JAR Tool: Using Document Analysis for Improving the Throughput of High Performance Printing Environments

Mariana Kolberg
UFRGS
Porto Alegre, Brazil
mlkolberg@inf.ufrgs.br

Luiz Gustavo Fernandes
GMAP - FACIN - PUCRS
Porto Alegre, Brazil
luiz.fernandes@pucrs.br

Mateus Raeder
GMAP - FACIN - PUCRS
Porto Alegre, Brazil
mateus.raeder@acad.pucrs.br

Carolina Fonseca
GMAP - FACIN - PUCRS
Porto Alegre, Brazil
carolina.fonseca@acad.pucrs.br

ABSTRACT

Digital printers have consistently improved their speed in the past years. Meanwhile, the need for document personalization and customization has increased. As a consequence of these two facts, the traditional rasterization process has become a highly demanding computational step in the printing workflow. Moreover, Print Service Providers are now using multiple RIP engines to speed up the whole document rasterization process, and depending on the input document characteristics the rasterization process may not achieve the print-engine speed creating a unwanted bottleneck. In this scenario, we developed a tool called Job Adaptive Router (JAR) aiming at improving the throughput of the rasterization process through a clever load balance among RIP engines which is based on information obtained by the analysis of input documents content. Furthermore, along with this tool we propose some strategies that consider relevant characteristics of documents, such as transparency and reusability of images, to split the job in a more intelligent way. The obtained results confirm that the use of the proposed tool improved the rasterization process performance.

Categories and Subject Descriptors

I.7 [**Document and Text Processing**]: Document management; I.7.2 [**Document and Text Processing**]: Document Preparation; I.7.4 [**Document and Text Processing**]: Electronic Publishing

Keywords

Document Analysis; Document profiling; High performance printing; Load balancing

DocEng'14, September 16–19, 2014, Fort Collins, Colorado, USA.

http://dx.doi.org/10.1145/2644866.2644887.

1. INTRODUCTION

Digital printers have consistently improved their speed in the past years. Meanwhile, the need for document personalization and customization has increased. As a consequence of these facts, the traditional rasterization process, responsible for transforming documents described in formats that most printers cannot interpret (e.g. *PDF*) in printable formats (e.g. *Bitmap*), has become a highly demanding computational step in the printing workflow. In general, a Print Service Provider (or PSP) uses a set of RIP (*Raster Image Processing*) engines in parallel to achieve the best possible performance considering a queue of n initial jobs to be processed and new jobs being inserted in the queue at any time.

However, a wide variety of PDF documents with different characteristics can be found in the context of PSPs. The presence of each characteristic in a document page has a different impact on the computational time needed to raster it. In this context, the information about documents characteristics might be used to split the document in fragments in a clever way, improving the load balance of documents among RIP engines and leading to a better performance of the rasterization process. This paper presents the development of a tool named Job Adaptive Router (JAR) aiming at balancing the workload among RIP engines and optimizing the rasterization process. JAR uses the metrics proposed in [1, 3] to estimate the job rasterization cost and chooses the most suitable strategy to split the document considering the job characteristics that might have an important influence on the computational time. Our main contribution in this paper can be summarized as (i) the development of a Job Adaptive Router (JAR) to obtain a fair load balance among RIPs and (ii) the definition of 5 different strategies based on document analysis to split the queue in such a way that the workload will be balanced among RIPs.

2. JOB ADAPTIVE ROUTER

In general, load balance algorithms for the rasterization process do not consider documents internal content. However, the document content is a key information in PSP scenario and a detailed analysis of the PDF document characteristics may improve the rasterization throughput. The identification of some specific characteristics in each document page allows a more intelligent workload division among RIPs.

We first had to modify the PDF Profiler tool to be able to compute the computational cost of each page of the document. In addition, we developed a tool called PDF Splitter, which breaks the PDF document into several fragments (which will be the tasks to be rasterized). PDF Splitter can be used to break the input document in two ways: individual pages or groups of pages. Note that the least grain of each piece of a fragment is a page.

2.1 JAR Architecture

The Job Adaptive Router main goal is to decide during the runtime which developed strategy will be selected for a given job. This decision is based on the document information contained in a XML file provided by PDF Profiler. Based on this information, JAR will split the document generating fragments that will be rasterized. Figure 1 shows an overview of the rasterization process using JAR tool.

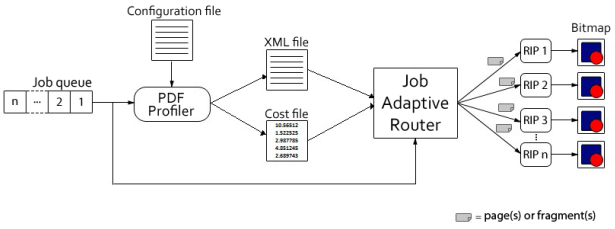

Figure 1: Overview of rasterization process using JAR

The *Job queue* has all jobs (PDF documents) that must be rasterized, and each one of them is sent to PDF Profiler tool. In addition to that, PDF Profiler also receives as input a *Configuration file* which indicates which information must be extracted from the job. Thus, PDF Profiler analyzes the job and generates two files: (i) the file containing the computational cost of each page; (ii) a XML file containing information about each document page (which pages contain text, which contain images, whether these images are opaque, transparent or reusable, etc.). After that, JAR analyzes these files and discovers which strategy should be used for a given job.

Figure 2 presents the JAR internal architecture. JAR is divided into 3 modules: Strategy Identifier, Fragment Organizer and PDF Splitter. Each one of these modules has a specific function and, in order to improve the performance, they run in different concurrent threads. First, the XML and cost files are read by the Strategy Identifier module. This module is mainly responsible for finding out which strategy will be used over the jobs. According to the information obtained from the XML file, Strategy Identifier knows the characteristics of the job, such as the amount of pages with transparency and/or reusability, choosing the best strategy. Strategy Identifier also reads the cost file to create a list of job pages with their characteristics and costs.

After discovering the best strategy, the Strategy Identifier module sends it along with a list of jobs pages characteristics for the next module, the Fragment Organizer. This module is responsible for informing the PDF Splitter module how the job must be split. As mentioned before, the PDF Splitter only divides the job into fragments and needs to known which pages constitute each fragment. Thus, the way fragments will be created depends on the selected strategy. In this context, Fragment Organizer module receives

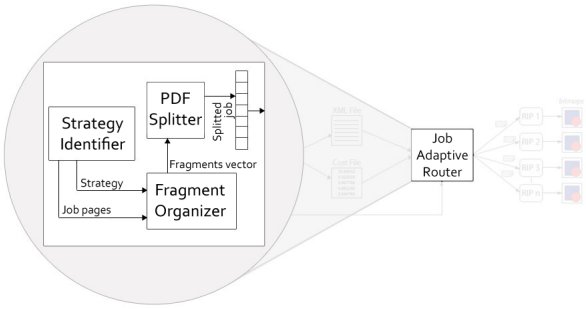

Figure 2: JAR internal architecture

the Strategy Identifier output and then creates a *Fragments vector*, which contains the information of which pages will be grouped to form each fragment. This *Fragments vector* is sent to the PDF Splitter that creates the job fragments and inserts them into another queue of jobs ready to be sent to the available RIPs.

2.2 Developed Strategies

This section proposes five strategies to obtain a better load balancing among RIPs, focusing on two job characteristics that might have an important influence on the computational time for rasterizing a job [3]: the amount of reusable objects and transparent images. The computational cost of each page is obtained using PDF Profiler tool.

The first strategy, called *Transparency Strategy*, is suitable for documents which have only transparent images without any reusable images. Since the cost for rasterizing pages with transparent images is high, these pages should not be allocated to a single RIP. This algorithm separates the pages with transparency, ordering them by their computational cost (from the highest to the lowest cost) in a queue of pages. After ordering these pages with transparency, the remaining pages of the job (pages with opaque images and/or text) are also sorted by their computational cost and are inserted individually into the pages queue.

The *Reusability Strategy* deals with documents which have reusable images but no transparent images. The proposed algorithm first combines the pages with reusable objects, creating sets of pages that contains the same image. After that, these sets are inserted into the queue of pages, sorted by their computational cost (from the highest to lowest). The remaining pages are inserted in the same queue individually. Finally, the load distribution is performed creating n fragments (where n is the number of RIPs).

The *More-Transparency-than-Reusability Strategy* is a first mixed strategy and deals with documents that have more pages with transparent images than pages with reusable images. As the most costly pages of a PDF document are that ones that contain transparency, these pages are sorted by their computational cost and inserted in the page queue. After that, pages with reusability are grouped, creating sets of pages that are also inserted in the page queue, always sorted by their computational cost. The remaining pages of the job are inserted individually in the same queue. When all pages are in this queue, the distribution of the pages or sets among the fragments starts.

The *More-Reusability-than-Transparency Strategy* is suitable for PDF documents which combine transparency and

reusability but there are more pages with reusability than pages with transparency. Similarly to the *Reusability strategy*, the algorithm groups all pages with the same reusable image, creating different sets. They are then inserted in the page queue according to their computational cost. Pages containing transparency are the next to be inserted in the queue followed by pages with opaque images and/or text. Once all pages/sets are in the queue, they are distributed among the fragments.

Finally, the *No-Transparency, No-Reusability Strategy* was developed to be used in the case where no transparent or reusable object are found. In that case, the algorithm sorts all pages by their computational cost (from the highest to the lowest cost). After that, pages are inserted in the page queue and they are distributed among fragments.

3. PERFORMANCE EVALUATION

This section presents the environment setup and input job queues used to run the Job Adaptive Router and discusses the average gain achieved using our tool. Results were obtained using the ImageMagick convert tool, which is an open-source RIP engine. In the tests, PDF documents were rasterized using a 40 dpi resolution.

Aiming at verifying JAR performance, we compared the results with a well known algorithm: Largest Processing Time First (LPT) [2]. Since we need the computational time to use LPT algorithm, it is possible that RIP engines are idle waiting for jobs that are blocked in the queue waiting to be analyzed. To overcome this problem, we made a slight modification in the LPT algorithm: while a thread executes the PDF Profiler step, another thread concurrently sends jobs to the RIPs - even though they were not analyzed by the PDF Profiler tool. This improved version was called Optimized-LPT (O-LPT).

In order to get closer to the reality of the PSPs, we perform tests on a cluster composed of 32 machines connected by a high-speed Gigabit Ethernet network. Each machine consists of two AMD Opteron 246 2.0 GHz, 8 GB of main memory and 1TB hard drive. The operating system of these machines is Linux (Ubuntu 4.2.4-1ubuntu4) version 2.6.24. Each node of this cluster run one RIP engine.

To implement the five previously described strategies, the programming language we used was Java (version 1.6). We also used the Java iText library (version 2.0.7) to create the set of input jobs. The MPJ-Express library was used to enable the communication among processes.

A set of 75 jobs with different characteristics was created to evaluate the JAR Tool and the five developed strategies. The number of pages of each job vary from 60 to 220. These jobs are divided into five groups: PDFs with transparencies, PDFs with reusability, PDFs with more transparency than reusability, PDFs with more reusability than transparency and PDFs that present neither transparency nor reusability.

Moreover, for testing JAR using the five strategies developed, six different queue configurations were created: Queue 1 - 15 PDF documents with transparency; Queue 2 - 15 PDF documents with reusability; Queue 3 - 15 PDF documents with more transparency than reusability; Queue 4 - 15 PDF documents with more reusability than transparency; Queue 5 - 15 PDF documents with no transparency or reusability; Queue 6 - 15 PDF documents with mixed characteristics as follows: 3 with transparency; 3 with reusability; 3 with more transparency than reusability; 3 with more reusability than transparency; 3 with no transparency or reusability.

3.1 JAR Performance Analysis

This section presents the performance analysis of the JAR tool in comparison to the Optimized-LPT algorithm. In this sense, 30 runs were performed over each one of the six queues already described, from which the highest and lowest times were removed, obtaining an average of the other 28 values. For each execution, one process will execute JAR, meaning that to exist RIPs in parallel, the number of processes should not be less than three (one to run JAR and 2 for RIPs). Moreover, the number of processes ranged from 3 to 19.

Queue 1	3	4	5	6	7	8	9	10	11
JAR	447.6	312.4	305.2	220.2	190.8	170.6	146.6	131.8	126.6
O-LPT	488.6	308.8	258.6	250.8	189.4	150.8	154.8	139.6	157.4
	12	13	14	15	16	17	18	19	
JAR	122.4	120.2	119.8	121.2	117.6	122.8	123.8	118.4	
O-LPT	136.8	137.2	131.6	131.8	148.2	147.2	133.6	148.4	
Queue 2	3	4	5	6	7	8	9	10	11
JAR	758.4	537.8	515.4	422.2	337.6	300.2	292.4	274.8	261.2
O-LPT	896.8	480.6	504.8	328.2	301.4	310.2	260.6	259.6	273.6
	12	13	14	15	16	17	18	19	
JAR	232.4	226.6	223.8	227.2	225.8	225.4	226.8	224.4	
O-LPT	205.6	204.4	231.6	232.2	241.4	250.2	249.8	213.6	
Queue 3	3	4	5	6	7	8	9	10	11
JAR	467.4	333.8	292.8	240.4	181.8	151.6	134.2	129.6	128.4
O-LPT	485.4	265.6	264.6	241.6	197.8	169.4	153.2	148.4	164.2
	12	13	14	15	16	17	18	19	
JAR	118.4	115.8	115.6	116.8	118.4	116.8	117.6	118.4	
O-LPT	122.6	137.8	132.2	123.4	147.8	131.8	131.6	139.2	
Queue 4	3	4	5	6	7	8	9	10	11
JAR	408.4	325.4	234.6	213.6	179.8	141.6	129.8	114.6	111.8
O-LPT	425.8	224.2	221.4	208.6	147.8	159.2	133.4	125.4	137.4
	12	13	14	15	16	17	18	19	
JAR	108.4	108.8	109.8	108.8	112.2	109.4	113.8	113.6	
O-LPT	117.4	134.2	120.2	115.2	136.6	127.2	126.2	124.4	
Queue 5	3	4	5	6	7	8	9	10	11
JAR	381.8	285.2	259.8	196.6	166.6	150.8	146.2	122.4	120.2
O-LPT	401.8	266.4	263.8	219.4	167.6	150.2	148.2	126.4	133.8
	12	13	14	15	16	17	18	19	
JAR	118.4	118.8	103.4	103.2	105.4	107.8	107.8	107.2	
O-LPT	121.6	132.2	115.4	117.8	127.4	132.2	119.2	128.2	
Queue 6	3	4	5	6	7	8	9	10	11
JAR	408.6	350.2	285.2	233.8	213.2	167.6	159.6	145.4	147.2
O-LPT	498.4	300.6	288.2	232.8	218.6	169.4	164.2	150.8	168.6
	12	13	14	15	16	17	18	19	
JAR	147.6	146.4	140.6	143.8	136.8	138.2	132.4	131.6	
O-LPT	160.8	143.4	137.2	133.6	153.4	161.2	146.8	162.2	

Table 1: Execution times (in seconds)

It is possible to see in Table 1 the situation in which the *Transparency* strategy was selected by JAR for Queue 1. This strategy achieved better results in 13 of the 17 processes configurations, and when using 5 process JAR execution time presented a execution time more than 40 seconds faster than Optimized-LPT. The values obtained for *Reusability* strategy are also shown in Table 1. In this case, the Optimized-LPT, in general, presented better performance than JAR selected strategy in 9 of the 17 configurations. The better results of Optimized-LPT over the *Reusability* strategy can be explained by the chosen grain size. It is possible that the gain of grouping pages with reusable images may not compensate the time spent by JAR.

The time obtained using the queue containing documents with more transparency than reusability (Queue 3) can be seen in Table 1. Note that the performance of Optimized-LPT is better than the *More-Transparency-than-Reusability* strategy only in processes 4 and 5, showing that JAR chosen strategy has a better performance in this case. In the case of queue 4 (documents with more reusability than trans-

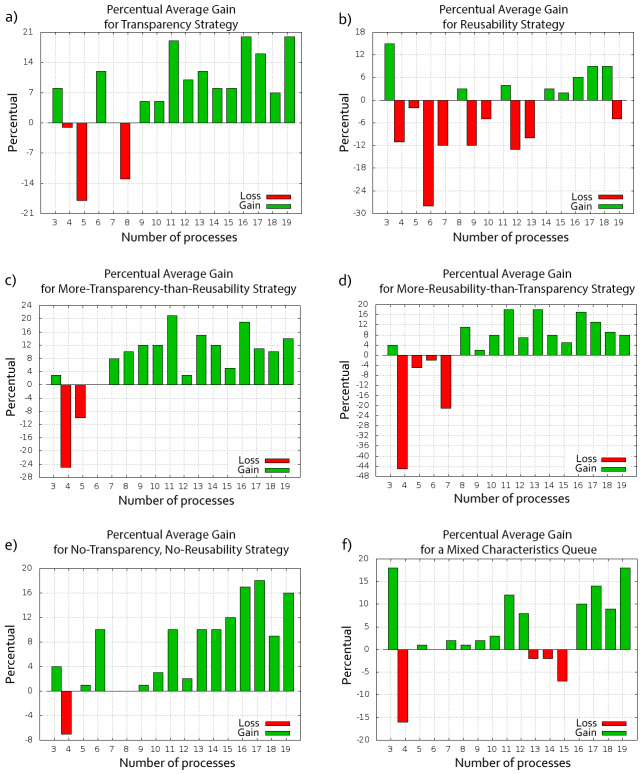

Figure 3: JAR Percentual Average Gain

parency), the results obtained with JAR lost in some configurations, but in general JAR presents better performance than Optimized-LPT. For instance, the strategy selected by JAR performed the rasterization process 25.6 seconds faster than Optimized-LPT over 11 processes.

For a queue composed of documents with no transparency nor reusable images, the strategy chosen by JAR wins in 15 of the 17 occasions compared to Optimized-LPT. Only in 2 occasions the performance obtained with Optimized-LPT is better, and in the case of 8 processes the difference is only 0.6 seconds; Finally, the results for the sixth queue (using for each document the suitable strategy, thereby addressing all five strategies) once again show that JAR performs better than Optimized-LPT in most cases (12 out of 17).

3.2 JAR Percentual Avarage Gain

This section discusses the percentual average gain for each strategy using JAR. Figure 3 presents six graphs comparing all obtained results between JAR and Optimized-LPT in terms of percentage difference. Moreover, Table 2 contains important information about each queue tested.

As can be seen in Figure 3a, only in 4 situations JAR presented worse results. However, in general the *Transparency* strategy performed better than Optimized-LPT, with an average of percentage difference equals to 7.22%. That also occurs with the *More-Reusability-than-Transparency* strategy (Figure 3d), which lost also in four cases but in most cases obtained a better gain than Optimized-LPT, with an average of the percentage differences equals to 3.60%.

Queue	Average of % differences	JAR wins	Highest % gain
1	7.22	13/17	20.64
2	-2.79	8/17	15.43
3	7.50	15/17	21.80
4	3.60	13/17	18.93
5	7.17	15/17	18.47
6	4.38	12/17	18.87

Table 2: *Performance summary*

More-Transparency-than-Reusability (Figure 3c) and *No-Trans-parency, No-Reusability* (Figure 3e) strategies presented worse results than Optimized-LPT only in 2 situations. When a queue of heterogeneous documents was used (Figure 3f), JAR obtained a better performance than Optimized-LPT with an average of percentage differences equals to 4.38%. Only the *Reusability* strategy showed a performance loss in general (Figure 3b), resulting in an average of -2.79%.

4. CONCLUSION

This paper presented the Job Adaptive Router tool for achieving a fair load balance among RIPs in the rasterization process of high performance printing environments. JAR architecture uses the information about documents content to decide during runtime which load balance strategy should be used for a given job considering its characteristics. The results obtained using JAR presented a better load balance, resulting in performance gains in comparison to Optimized-LPT. It is important to highlight that PSPs deal with a wide variety of documents with different characteristics, like the scenario of queue 6. As can be seen in Figure 3f, in this situation JAR presents average gains up to 18.87%. This scenario is specially important since it illustrates the JAR ability to dynamically choose the best strategy for each document. Moreover, it is possible to notice that the more processes we use, the better is the average gain of JAR in comparison with Optimized-LPT, which indicates a good scalability. Based on the good results, a further research is needed for investigating the JAR portability for multiprocessor machines, once nowadays clusters are typically multiprocessor machines. In addition, the analysis of the impact of other document characteristics in the rasterization time will be conducted aiming at the creation of new strategies.

5. REFERENCES

[1] L. G. Fernandes, T. Nunes, M. Kolberg, F. Giannetti, R. Nemetz, and A. Cabeda. Job profiling and queue management in high performance printing. *Computer Science - Research and Development*, 27(2):147–166, May 2012.

[2] R. L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.

[3] T. Nunes, F. Giannetti, M. Kolberg, R. Nemetz, A. Cabeda, and L. G. Fernandes. Job profiling in high performance printing. In *Proceedings of the 9th ACM symposium on Document engineering*, DocEng '09, pages 109–118, New York, NY, USA, 2009. ACM.