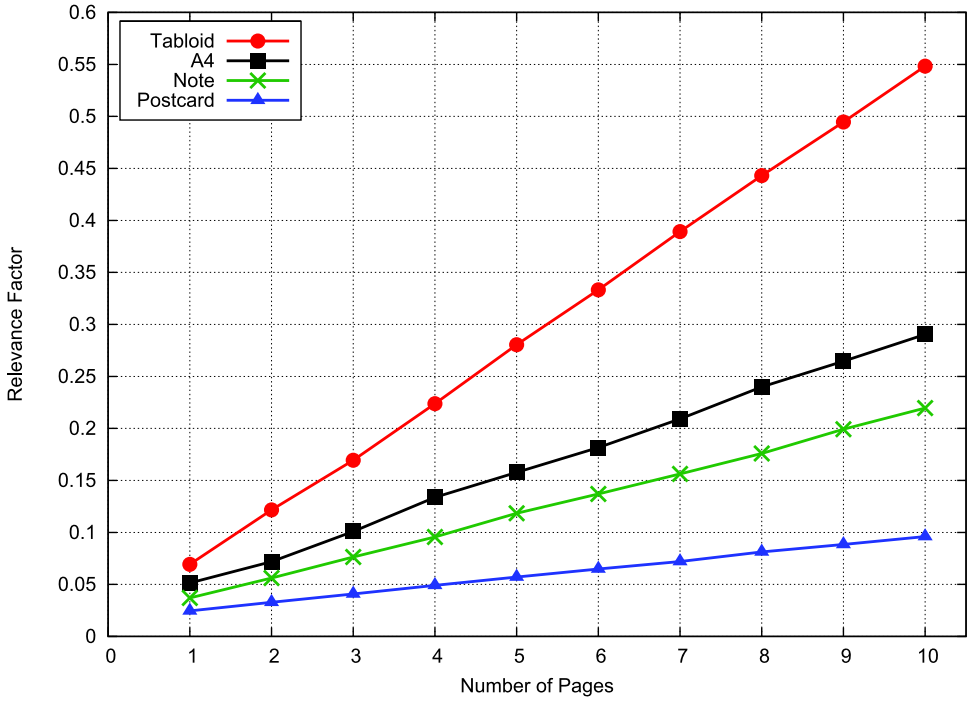
5.1 Pages

The number of pages in a PDF document is directly related to the number of images generated at the end of the ripping process. Therefore, the higher the number of existing pages, the higher will be the quantity of I/O operations that must be performed by the RIPs. The cost of blank pages (without any graphical objects) is now analyzed in order to verify their impact on the overall ripping process. Distinct page sizes have been considered (with their respective sizes denoted as "width × height" in pixels): *Postcard* (283 × 416), *Note* (540 × 720), A4 (595 × 842) and *Tabloid* (792 × 1224). Figure 3 presents the weighted costs obtained through the addition of more pages in a PDF document.

To establish the ripping computational cost for a single page, the weighted cost increment is analyzed as more pages are added. Note that the higher are the page dimensions (or the page area), the higher is the increment of the weighted cost. This occurs because pages with larger areas generate a larger output image resulting in more I/O operations.

The cost of a page is denoted as $cPag_{tPag}$, where *tPag* represents the dimensions of the corresponding page. The approximated (average) costs obtained for the experiments were: $cPag_{283\times416} = 0.007$, $cPag_{540\times720} = 0.019$, $cPag_{595\times842} = 0.026$, $cPag_{792\times1224} = 0.053$. In order to establish the cost $cPag_{tPag}$ for any *tPag*, the smallest cost obtained in the experiments has been chosen: $cPag_{283\times416}$. Let $area_{tPag}$ be the area of a rectangle with dimensions defined by *tPag*. Therefore, the cost of a page with the size *tPag* will

Fig. 3 Blank pages



be obtained through the proportion $\frac{area_{tPag}}{area_{283\times416}}$ which will result in a factor that will be multiplied by the cost. This strategy is presented in (1) as follows:

$$cPag_{tPag} = \frac{area_{tPag}}{area_{283\times416}} * cPag_{283\times416}. \quad (1)$$

Considering the direct relation between page and the weighted cost increment, the above equation is capable of approximating the weighted cost. Therefore, the cost of all pages of a given document may be calculated throughout the sum of the cost of each page. Thus, as shown in (2), the cost of all pages of a PDF document is obtained through the cost of the sum of the pages sizes (denoted as *tPagTot*):

$$cPag_{tPagTot} = \frac{area_{tPagTot}}{area_{283\times416}} * cPag_{283\times416}. \quad (2)$$

5.2 Images

The following experiments take into account a PDF document containing only one A4 page and images with different sizes. The images are defined using xObjects due to its capability to encapsulate images with no size restrictions. The selected image sizes were (described as "width × height" in pixels): 1190 × 1684, 1785 × 2526, 2380 × 3368 and 2975 × 4210. The first chosen image size is twice the dimensions of an A4 sized page (595 × 842). Furthermore, distinct types of images were considered, such as (high resolution) photos and artistic images (with gradients and graphical effects), and reusability features for the xObjects.

5.2.1 *Non reusable images*

For the experiments presented on this section, each image object is used only once to avoid reusability. The following image objects aspects are analyzed: coverage (area) over the page, image dimensions, number of objects and the use of transparency.

The first experiment carried out is the page coverage. For this purpose, four test cases containing a single image were created (using the dimensions previously specified), varying its coverage percentage over the page from 1% up to 100%. Figure 4 illustrates the obtained results.

The results indicate that the coverage of an image object *is irrelevant* to the ripping process since increasing the coverage percentage does not result in a higher cost. However, the results also show that image dimensions *are* relevant for such process once larger images present higher weighted costs. The RIP must interpret the image content, convert it in the selected output format, scale it down and finally paint the image. All these operations are directly related to the image dimensions.

In order to complement the previous experiment, test cases were generated by varying the number of image objects on a single page from 1 to 8. Thus, it was possible to discover the cost associated to an image object. In Fig. 5, it is possible to verify that the weighted cost becomes higher as the number of objects within a page grows. The larger the

Fig. 4 Images coverage

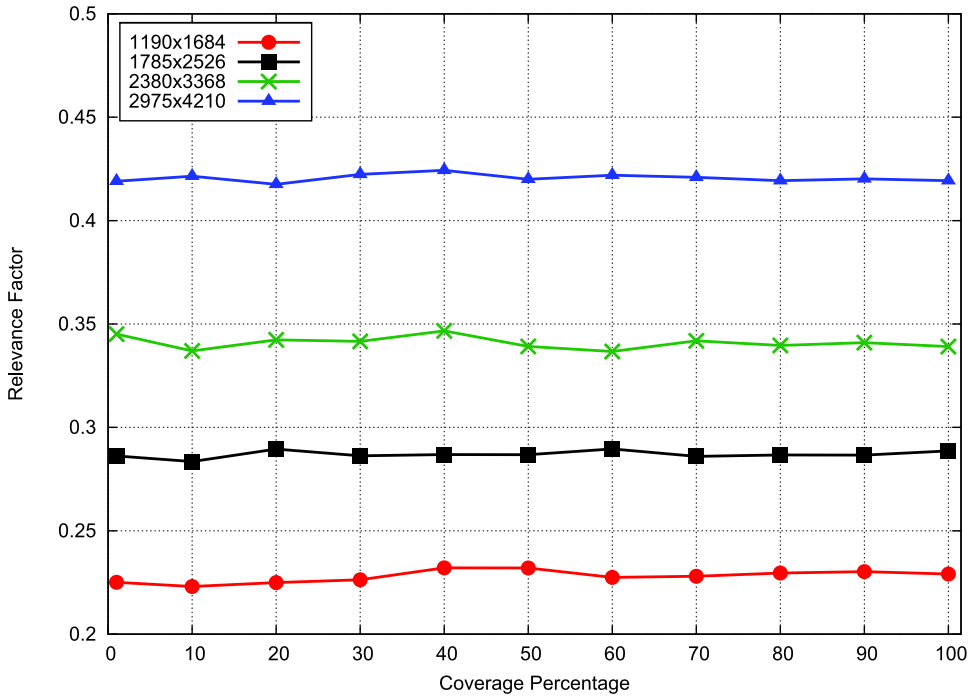
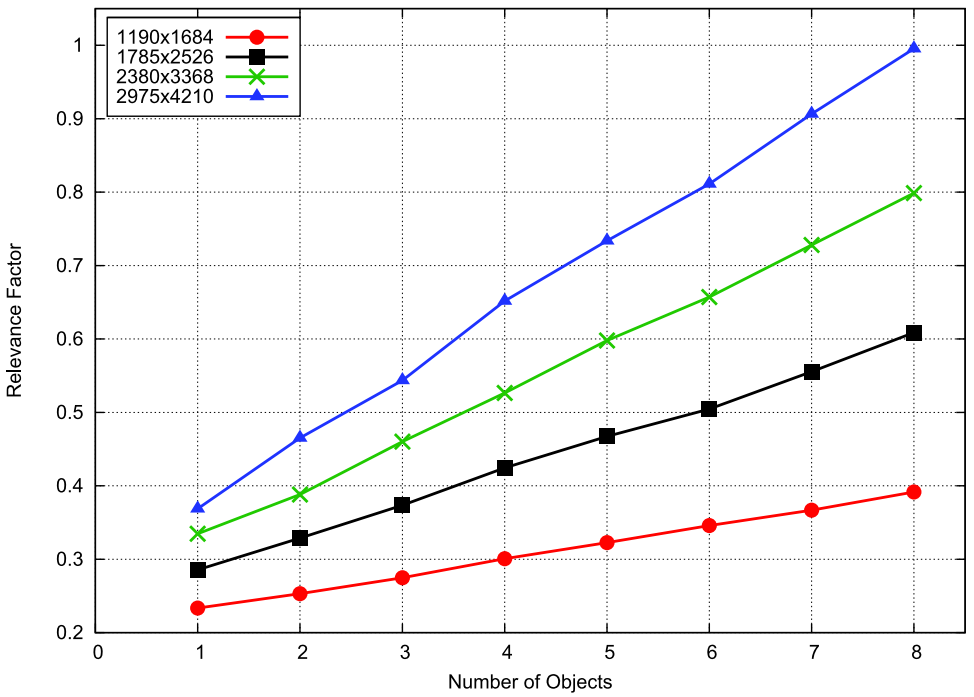


Fig. 5 Number of image objects



area of the inserted object, the higher the increment on the weighted cost.

The method previously established defines the cost of an image object, with size *tIm*. The cost of the image objects of a PDF can be obtained from the sum of the individual cost for each object. Thus, the total cost can be obtained through the cost of the sum of the *tIm* sizes of the objects (denoted as *tImTot*) as follows:

$$cIm_{tImTot} = \frac{area_{tImTot}}{area_{1190\times1684}} * cIm_{1190\times1684}. \quad (3)$$

The application of transparency over the graphic objects is the last aspect to be evaluated. In order to perform this analysis, experiments were carried out considering two dif-

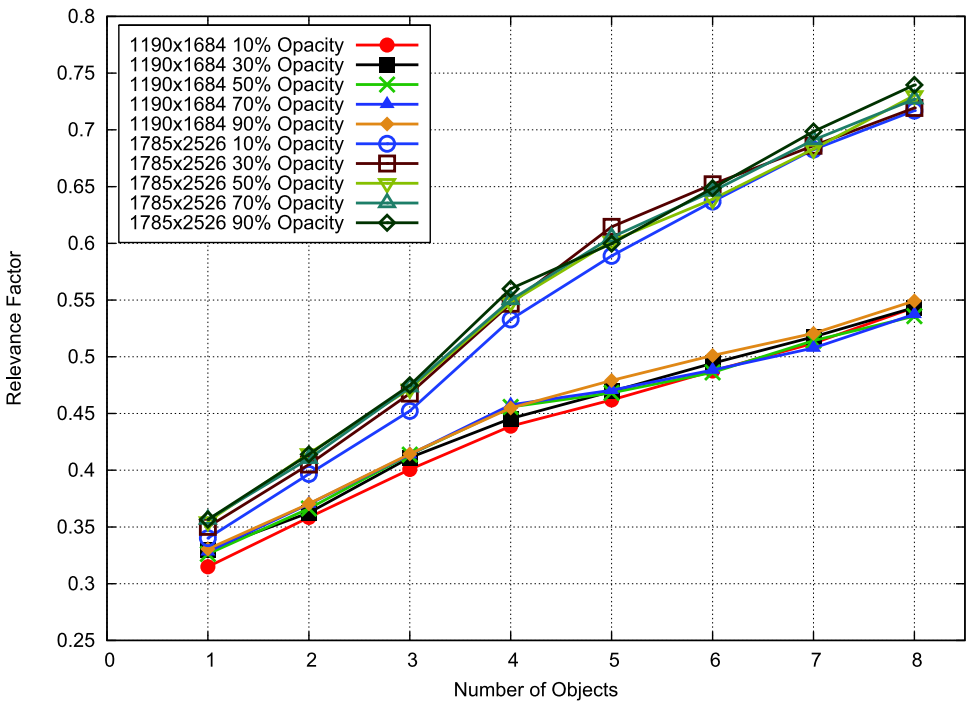


Fig. 6 Transparent image objects

ferent image sizes with 5 different levels of opacity each. Figure 6 shows the obtained results.

Comparing the results of experiments with dimensions 1190×1684 and 1785×2526 in Fig. 6 with the results presented in Fig. 5 for the same images, it is possible to observe that the application of transparency increases the weighted cost. On the other hand, it also can be noticed that the degree of opacity does not affect the computational cost of a given image. The additional cost associated to transparency is related to the composition of the color of transparent objects based on their color overlapping with page and other objects colors. Furthermore, it can be seen that the larger the area of the transparent object, the greater is the associated weighted cost. Applying the same strategy, based on the increment of the weighted cost, is possible to define the cost for computing a transparent image object as $cImT_{tImT}$. Thus, similarly to the image objects without transparency, this factor can be calculated as $\frac{area_{tImT}}{area_{1190\times1684}}$ and the total cost of transparent image objects can be obtained through the cost of these objects total size *tImTTot*, as follows:

$$cImT_{tImTTot} = \frac{area_{tImTTot}}{area_{1190\times1684}} * cImT_{1190\times1684}. \quad (4)$$

5.2.2 Reusable images

The test cases presented in this section are based on a single image reused as many times as necessary (implemented as a single instance of an *image xObject* previously defined). This makes it possible to analyze the impact of the reusability feature on the ripping computational cost. The obtained results are presented on Fig. 7.

As it can be noticed, the results presented follow the same behavior of the ones without reusability, however the cost added as more objects are inserted is lower. Let the cost of reusable objects be denoted as cRe_{tRe}. Considering that once again there is a relation between the area of the added object with its associated cost, the total computational cost of objects can be computed by the cost of the object sizes sum. Therefore, *tReTot* refers to the total size of the reused objects. Equation (5) demonstrates how the cost for such objects is computed:

$$cRe_{tReTot} = \frac{area_{tReTot}}{area_{1190\times1684}} * cRe_{1190\times1684}. \quad (5)$$

The same way as for the objects without reusability, some experiments were carried out considering the application of transparency for the reusable objects. Figure 8 illustrates the obtained results.

The presented graph demonstrates that for the reusable objects with transparency the same increase of the weighted cost could be observed. Applying the same methodology as before, the total cost of the transparent reusable objects can be computed though the cost of the sum of their sizes (*tReTTot*), as demonstrated on (6) as follows:

$$cReT_{tReTTot} = \frac{area_{tReTTot}}{area_{1190\times1684}} * cReT_{1190\times1684}. \quad (6)$$

Fig. 7 Reusable image objects

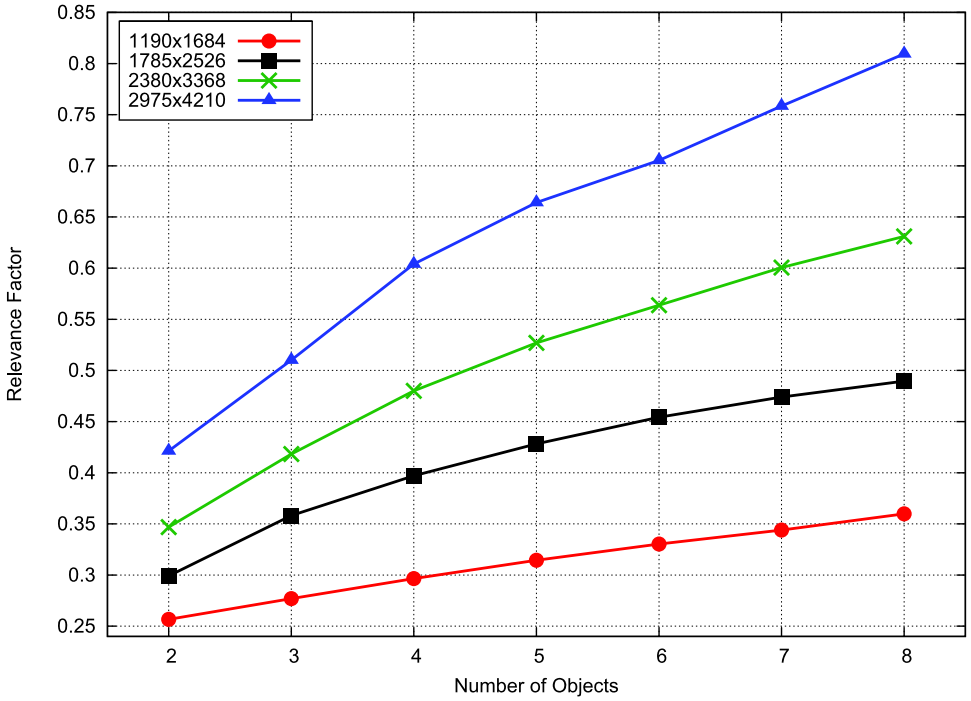
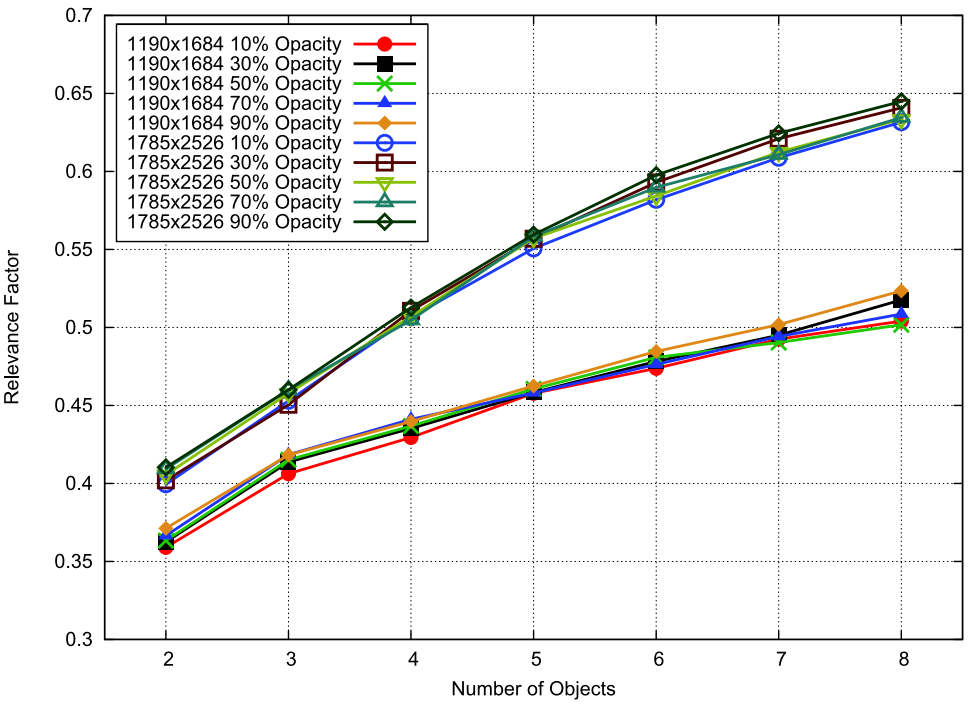


Fig. 8 Transparent reusable image objects

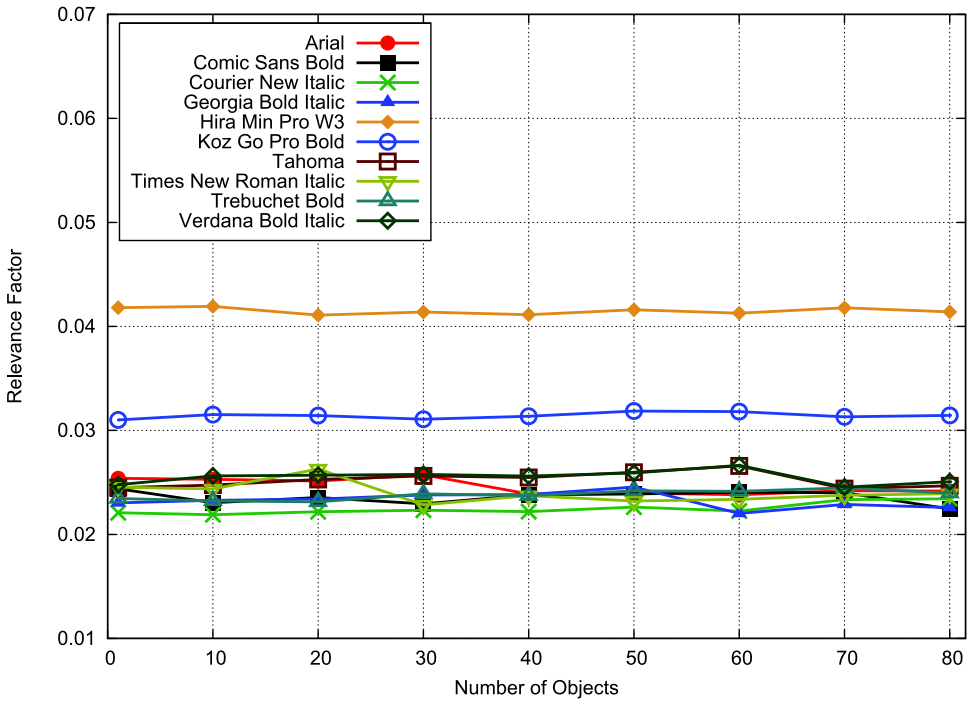


5.3 Texts

Document personalization usually involves text change and re-flow. In PDF, the text content is represented using text objects. The variability of text objects is represented through the amount of contained text, font sizes and the application of transparency. Tests files have been created using A4 page(s) and 70 individual fonts, applying bold, italic or both styles. Overall, about 200 tests for each feature have been analyzed. In order to illustrate the obtained results, 10 fonts (representing the common behavior among all test cases) have been chosen.

The first aspect considered was the impact on the ripping cost when the amount of text in a PDF document is

Fig. 9 Text amount



increased. Figure 9 demonstrates the obtained results. It is worth mentioning that for these experiments a phrase corresponds to a text object.

Although the presence of text has a higher influence on the ripping time compared to the blank page test cases (as illustrated in Fig. 3), the amount of text objects does not indicate any substantial impact on the ripping computational cost. This can be observed in Fig. 9, in which it can be seen that the increase in text objects does not show any change in performance for the 10 test cases. Additional experiments varying the font sizes (from 10 to 100 points) indicate that the font size does not affects the ripping time since the weighted cost remains the same for all font sizes tested.

Another finding is the existence of three different font groups in terms of computational cost. The first group presents the highest computational cost and includes 5% of all the fonts tested (e.g., *Hira Min Pro W3*). The second group represents 9% of the tested variations (such as *Koz Go Pro Bold*) with a medium influence on the weighted cost. Finally, the third group is composed of 86% of the font types and represents the smallest influence on the weighted cost.

These differences among the fonts weighted costs is related to each font design and amount of details. Documents may present several font variations and therefore the analysis of font complexity could compromise the metric definition process. In order to simplify this scenario, a constant cost for each page will be assigned when the presence of text is detected, independently of the number of text objects and fonts. This cost will be obtained through an overall average of the individual costs for each of the three previously mentioned font groups. Thus, a cost $cTxt = txt$ is going to be computed in the presence of text on a page of the PDF document. In order to find out the approximated value of *txt* for the different groups, experiments using a fixed number of text objects and a variable number of pages (from 1 to 5) were generated. These results are shown in Fig. 10.

To evaluate the text influence on PDF ripping process, the page cost was discarded. Analyzing the remaining cost, it is clear that it increases as more pages with texts are ripped. Dividing this partial cost by the number of pages, the 86% group presented a per page cost of 0.042, the 9% and 5% groups indicate a higher cost per page of 0.061 and 0.088 respectively.

Using these results, it is possible to obtain a single cost for the presence of text using a weighted average: $cTxt = 0.012$. Furthermore, the total cost of text (denoted as *cTxtTot*) for *np* pages can be calculated using (7) as follows:

$$cTxtTot = \sum_{i=1}^{np} txt. \tag{7}$$

The next step was to evaluate the impact of transparency over text objects. Different opacity values were tested using a fixed number of objects (80 objects). The obtained results are presented in Fig. 11. This graphic indicates no relation between the weighted cost and the applied opacity value, presenting a similar behavior to the image objects. On the other hand, transparent text does increase the weighted cost. In order to evaluate which is the impact of transparency over text objects, tests were carried out varying the number of

Fig. 10 Texts and pages

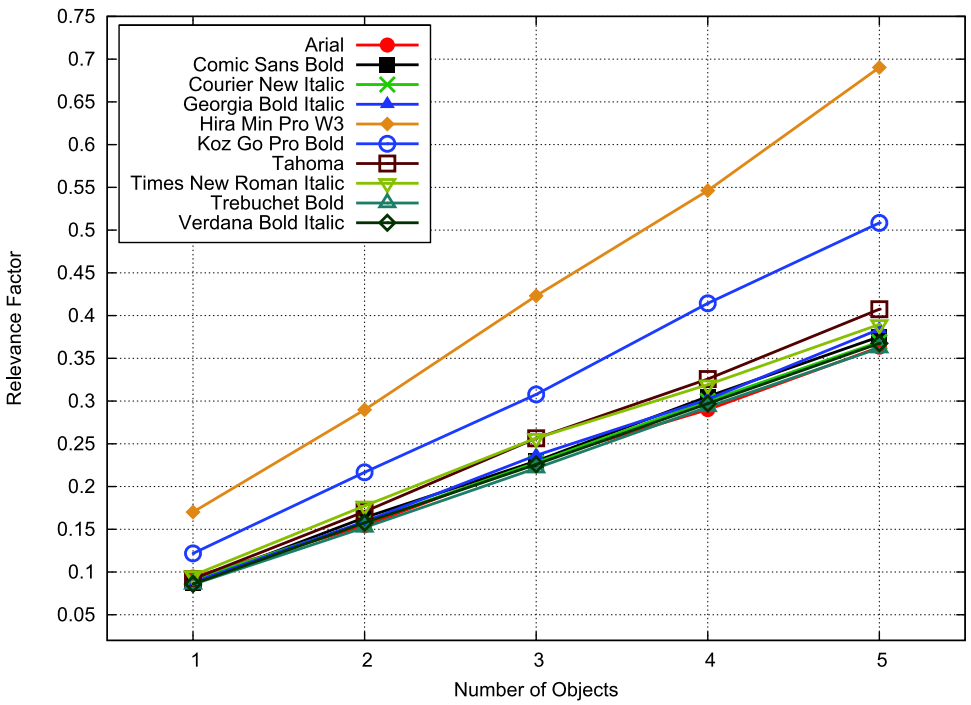
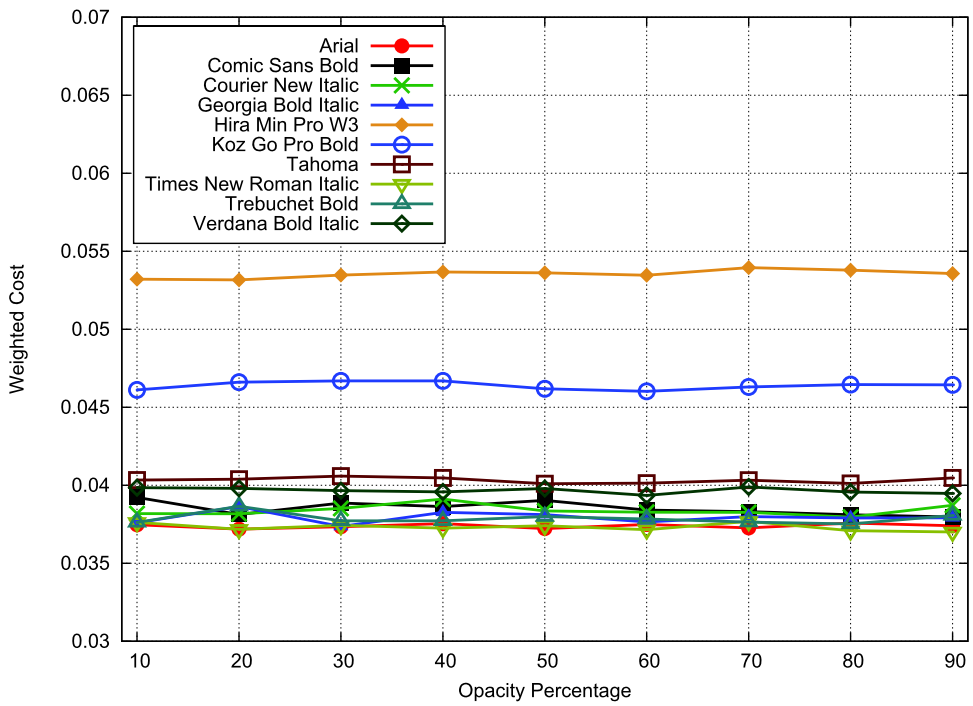


Fig. 11 Text transparency



pages with transparent text. These results are illustrated in Fig. 12.

For the three established font groups, the following per page costs were obtained: the 86% group transparency cost was 0.092 while the 9% and 5% groups indicate a higher cost of 0.127 and 0.154 respectively. Comparing these results to the one without transparency, it can be evinced that the use of transparency doubles ($incTxt = 2$) the weighted cost. Thus, the texts cost with and without transparency can be computed as follows:

$$cTxtTot = \sum_{i=1}^{np} txt + (tr_i * (incTxt - 1) * txt). \quad (8)$$

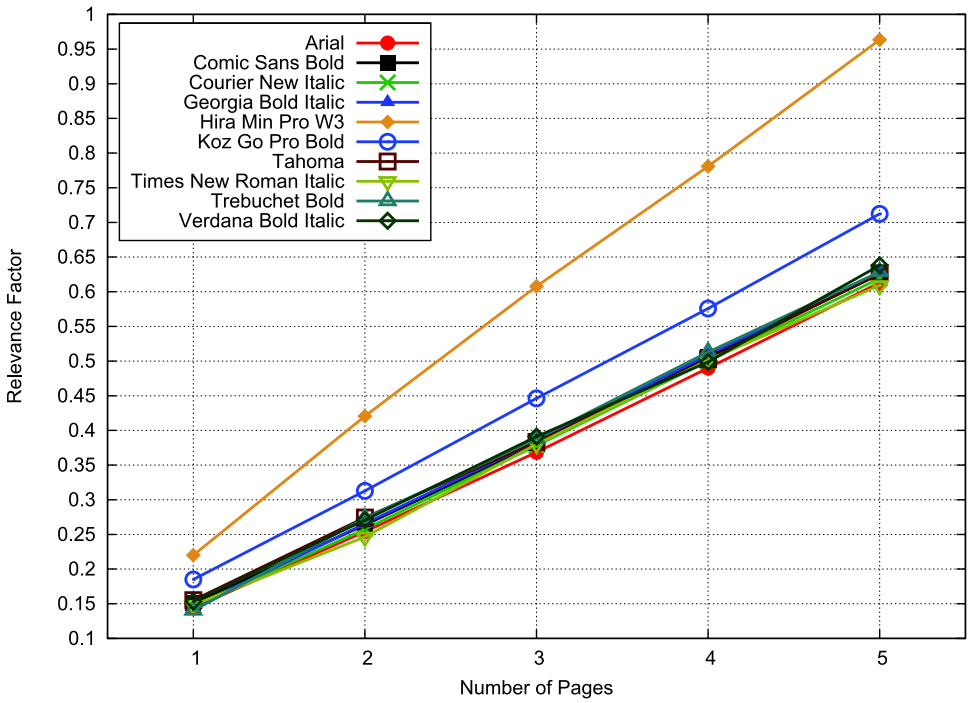


Fig. 12 Page and text transparency

The variable tr_i indicates the existence or not of text transparency in page i. The value 0 is assumed in the case that there is no text transparency in page i, otherwise the value 1 assumed.

5.4 Total computational cost

The experiments presented until this point define the individual costs for different objects and pages in a PDF document. However, the goal is to estimate a computational cost associated to the whole PDF document (or job), providing an approximation of the total computational ripping effort. Equation (9) combines the individual costs to obtain the final cost of a PDF document, denoted as $cDoc$:

$$\begin{aligned} cDoc = {} & cPag_{tPagTot} + cIm_{tImTot} \\ & + cImT_{tImTTot} + cRe_{tReTot} \\ & + cReT_{tReTTot} + cTxtTot. \end{aligned} \tag{9}$$

6 Metrics evaluation

The total cost introduced in (9) was validated using the same customers jobs previously described in Sect. 3 (Table 1). Aiming at a more accurate simulation, the hardware used in these tests is very similar to the one usually used in real PSPs: a cluster of blades composed by 5 nodes, in which, each node contains a Intel Xeon 3.0 GHz processor (64-bit quad-core), with 8 GB of RAM and 400 GB of disk. Each core was considered as an active processing unit, enabling up to 20 processes simultaneously (four per blade). The blades are connected through a Gigabit Ethernet. The operating system used is Windows Server 2003 R2 Standard x64 Edition, which is the one used in the printing environments due to its compatibility with several industrial RIPs.

Despite Java is not usually employed in high performance programming, in the last decade this language has already been used by researchers in this field (such as [5]). Furthermore, Java offers some advantages like portability, extensibility and high level of abstraction. Moreover, in this work the native support for the PDFBox library was a considerable benefit. Therefore, the scheduling approaches described in Sect. 2 are implemented in Java using the JDK (Java Development Toolkit)1.5_16 amd64 version and MPJE (MPJ Express) [10], which is an implementation of the MPI (Message Passing Interface) [8] standard for Java language and presents a similar performance when compared to libraries such as MPICH (MPI Chameleon) [9] and LAM (Local Area Multicomputer)/MPI [7].

So far, several metrics for calculating a PDF job computational cost have been presented through experimental test cases. Such metrics should now be validated over real customer jobs. Thus, the selected jobs were executed (ripped) sequentially, obtaining an execution time for each one of them. With these results, it is possible to compare the real computational cost to the estimated execution time, verifying if the proposed metric is consistent with the real costs. These results are shown in Table 3 where jobs are sorted by the decreasing order of computational cost. It is worth men-

Table 3 Ripping time and estimated cost

Job	Ripping time (sec)	Estimated computational cost
Brochure 2	1907.12	90.07
Newspaper 2	1830.28	89.73
Letter 2	1703.20	80.98
Newsletter 1	1614.08	71.97
Brochure 3	1399.97	51.34
Newspaper 1	1358.73	49.61
Postcard 1	959.20	40.02
Flyer 3	862.21	39.52
Poster 1	877.05	39.44
Letter 1	841.30	36.43
Card 1	830.23	36.36
Newsletter 2	879.18	35.21
Letter 3	874.15	35.00
Flyer 4	572.58	25.55
Flyer 2	276.67	17.09
Flyer 1	375.92	16.38
Poster 2	50.00	6.81
Brochure 1	41.84	6.76
Postcard 2	16.47	2.38
Card 2	13.40	2.15

tioning that each job was executed 20 times, and the computational time shown in the table is actually the average of these executions.

It is possible to observe that the proposed metric in general provides a satisfactory estimation of the job cost. Based on these results, it is possible to detect different levels of estimated computational costs among the jobs which were correctly described by the metric values. Thus, the use of the proposed metric makes possible the distinction of high cost jobs from those that have a small computational cost.

However, when costs are similar, it is possible to observe some incorrect estimations. This situation can be observed in the jobs Flyer 1, Newsletter 2 and Letter 3, which present a high execution time but a low estimated cost. This may be caused by the existence of PDF objects not considered in the metrics formulation, for example, paths and shading pattern objects. These objects may represent an additional ripping computational time, which is not estimated in the proposed metrics. However, even with this additional computational cost in some jobs, the metric is able to give a valid cost estimation for the jobs, and therefore can be applied for predicting the ripping effort for each job.

7 Queue management

The computational costs estimated through the application of the PDF Profiler tool are used in this section to illustrate how the (re-)organization of the printing jobs queue can result in a better utilization of the RIPs, consequently offering an increased ripping phase performance. New scheduling and load balancing strategies *are not* in the scope of this work. The experiments presented in this section are an exercise to evaluate the printing jobs queue management over the existent parallel RIP strategies. Based on that analysis, some directions will be proposed on how to use the computational cost predicted by the metrics to allocate jobs in a clever way.

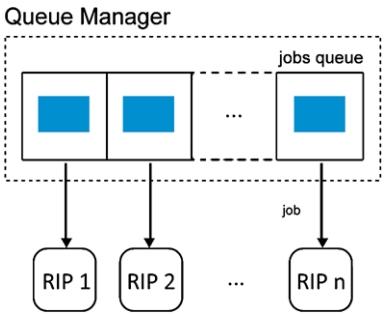


Fig. 13 One RIP per job

First of all, this section introduces the current available parallel RIP strategies. After that, the experiments environment scenario is described (printing jobs queue and configuration of multiple RIPs). Following, the obtained results are presented for each individual existing ripping strategy. Finally, a discussion about the obtained results points out some interesting conclusions, identifies the benefits and drawbacks of re-organizing the printing queue and points out some directions about how to use this information to guide the design of the Adaptative Job Router tool.

7.1 Current parallel ripping strategies

The current strategies based on parallel and distributed systems in traditional RIP environments are based on several RIPs being applied together to rip a given queue of jobs in parallel. Three strategies usually used in such environments are introduced in this section.

7.1.1 Strategy 1—one RIP per job

The first strategy is illustrated in Fig. 13. Through the application of this configuration, each RIP processes a whole job. In this case, there are two situations that may happen: the load distribution is unfair or several RIPs are left idle. This type of setting works well only for a constant and large queue of small jobs with large reusability;

7.1.2 Strategy 2—all RIPs allocated to one job

Strategy 2 is a "brute force" approach (Fig. 14), in which each RIP asks for more work, whenever it becomes idle.

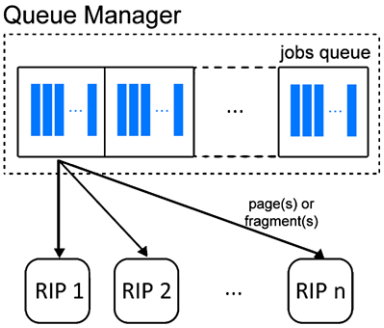


Fig. 14 All RIPs to one job

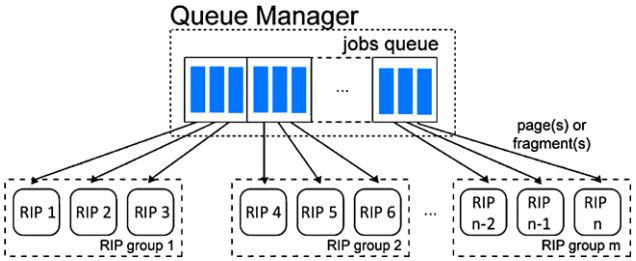


Fig. 15 Fixed RIPs per job

This strategy works well with jobs that present low reusability and demand high computational effort. On the other hand, in a mixed job environment, where many of these jobs may be quickly processed, it is really counterproductive.

7.1.3 Strategy 3—fixed RIPs per job

Strategy 3 configuration (Fig. 15) is based on the fact that an "average" job requires a certain number of RIPs to be successfully consumed at engine speed. As any static configuration, it only works if the PSPs work with some specific types of jobs. Furthermore, a set of RIPs will only be allocated for a new job when all the RIPs of the group are idle, i.e., at the moment that the current job is fully processed.

7.2 Experiments description

Clearly, there is a multitude of possible queues configuration and RIP farms scenarios which could be tested. However, this work does not intend to perform exhaustive experiments. The intention is to investigate whether or not the metrics are useful to improve the ripping task through the re-organization of the printing jobs queue.

Two queues configuration were chosen to be used in the experiments. They have the same set of documents, but are organized differently. The first queue is *not* organized by the estimated costs obtained through the metrics. The position of the jobs in the queue is the same as the one presented in Table 2. From now on, this queue will be referred as the *random* queue. The second queue (*estimated cost* queue) is organized using the metrics as it can be seen in Table 3.

The RIP farm scenario used in the experiments is composed by four independent RIPs. They all share the same hard disc in which the document jobs are stored. On the other hand, the results of each RIP unit are written on a local disk (to avoid extra communication costs) and is directly transmitted to its associated printer.

The adopted methodology for the experiments is to apply the three current parallel ripping strategies for both queue configurations over the RIP farm scenario described above. For each strategy, a comparison between the ripping times for both queues (with and without the use of the metrics to organize them) is presented.

7.3 Results

In this section, the results obtained through the use of the three strategies for the parallel ripping over both printing job queues (random and organized by the estimated computational cost) will be presented.

7.3.1 Strategy 1

In Strategy 1, each existing RIP will process an entire job (Fig. 13). In this context, a scheduling implementation will pass a job for each free RIP and as each RIP finishes its work, it will order more tasks to the scheduler. This scenario works well for small jobs, with high reusability, thus allowing each RIP to take advantage of reusability and to finalize the computation of their job in time to continually feed printers. However, if the jobs size vary from job to job (which is a common situation), the RIP units will be sub/over loaded, what will also affect the performance. Table 4 shows how the jobs would be distributed among the RIPs using the random queue.

It is possible to observe that this load distribution is not balanced, specially when comparing RIP 1 and RIP 4. The total time RIP 1 is being used is 3852.93 s against 4936.29 s of RIP 4. Actually, RIP 4 will still be ripping for 1083.36 seconds (approximately 18 minutes) longer than RIP 1. This time difference drops considerable when the queue is ordered by the jobs estimated computational cost obtained through the metrics (Table 5). The total time RIP 1 is being used is 4232.20 s against 4504.93 s of RIP 2. In this case, RIP 1 will still be ripping for 275.66 s (approximately 4.5 minutes) longer than RIP 2. In this example, the use of a simple scheduling strategy considering the estimated computational cost of a job can reduce the idle time of the fastest RIP in about 4 times. The adopted strategy forces the computation of larger jobs first, eliminating the possibility of assigning a large job to a RIP and small jobs to the others when the queue is almost empty.

Table 4 Strategy 1—random order (times in seconds)

Job	Ripping time	Rip1	Rip2	Rip3	Rip4
Brochure 1	41.84	41.84			
Brochure 2	1907.12		1907.12		
Brochure 3	1399.97			1399.97	
Card 1	830.23				830.23
Card 2	13.40	55.25			
Flyer 1	375.92	431.17			
Flyer 2	276.67	707.84			
Flyer 3	862.21	1570.05			
Flyer 4	572.58				1402.81
Letter 1	841.30			2241.27	
Letter 2	1703.20				3106.01
Letter 3	874.15	2444.20			
Newsletter 1	1614.08		3521.20		
Newsletter 2	879.18			3120.45	
Newspaper 1	1358.73	3802.93			
Newspaper 2	1830.28				**4936.29**
Postcard 1	959.20			4079.65	
Postcard 2	16.47		3537.67		
Poster 1	877.05		4414.72		
Poster 2	50.00	**3852.93**			

Table 5 Strategy 1—estimated cost order (times in seconds)

Job	Ripping time	Rip1	Rip2	Rip3	Rip4
Brochure 2	1907.12	1907.12			
Newspaper 2	1830.28		1830.28		
Letter 2	1703.20			1703.20	
Newsletter 1	1614.08				1614.08
Brochure 3	1399.97				3014.05
Newspaper 1	1358.73			3061.93	
Postcard 1	959.20		2789.48		
Flyer 3	862.21	2769.33			
Poster 1	877.05	3646.38			
Letter 1	841.30		3630.78		
Card 1	830.23				3844.28
Newsletter 2	879.18			3941.11	
Letter 3	874.15		**4504.93**		
Flyer 4	572.58	4218.96			
Flyer 2	276.67				4120.95
Flyer 1	375.92			4317.03	
Poster 2	50.00				4170.95
Brochure 1	41.84				4212.80
Postcard 2	16.47				**4229.27**
Card 2	13.40	4232.36			

The experiments over Strategy 1 showed that the use of an estimated computational cost to order the queue can improve the overall ripping performance. However, if very large jobs are present in the queue together with a high number of small jobs, it is hard to achieve good performances since the larger job will set the final ripping time. Moreover, another drawback in this particular strategy is the fact that several RIPs will not be used if the number of jobs in the queue is smaller than the number of available RIPs.

7.3.2 Strategy 2

Strategy 2 implements the idea of using all RIPs available for a single job (Fig. 14) and each RIP will process a portion of a given job. The scheduler will be responsible for defining these portions, and inform each available RIP about the page interval it should process.

Each RIP will then split the job and generate its portion to be processed. Therefore, the split process itself is done by each RIP, and not by the scheduler, which only define which pages each fragment must contain. For the following experiments, it is assumed that the jobs could be split in exactly n portions, where n corresponds to the number of RIPs. This would be the ideal situation and, considering the test case above, the total ripping time for all jobs (17283.59 s) would be divided by 4 (the number of RIPs available in this test) resulting in 4320.09 s for each RIP. This perfect load balance is barely possible to achieve in reality for the following reasons:

- Each portion would probably consist of a fractionary number of pages of the job, what is not possible since the atomic unit of a job is one page.
- If the number of pages in a given job is smaller than the amount of RIPs, some RIPs will be idle.
- There is always the possibility to have a different number of pages in each portion resulting in different computational costs and consequently a bad load balance.

7.3.3 Strategy 3

Finally, the third strategy requires the use of a fixed number of RIPs (r) for each job. Figure 15 exemplifies this strategy considering the situation in which each group is composed by three RIPs. This configuration is based on the fact that the "standard" PSPs jobs require a specific number of resources to maintain the printers working continuously. Thus, several groups of RIPs are available to process the jobs and each job will be assigned to a distinct group of RIPs.

In this configuration, each job is split into r fragments and, as in Strategy 2, the split process itself is done by each RIP and not by the scheduler, which only defines which pages each fragment must contain. Differently from Strategy 2, jobs now have to be split among a fewer number of

Table 6 Strategy 3—random order (times in seconds)

Job	Ripping time	Rip1 and Rip2	Rip3 and Rip4
Brochure 1	20.92	20.92	
Brochure 2	953.56		953.56
Brochure 3	699.99	720.91	
Card 1	415.12	1136.02	
Card 2	6.70		960.26
Flyer 1	187.96		1148.22
Flyer 2	138.34	1274.36	
Flyer 3	431.11		1579.33
Flyer 4	286.29	1560.65	
Letter 1	420.58	1981.30	
Letter 2	815.6		2430.93
Letter 3	437.08	2418.37	
Newsletter 1	807.04	3225.41	
Newsletter 2	439.59		2870.52
Newspaper 1	679.37		3549.88
Newspaper 2	915.14	4140.55	
Postcard 1	479.60		4029.48
Postcard 2	8.235		4037.72
Poster 1	438.53		4476.24
Poster 2	25.00	4165.55	

Table 7 Strategy 3—estimated cost order (times in seconds)

Job	Ripping time	Rip1 and Rip2	Rip3 and Rip4
Brochure 2	953.56	953.56	
Newspaper 2	915.14		915.14
Letter 2	851.60		1766.74
Newsletter 1	807.04	1760.60	
Brochure 3	699.99	2460.59	
Newspaper 1	679.37		2446.11
Postcard 1	479.60		2925.71
Flyer 3	431.11	2891.69	
Poster 1	438.53	3330.22	
Letter 1	420.65		3346.36
Card 1	415.12	3745.33	
Newsletter 2	439.59		3785.95
Letter 3	437.08	4182.41	
Flyer 4	286.29		4072.24
Flyer 2	138.34		4210.57
Flyer 1	187.96	4370.37	
Poster 2	25.00		4235.57
Brochure 1	20.92		4256.49
Postcard 2	8.24		4264.73
Card 2	6.70		4271.43

RIPs (typically 2 or 3 RIPs per group) resulting in larger jobs. In this case, larger jobs are easier to manipulate since they probably would be composed by several pages what makes it easier to split them in a more balanced way.

In the following experiment, for the sake of simplicity, it is supposed that it is possible to split the jobs equally among 2 groups composed by a pair of RIPS. Considering the queue ordered in a random way as presented in strategy one, Table 6 presents the sequence that the jobs would be assigned to the RIPs. When comparing the total ripping time needed for each group of RIPSs, a difference of 310.76 s (approximately 5 minutes) is found. This time can be reduced by reorganizing the queue by the jobs estimated cost.

According to Table 7, when using the estimated costs, the difference between RIPs groups drop down. The total time for the group formed by RIPs 1 and 2 is 4370.37 s against 4271.45 for the group with RIPs 3 and 4. This means the RIPs 1 and 2 will be still ripping for only 98.92 seconds (approximately 1.5 minutes) longer than RIPs 3 and 4. That can be explained by the fact that a reduction of the size of the job leads to a better load balance among the RIPs. At this point, it is important to remember that this experiment supposes the jobs are always divided in two equal portions, what is barely possible in reality. However, as stated before, a well balanced division of the workload can be obtained with larger jobs over a few number of RIPs in a group. In fact, this is essential, since this strategy will only work as expected if the jobs have a similar computational cost.

Table 8 Strategies comparison (times in seconds)

Job	Rip1	Rip2	Rip3	Rip4
Strategy 1—Random	3852.93	4414.72	4079.65	**4936.29**
Strategy 1—Estimated cost	4232.36	**4504.93**	4317.03	4229.27
Strategy 3—Random	4165.55		**4476.24**	
Strategy 3—Estimated cost	**4370.37**		4271.43	

The major drawback of Strategy 3 is the fact that the RIPs will only be allocated for a new job when all RIPs in a group are free (i.e., at the time the current job is completely processed by the group). Therefore, some RIPs of a group can be idle, while others of the same group did not finish computing their portions, under-using the resources available.

7.4 Discussion

Table 8 introduces a comparison between the total ripping time of Strategies 1 and 3 computing both queues (Strategy 2 is not considered since it is only possible with a fine control of the jobs split procedure).

The smallest difference between the fastest and the slowest ripping units is achieved using Strategy 3 over the ordered queue by estimated cost. It is again Strategy 3 that presents the fastest overall ripping time among all configurations (4370.37 s).

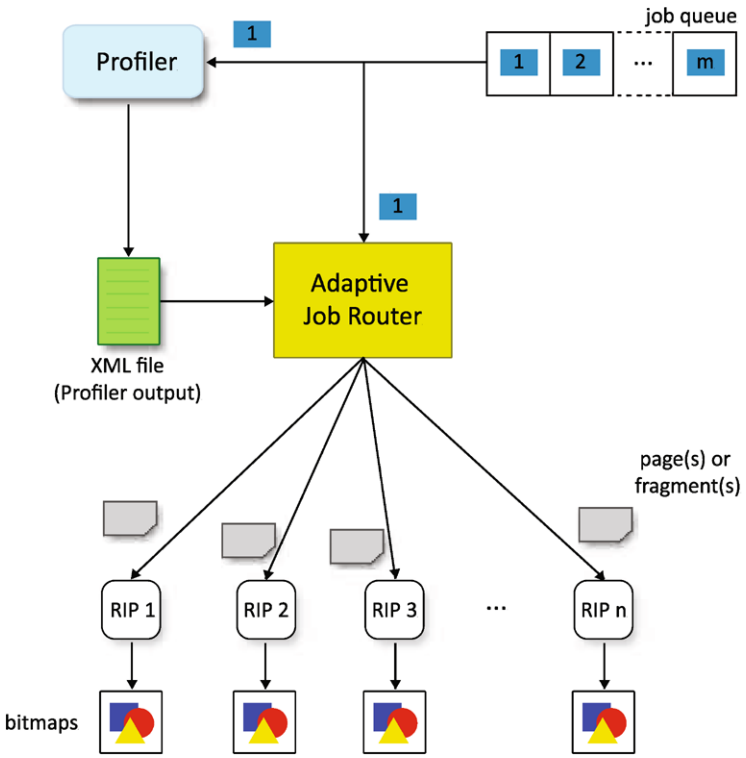


Fig. 16 Adaptive Job Router

Another interesting observation is that Strategy 3 applied to the random queue is faster than Strategy 1 over the ordered queue. This indicates that not only the queue management is important to improve the throughput, but also the chosen parallel ripping strategy clearly affects the ripping performance. Finally, in between Strategies, it can be noticed that the ordered queue always outperforms the random queue in terms of a better utilization of the resources.

The previous experiments suggest some elements that should be considered in a future dynamic scheduling approach:

- the idea of splitting a job in smaller portions leads to a better load balance among the RIPs;
- on the other hand, too small jobs lead to too small portions which can harm the performance since it is harder to split the jobs in a balanced way;
- the estimated computational cost of a job can be used to organize a queue to achieve better load balance.

8 Conclusion and future works

Characteristics and objects of PDF jobs were analyzed to define metrics to allow the estimation of the computational cost to rip PDF documents (or jobs). The results obtained so far indicate that the proposed metrics are a reliable way to weight the ripping cost.

Using the PDF Profiler tool, it is possible to gather the necessary information to find the complexity of a job when the metrics are applied. The estimated cost could be used by the Adaptive Job Router (as illustrated in Fig. 16) to organize the processing queue and thus improve the overall load balance.

A further research topic would be to optimize the Adaptive Job Router in such a way it could schedule the documents and re-order the queue when appropriate in order to improve the overall throughput. The system also could work based on the concept of preemptive queues. The Adaptive Job Router would decide the best order for the queue while performing the profiling for each job. For this approach to be successful, it is crucial to assign a priority for each job based on its processing weight. Also, this application would be able to decide how many processing resources are necessary to rip a single job at engine speed. Thus, we believe the metrics defined in this work are suitable to address the queue management needs.

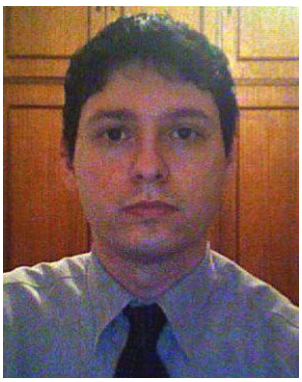
Finally, the advantages introduced with the new page allocation strategy would need to be weighted against the profiling overhead. Also it is important to identify where the job can be profiled with the least impact in the workflow.

References

1. Adobe Systems (2003) PDF Reference, 4th edn. Adobe Systems Incorporated, San Jose
2. Davis P, deBronkart D (2000) PPML (Personalized Print Markup Language): a new XML-based industry standard print language.

In: XML Europe 2000, pp 1–14, Paris, France. International Digital Enterprise Alliance
3. Déjean H, Meunier J-L (2006) A system for converting PDF documents into structured XML format. In: DAS'06: Proceedings of the 7th International Workshop on Document Analysis Systems, Nelson, New Zealand. LNCS, vol 3872. Springer, Berlin, pp 129–140
4. Extensible Markup Language (XML Home Page) (2009) Extracted from http://www.w3.org/XML, 30th November, 2009
5. Getov V, Hummel SF, Mintchev S (1998) High performance parallel programming in Java: exploiting native libraries. Concurr Pract Exp 10(11):863–872
6. ImageMagick Home Page (2009) Extracted from http://www.imagemagick.org, 30th November 2009
7. LAM/MPI Home Page (2009) Extracted from http://www.lam-mpi.org/, 30th November 2009
8. MPI Home Page (2009) Extracted from http://www.mpi-forum.org/, 30th November 2009
9. MPICH Home Page (2009) Extracted from http://www-unix.mcs.anl.gov/mpi/mpich/, 30th November 2009
10. MPJ Express Home Page (2009) Extracted from http://mpj-express.org/, 30th November 2009
11. Nunes T, Giannetti F, Kolberg M, Nemetz R, Cabeda A, Fernandes LG (2009) Job profiling in high performance printing. In: ACM DocEng'09: Proceedings of the 9th ACM Symposium on Document Engineering, Munich, Germany. ACM, New York, pp 109–118
12. Nunes T, Raeder M, Kolberg M, Fernandes LG, Cabeda A, Giannetti F (2009) High performance printing: increasing personalized documents rendering through PPML jobs profiling and scheduling. In: IEEE CSE'09: Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, Vancouver, Canada. IEEE Comput Soc, Los Alamitos, pp 285–291
13. Nunes T, Fernandes LG, Giannetti F, Cabeda A, Raeder M, Bedin G (2007) An improved parallel XSL-FO rendering for personalized documents. In: Euro PVM/MPI'07: Proceedings of the 14th European PVM/MPI Users Group Meeting. Recent advances in parallel virtual machine and message passing interface, Paris, France. LNCS, vol 4757. Springer, Berlin, pp 56–63
14. Nunes T, Giannetti F, Fernandes LG, Timmers R, Raeder M, Castro M (2006) High performance XSL-FO rendering for variable data printing. In: ACM SAC'06: Proceedings of the 21st ACM Symposium on Applied Computing, Dijon, France. ACM, New York, pp 811–817
15. PDFBox Home Page (2009) Extracted from http://www.pdfbox.org, 20th March 2009
16. Purvis L, Harrington S, O'Sullivan B, Freuder EC (2003) Creating personalized documents: an optimization approach. In: ACM DocEng'03: Proceedings of the 2003 ACM Symposium on Document Engineering, Grenoble, France. ACM, New York, pp 68–77
17. Yuan F, Liu B, Yu G (2005) A study on information extraction from PDF files. In: ICMLC'05: Proceedings of the 4th International Conference Advances in Machine Learning and Cybernetics. LNCS, vol 3930. Springer, Berlin, pp 258–267

Luiz Gustavo Fernandes is an Associate Professor of the Postgraduate Computer Science Program (PPGCC) at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil. His primary research interests are Parallel and Distributed Computing, High Performance Applications Modeling, Document Engineering and Performance Evaluation. Dr. Fernandes received his Ph.D. in Computer Science from the Institut National Polytechnique de Grenoble, France, in 2002. He currently leads the Parallel Applications Research Group (GMAP) at PUCRS.



Thiago Nunes joined ThoughtWorks in 2009 (Porto Alegre, Brazil office). His primary research interests are Parallel and Distributed Computing, Performance Evaluation and Software Engineering. Thiago received a Master Degree in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil, in 2009, whilst he was a member of the Parallel Applications Research Group. Currently, he is working with agile methodologies and distributed enterprise applications.

Mariana Kolberg is an Associate Professor at the Universidade Luterana do Brasil (ULBRA), Canoas, Brazil. In 2009, she was a research assistent financied by HP at the Pontifical Catholic University of Rio Grande do Sul (PUCRS). Dr. Kolberg received a Ph.D. in Computer Science from Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil in 2009. During her PhD, she spent two years at Karlsruhe University, Germany. Her primary research interests are Verified Computing, Interval Arithmetic, Parallel Computing, Parallel Applications Modeling and Performance Evaluation.

Fabio Giannetti is a Senior Researcher at HP Labs. Since he joined HP in 2000, his focus has been on Digital Publishing, Variable Data Print and Printing Workflow Technologies. He has been actively involved in the research community releasing open-source tools as well as advancing the state of the art in numerous HP products. Fabio holds an Ms.Eng. (Hons) Computer Science from University of Genoa (Italy). Fabio represents HP at the W3C XSL Working Group.

Rafael Nemetz is Computer Engineer by Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil. Nowadays, he is Master degree student of the Postgraduate Computer Science Program (PPGCC) at PUCRS. He is also member of the Parallel Applications Research Group (GMAP), where his researches cover Distributed Computing, Parallel Applications, Performance Evaluation and Document Engineering. Rafael also has experience in Embedded Systems.

Alexis Cabeda is a member of HP Brazil research team and his primary research interests include Digital Publishing and Variable Data Print. Alexis received a Master Degree in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS), Brazil, in 2006.