Computers & Industrial Engineering 88 (2015) 191–205

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A job profile oriented scheduling architecture for improving the throughput of industrial printing environments

Andriele Busatto do Carmo [a,*], Mateus Raeder [a], Thiago Nunes [a], Mariana Kolberg [b], Luiz Gustavo Fernandes [a]

[a] GMAP-PPGCC-PUCRS, Av. Ipiranga, 6681 – Prédio 32, 90619-900, Porto Alegre, RS, Brazil
[b] INF-PPGC-UFRGS, Av. Bento Gonçalves, 9500, Campus do Vale – Bloco IV, 91501-970, Porto Alegre, RS, Brazil

ARTICLE INFO

Article history:
Received 23 December 2012
Received in revised form 20 February 2015
Accepted 1 July 2015
Available online 8 July 2015

Keywords:
Scheduling
High performance printing
Documents ripping
Load balancing
Documents profiling
Parallel and distributed computing

ABSTRACT

The Digital Printing industry has become extremely specialized in the past few years. The use of personalized documents has emerged as a consolidated trend in this field. In order to meet this demand, languages to describe templates for personalized documents were proposed along with procedures which allow the correct printing of such documents. One of these procedures, which demands a high computational effort, is the ripping phase performed over a queue of documents in order to convert them into a printable format. An alternative to decrease the ripping phase computational time is to use high performance computing techniques to allow parallel ripping of different documents. However, such strategies present several unsolved issues. One of the most severe issues is the impossibility to assure a fair load balancing for any job queue. In this scenario, this work proposes a job profile oriented scheduling architecture for improving the throughput of industrial printing environments through a more efficient use of the available resources. Our results show a performance gain of up to 10% in average over the previous existing strategies applied on different job queue scenarios.

1. Introduction

The recent evolution of digital printers has opened new research opportunities in the Document Engineering field. With this new technology, printing high quality documents is no longer a daunting task for users. It now can be performed efficiently. In this scenario, we could observe the consolidation of an emerging trend: personalized documents. Before, a single instance of a document was produced for a large number of recipients, meaning that the same content was sent to them all. Nowadays, it is possible to customize documents based on the idea of adapting the document content in such a way that messages are sent to specific recipients.

In order to meet this demand, automated procedures for creating and processing documents must be developed. A new discipline called Variable Data Printing (VDP) (Purvis, Harrington, O'Sullivan, & Freuder, 2003) was introduced providing a variety of techniques, technologies, concepts and standards to enable the creation of documents with dynamic content. Several tools were developed to assist designers to create a template, which will be used to generate different document instances. Thus, the same layout is applied to various information generating a job composed of a set of personalized documents (i.e., with variable content). In this context, new languages with the necessary degree of flexibility have been developed allowing the definition of static and dynamic regions in documents.

Currently, most printers cannot interpret those languages, so a preprocessing phase is necessary to enable the correct printing of documents. Two of these phases are known as rendering and ripping, which aim at providing the final means of printing the document in a format that the printer will be able to process. With the introduction of the VDP, companies specialized in digital publishing (Print Service Providers – PSPs) have to perform these procedures on each page of personalized documents. This preprocessing phase impacts on the computational cost of the entire printing workflow.

1.1. Motivation and objectives

In general, a PSP manages a queue of n initial jobs to be processed and new jobs can be inserted in the queue at any time. The main goal is to achieve the best possible performance

* Corresponding author. Tel.: +55 5133203611; fax: +55 5133203621.
E-mail addresses: andriele.carmo@acad.pucrs.br (A.B. do Carmo), mateus.raeder@acad.pucrs.br (M. Raeder), thiago.nunes@acad.pucrs.br (T. Nunes), mariana.kolberg@inf.ufrgs.br (M. Kolberg), luiz.fernandes@pucrs.br (L.G. Fernandes).

http://dx.doi.org/10.1016/j.cie.2015.07.001
0360-8352/

considering the whole queue, and not just a single job. For this purpose, PSPs use printers with a high processing capacity, which might be able to print up to 1 page per second. In this scenario, all activities related to the preprocessing phase should be finalized in a limited time window, in such a way that the printer does not have to wait for jobs. Moreover, PSPs commonly use printers in parallel to increase the consumption of relevant documents to a given job. Thus, the performance of preprocessing phase should increase proportionally to keep the printers working continuously.

Aiming at increasing the performance of preprocessing phase, modifications on existing document description languages or even new formats must be proposed. For this purpose, reusability features were added to languages such as Portable Document Format (PDF) (Adobe Systems, 2003) to improve the throughput of document preprocessing phase. New formats *ad hoc* were also introduced, and the most widespread is PPML (Personalized Print Markup Language) (Davis & deBronkart, 2000).

The idea behind PPML is to reduce the amount of document content to be ripped since Raster Image Processing (RIP) engines are capable of capturing reusable objects in such way they need to be processed just once and the result can be used any time this object is found in the document. Similarly, the idea of optimizing the ripping process considering document characteristics seems to be an appropriate choice, specially due to the high variability of personalized documents. In this scenario, the improvement of rendering and ripping procedures is a challenging task that could also increase the PSP market competitiveness.

In several works (Nunes et al., 2006; Nunes et al., 2007; Nunes, Raeder et al., 2009), high performance computing strategies were used to improve the rendering phase throughput. In those works, the main idea was to explore the use of rendering engines in parallel.

In the context of ripping phase, there are some existing strategies to increase the performance through the use of parallel and distributed systems. These strategies do not use any previous information about the jobs in the queue and their load balancing is quite simple. Existing ripping strategies apply a simple algorithm for task scheduling among the available RIPs so that the first task in the queue will be assigned to the first RIP that finishes its computation. This simple distribution does not guarantee an equal load balance and may cause overload of some RIPs and underload of others (Fernandes et al., 2012). Therefore, the overall ripping time becomes unsatisfactory and may present unpredictable behavior.

The main goal of this work is to define a scheduling architecture that ensures a balanced load distribution based on the task computational cost obtained through the document characteristics. Works developed by Nunes, Giannetti et al. (2009) and Fernandes et al. (2012) employ documents content-based metrics to allow the estimation of computational cost to rip PDF documents (or jobs) in a process called job profiling. The PDF Profiler tool (Nunes, Giannetti et al., 2009), was developed to gather the necessary information to estimate this job cost using predefined metrics. The estimated cost can be used during the scheduling to organize the processing queue and thus improve the overall load balance (Fernandes et al., 2012). Thus, the aim of this research is to optimize the ripping performance for an entire job queue providing a fair and clever use of the available resources.

Current analysis is focused on PDF format, since it is a widespread format in the description of personalized documents. In this sense, scheduling strategies can be applied on a set of jobs fully described in PDF format or other high level abstraction formats used to describe the content of their documents.

The main contributions of this paper can be summarized as follows:

- we define a scheduling architecture in the context of PSPs, improving the overall ripping process;
- we use the created architecture to distribute the load among the available RIP engines in a more clever way, using the document computational ripping cost provided by PDF Profiler;
- we provide an adaptive environment, which allows the use of different existing scheduling algorithms and enables the experimentation of new ones.

1.2. Document structure

This paper is structured as follows: Section 2 describes the scenario overview, explaining the PSP printing environment, some general scheduling aspects on distributed environments, as well as the traditional ripping process and some scheduling strategies. The proposed approach is introduced in Section 3. Section 4 shows several kinds of input documents, in addition to three different job queue configurations we used for our tests. Section 5 presents the proposed architecture evaluation using the previously presented job queues with detailed performance analysis. Finally, Section 6 concludes the work with final remarks and considerations about future researches.

2. Scenario overview

This section introduces the PSP printing scenario which employs the VDP technique. Section 2.1 presents a brief overview of the overall printing process, describing some important aspects about the printing workflow and introducing the context of the Ripping phase. Section 2.2 introduces some general aspects of scheduling on distributed environments. Section 2.3 presents the traditional ripping process and the scheduling problem that arises. The traditional scheduling strategies used in this process, which do not consider any information about the jobs, are also presented.

2.1. General PSP printing workflow

A widely used technique in VDP scenario is VDT (Variable Data Templates) (Giannetti, 2007). This technique proposes a workflow intending to align the document creation and printing process. The goal is to increase the whole process reliability, predictability and throughput.

This workflow is composed of four main phases, as can be seen in Fig. 1: project, rendering, ripping and printing. The project phase is responsible for the creation of the document template, which contains its layout and the definition of static and dynamic parts. After that, the template is submitted to a rendering phase. At this moment, different documents can be created using this template, replacing dynamic portions by specific data of a database. Rendering phase is generally performed in a centralized way, using only a single computational resource. However, this step can be done in parallel through some rendering strategies presented in (Nunes et al., 2007; Nunes, Raeder et al., 2009). The result of this phase will be the document content described through a high level abstraction language. Each new document created in the rendering phase (e.g., PDF documents) must be transformed into a printable format in the ripping phase.

This paper focuses in the ripping phase of the printing workflow. This step is necessary due to the fact that most printers are not able to interpret flexible languages. For that reason, they need to be transformed in well-known formats such as bitmap, for example. The output of ripping phase is a bitmap document ready to be printed (printing phase).

Several advantages arise from the application of this segmented workflow: (i) PSPs can deal with large amounts of data because of the well-defined phases; (ii) design is separated from content; (iii) layout can be created without previous data definition; (iv) it is

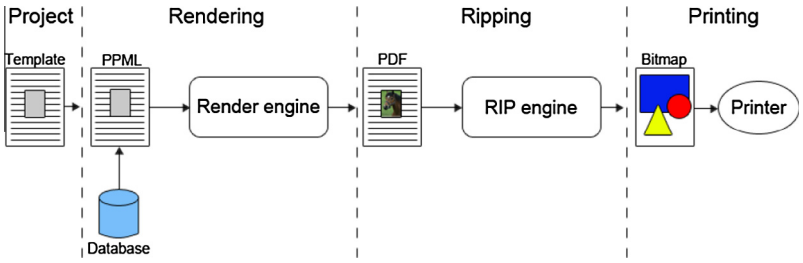

Fig. 1. Printing workflow.

possible to employ the technology that better fits the necessity of each company, and finally, (v) this workflow is easily extendable.

2.2. Scheduling on distributed environments

Meanwhile, during the development of a parallel or distributed version of a program, several characteristics must be taken into consideration to maximize the performance that is expected to be achieved. An important decision relates to the task size (grain) to be transmitted and/or processed by each processing unit. Another primary concerns is to choose a suitable scheduling algorithm (Kruatrachue & Lewis, 1988).

When dealing with distributed memory, the grain definition is critical for the proper functioning of the application. If a bad grain is set, the overhead for communication might be higher than the performance gain expected through the job division. Thus, the grain size should be defined based on specific characteristics of the application and architecture, trying to compromise cost of splitting jobs into tasks, ideal size of the tasks, communication cost and processing gain obtained by the parallelization.

Once the grain is defined, it is necessary to establish the best way to distribute available tasks among processing unities, so that none of them get overloaded or underloaded. If the adopted strategy is not a good choice, the parallelization efficiency can be directly affected.

It is not difficult to find a situation were tasks distribution has an important impact on the performance. Fig. 2 shows an example of this situation, in which using a sequential distribution (Fig. 2(a)) may lead to an unfair scheduling. In this case, tasks are assigned to machines one by one circularly (first task to machine 1, second to machine 2, third to machine 3 and so on). Such an approach will lead to an unbalanced distribution where machine 3 presents a total workload of 35 while machine 1 has a workload of just 8. Considering that, the processing of the entire task queue only ends when all machines finish their tasks, in such a way that the time to process the largest workload will define the total time spent to compute all jobs. On the other hand, when seeking a fair distribution (Fig. 2(b)), the largest workload to a single machine is 23. Thus, the total time for computing all tasks is reduced.

Based on that, the scheduling problem is characterized by searching for the optimal division of tasks. Thereby, the ultimate goal is to find a scheduling automated strategy that uses the resources in the most efficient way (in our context, the available RIP engines).

2.3. Traditional ripping process and scheduling strategies

Ripping is the process of transforming a document described by vector graphics to a bitmap format (i.e., a document composed of a dot matrix data structure representing a generally rectangular grid of pixels or points of color) that allows the printing device to interpret correctly the document content. This transformation is page-based, meaning that an image is created for each page of the input document. In this context, RIP engines (or only RIPs) are the computational elements responsible for ripping PDF documents.

Traditional RIP environments are based on parallel and distributed systems to increase its throughput and performance. Thus, several RIPs are used to rip a given queue of jobs in parallel. Two main approaches are applied to rip jobs from the input queue: splitting a single job into pieces, that will be processed in parallel, or distributing each entire job to different RIPs.

Since there is no information available about the jobs processing time, it is not possible to use elaborate scheduling algorithms. Based on that, there are three distinct scheduling strategies that are commonly applied to speed-up ripping process:

1. **Strategy 1:** one RIP per job;
2. **Strategy 2:** all RIPs per job;
3. **Strategy 3:** a fixed number of RIPs per job.

It is important to mention that in these scenarios, all jobs are stored in a file server shared by all RIPs which will read the necessary amount of information for the ripping process. On the other hand, the result produced by each RIP will be written on the local disk and directly transmitted to its correspondent printer to avoid extra communication cost.

By employing the first strategy, each existing RIP processes an entire job (Fig. 3). In this context, a job is assigned for each free RIP and as each RIP finishes it, it orders more tasks. This strategy is interesting in a scenario composed of small jobs which have a high level of reusable resources. However, for larger jobs which usually have different computational times, RIP engines will be underloaded and overloaded affecting the system load balance and consequently its performance. Furthermore, 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 less than the number of available RIPs.

Second strategy implements the idea of using all available RIPs for a single job (Fig. 4). This is the "brute force" strategy, where each RIP processes a portion of a given job. Since it is not desirable to process a fraction of a page, the atomic unit of each portion is a page. Thus, each job in the queue is always split into t tasks, where t corresponds to n (number of available RIPs) or, if the number of pages in the given job is smaller than the number of RIPs, t corresponds to pg (number of pages of the document). In this strategy, each available RIP is notified about the page interval it should process. RIPs then split the jobs generating their portions and process them. It is important to highlight that the split process is done by each RIP. An important drawback of this strategy is the possibility of splitting reusable parts of the document among different RIPs, not allowing the RIPs to take advantage of the document reusability characteristic. In addition, it is not possible to guarantee that the effort spent to rip different pages of a given document is the same. Therefore, even distributing pages equally to each RIP, the processing units may present an unbalanced load.

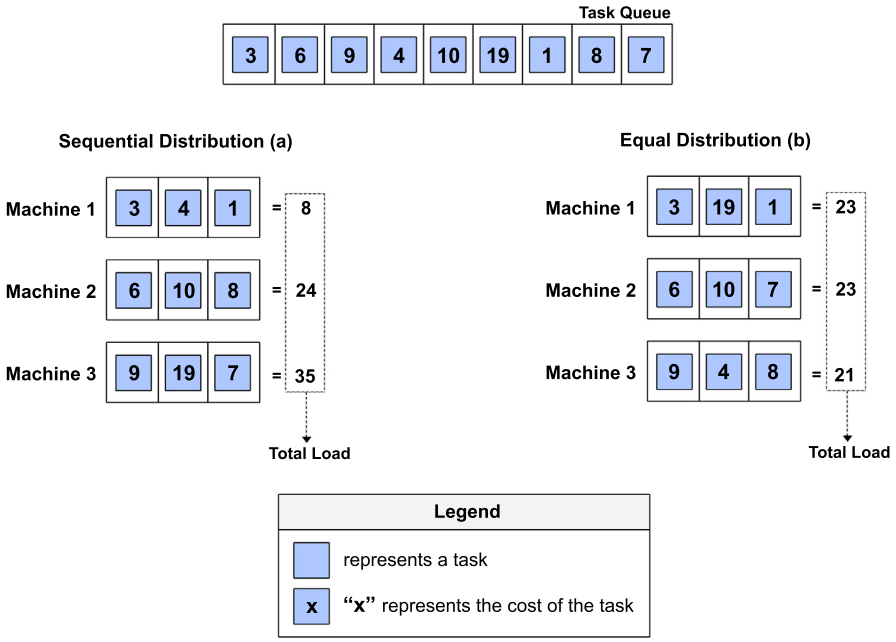

Fig. 2. Distribution example: all tasks over three processing units.

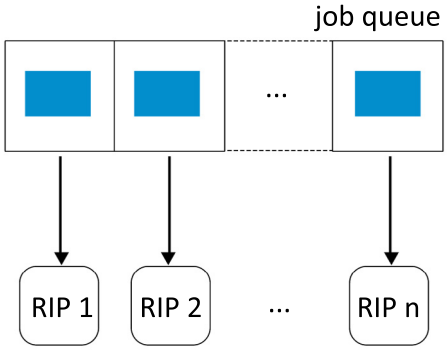

Fig. 3. Allocation of one RIP per job.

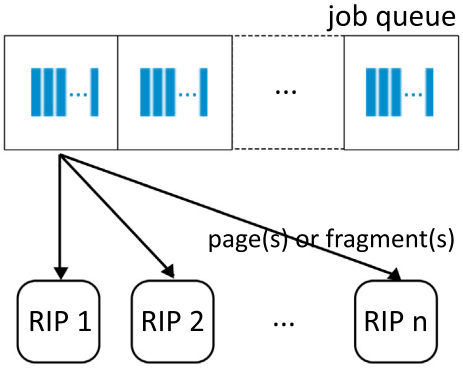

Fig. 4. Allocation of all RIPs to each job.

Finally, the third strategy suggests the use of a group of RIPs for each job in the queue. Fig. 5 illustrates this strategy considering the situation where each group includes a fixed number of three RIPs ($r = 3$). This strategy is based on the fact that "standard" PSP jobs require a specific number of resources to keep the printers working continuously. Thus, several groups of r RIPs are available to process the jobs and each job will be directed to a distinct group.

In this configuration, each job is split into r tasks by each RIP. Like any static configuration, this will only work as expected if the PSPs print jobs fit a specific profile. Otherwise, the number of allocated RIPs to a job may not be enough to keep the printers continuously fed. In addition, a major drawback in this strategy is the fact that each RIP of a group is only allocated for a new job when all RIPs in the group are free, i.e., at the time the current job is completely processed by the group. Therefore, many RIPs of a given group may be free while others of the same group did not finish computing its task underusing the available resources.

As can be observed, these three scheduling algorithms are very trivial and do not consider the computational cost to rip documents with characteristics that may be different. Next section presents a new scheduling architecture which deals with this particularity, aiming at improving the overall ripping workflow.

3. Proposed scheduling architecture

In traditional ripping, PDF documents were placed in the job queue and simply distributed among the available RIPs, with no concern about the document characteristics.

The proposed scheduling architecture is based on a module called Scheduler which was inserted between the job queue and the RIP engines. This module is responsible for distributing jobs to RIPs in a more intelligent way, in order to increase the performance and efficiency of the ripping phase.

The Scheduler is divided into three modules: PDF Splitter, PDF Profiler and Queue Manager as can be seen in Fig. 6. Section 3.1 describes these modules and their responsibilities, while Section 3.2 shows how these modules cooperate. Finally, Section 3.3 presents some scheduling algorithms we used to evaluate our architecture.

3.1. Modules

The Scheduler modules are described in this section. Sections 3.1.1–3.1.3 briefly present PDF Profiler, PDF Splitter and Queue

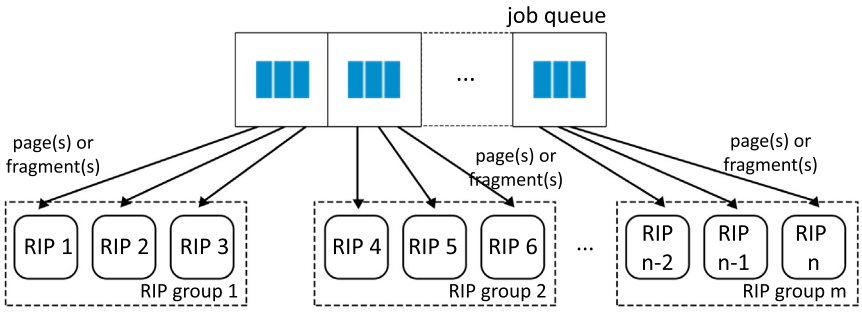

Fig. 5. Allocation of a fixed number of RIPs to each job.

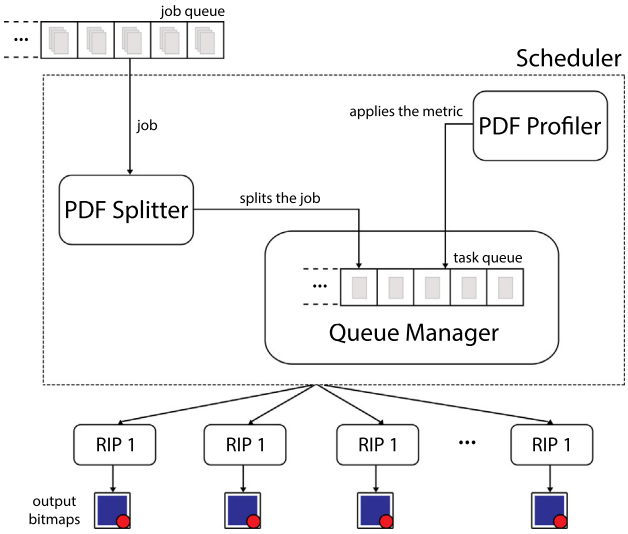

Fig. 6. Proposed architecture.

Manager, giving details about their functions in our proposed architecture.

3.1.1. PDF Profiler

PDF Profiler tool was developed in order to parse a PDF document, providing detailed information about it. This tool was implemented using the Java library PDFBox (PDFBox, 2015), which provides methods to navigate and find the necessary elements inside a PDF document. To execute the PDF Profiler, two input files are necessary: a PDF, that will be analyzed, and an XML (eXtensible Markup Language), which contains the description of the PDF file. This tool executes three main steps: (i) interpretation of the incoming XML file, identifying the data to be analyzed; (ii) search for the requested information in the PDF file, interpreting the required objects, and (iii) generation of an XML output file containing the obtained results.

Among all possible information that may be provided by PDF Profiler, some can be used to define scheduling strategies that may have an important impact on the ripping performance (Nunes, Giannetti et al., 2009). It is important to highlight that PDF Profiler tool supports the profile analysis of an entire document or several fragments (tasks) of this document separately. In the last case, despite the entire document is received as input, each fragment of this document will be parsed separately (Nunes, Giannetti et al., 2009).

Based on the XML file generated by PDF Profiler, some metrics are applied and an estimated ripping computational cost is assigned to each task. These metrics were proposed and validated in Nunes, Giannetti et al. (2009).

3.1.2. PDF Splitter

PDF Splitter tool was developed in order to split a PDF document into different fragments. In the traditional ripping process, the split was executed by the RIP engines. In the proposed scheduling architecture, a new module was developed to split documents in a clever way. Traditionally, documents were split in contiguous intervals. The PDF Splitter presents two main approaches in the splitting process: split by intervals and split by specific pages. By using the first mode, it is possible to obtain fragments that represent a division of the original document in every p pages, where p represents the desired range. The second one is a more flexible breaking mode in which it is possible to specify exactly the desired pages of the original document. Moreover, through the splitting approaches, it is possible to ignore certain pages of the PDF document and there is no need to split the entire document. This feature optimizes the performance of the splitting process, because it will not be necessary to analyze all pages of the original document to obtain the desired fragments.

Basically, PDF Splitter creates a new PDF document for each specified fragment, containing their corresponding pages. However, to finalize the new file generation, a copy of the selected pages objects must be made. PDF format is based on references, meaning that, in a given document, any object used on a page (not only graphical objects) can be used in the context of any other pages. Thus, when splitting a document into fragments, some objects need to be redefined for each new document. Fig. 7 exemplifies one situation where the Input Document has 3 pages (Page 1, Page 2 and Page 3). Page 1 defines and uses 6 objects (obj 10, obj 21, obj 29, obj 6, obj 25 and obj 32). Page 2 uses 3 objects (obj 10, obj 32 and obj 25), which do not need to be redefined, once they were already defined in Page 1 and will be reused. The same situation can be seen in Page 3, where objects will also be reused. In this example, this document will be split into 3 new ones (Doc 1, Doc 2 and Doc 3), each document with one page. Each output document must have the definition of all objects it uses, since a document cannot reuse object definitions from others.

3.1.3. Queue Manager

This section presents the Queue Manager module, that is responsible for sorting the task queue in a clever way aiming at improving the throughput of printed jobs. Each task (fragment of a document) in the queue have an associated cost, provided by PDF Profiler during the document analysis. This estimated computational cost is obtained according to a range of document information, as mentioned in Section 3.1.1.

The queue is dynamically sorted by the Queue Manager, meaning that whenever a task arrives, it is placed on the right position rather than the end of the queue. This positioning respects the scheduling policies, which varies depending on the used algorithm (scheduling algorithms used on this work are presented in Section 3.3). Whenever tasks are on their places in the queue, they

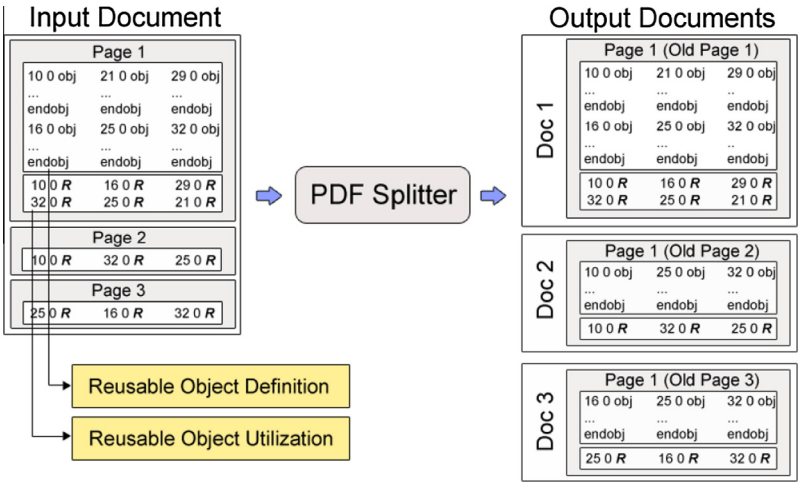

Fig. 7. Splitting example of a document with reusable objects.

are sent to the first available RIP, that is responsible for performing the ripping process before the document printing.

3.2. Scheduler

The Scheduler is the main part of the proposed workflow, controlling from the documents arrival at the job queue until the task is sent to the RIPs. Besides that, it is composed by three modules (PDF Profiler, PDF Splitter and Queue Manager, as described in Sections 3.1.1, 3.1.2, 3.1.3, respectively) that cooperate to improve the ripping phase performance. The Scheduler modularity feature allows to distribute tasks among RIPs in a smarter way. This is possible because PDF Profiler is capable of analyzing the characteristics of each document fragment, estimating the computational cost of ripping. Furthermore, in order to improve the ripping performance, the split process occurs inside the Scheduler (PDF Splitter) instead of RIP engines, unlike previous scheduling strategies described in Section 2.3.

When new PDF documents (jobs) are ready to be ripped, they are enqueued in the job queue. While there are jobs in the job queue, PDF Splitter dequeues a job and divides it in smaller parts (tasks) according to the scheduling algorithm, creating a task queue. At this moment, PDF Profiler is responsible for extracting some key information of each enqueued task aiming at calculating specific metrics, closely related to the task computational cost. After that, Queue Manager sorts the tasks in the queue by their computational costs. Thus, the Scheduler sends the tasks to the RIPs following the polices of a given scheduling algorithm.

Considering the existence of m available processing units, the scheduling algorithm A and the job queue ψ, containing n initial jobs, such that $\psi = \{job_1, job_2, \ldots, job_n\}$, the steps executed by the Scheduler are illustrated in Fig. 8 and described as follows:

1. For each job_i of queue ψ, where $1 \leqslant i \leqslant n$, Step 2 is executed.
2. Considering that job_j has a total number of pages tp_j, if tp_j is larger than m, so m fragments (or tasks) are established. In this situation, each one will contain tp_j/m pages. If the tp_j/m result is not an integer, the last fragment will receive the integer part of tp_j/m plus the remainder (e.g., if $tp_j = 10$ and $m = 3$, the first two fragments will contain 3 pages and the last one will contain 4 pages). On the other hand, if tp_j is smaller than m, only tp_j fragments are generated, each one with just one page. These fragments are created by the PDF Splitter tool.
3. Fragments are inserted into the task queue $\delta = \{T_1, T_2, \ldots, T_k\}$, in which each task T_l corresponds to a generated fragment (with $1 \leqslant l \leqslant k$).
4. Using the PDF Profiler tool, information about each T_l in δ is extracted. The metrics defined by Nunes, Raeder et al. (2009) are applied to the obtained information in order to estimate the cost associated to each analyzed task T_l.
5. Queue Manager applies scheduling directives defined in A to organize the tasks in δ.
6. Tasks are transmitted to each idle machine by the Scheduler according to algorithm A. Every time a machine is idle, the next task in the queue is transmitted for its execution.

It is important to point out that the job queue in the presented architecture is dynamic, since new input PDFs documents (jobs) may be included at any time (Step 1). Naturally, the task queue organization is also dynamic (Step 5), i.e., each time a new task is enqueued in the task queue (Step 3) the Queue Manager is responsible for dynamically organizing it. Another important remark is that in our architecture the Scheduler is responsible for splitting the jobs, unlike the previously mentioned scheduling strategies (Section 2.3), in which the split were performed by the RIPs.

Considering the generic description presented above, each scheduling algorithm A will execute different actions according to their own behavior. Section 3.3 describes the algorithms we use in Step 5.

3.3. Algorithms

The scheduling problem which arises from the Ripping phase deals with n jobs that should be scheduled to m equal machines (RIPs). With our new architecture, it is possible to evaluate the computational cost of each job. Therefore, the main goal of the scheduling problem using the new architecture is to reduce the time to complete the last task. This is a well-known problem called $Pm||C_{max}$ (Leung, 2004). This problem deals with n tasks $\delta = T_1, T_2, \ldots, T_n$ that should be scheduled to m equal machines, aiming at reducing the makespan (i.e., the time to complete the last task – C_{max}).

There are different algorithms to solve $Pm||C_{max}$ problem (Albers, 2013; Coffman, Garey, & Johnson, 1978; Dell'Amico & Martello, 1995; Friesen, 1984; Graham, 1966, 1969; Hochbaum & Shmoys, 1987). Among these, it is possible to find **Multifit** (Coffman et al., 1978) and **Largest Processing Time first** (LPT) (Graham, 1969), which were used to test the Scheduler. These algorithms were chosen because they are suitable to deal with this problem and they are also well-known in the literature.

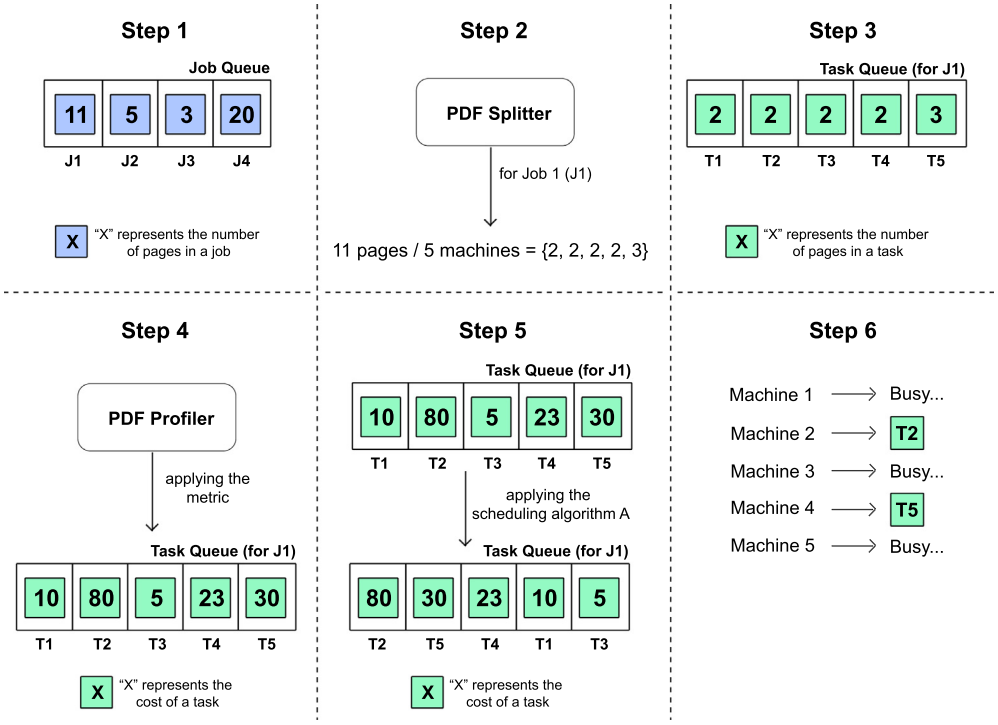

Fig. 8. Scheduler steps, considering an environment where $m = 5$.

It is important to notice that the strategies traditionally used in the ripping process do not consider the computational cost of tasks. In our context, the PDF Profiler module associates a ripping cost to all tasks, allowing the Queue Manager module to organize them according to the polices of a specific scheduling algorithm. Thus, due to our architecture it is possible to use any scheduling algorithm that needs computational cost information.

Among many possible algorithms, three algorithms were chosen to evaluate our architecture. Sections 3.3.1 and 3.3.2 present the algorithms aforementioned, discussing their advantages and drawbacks. Section 3.3.3 introduces an adaption of the LPT scheduling algorithm to better suit our context.

3.3.1. Multifit

Multifit algorithm was proposed by Johnson and is based on bin-packing techniques (Johnson, Demers, Ullman, Garey, & Graham, 1974). The algorithm proposes to pack any n tasks, with any load, in a finite number of bins (packages) in such a way that the number of used bins is as small as possible. The bin-packing problem is NP-complete (Book, 1975) and therefore several heuristics were used to try to solve this problem. In this context, Multifit algorithm uses the First-Fit Decreasing (FFD) heuristic to group the tasks up to m bins, considering that the load of a bin is the time to process all tasks inside it. The FFD strategy works as follows:

1. tasks are organized according to their processing time in a descending sequence;
2. each task is added in a given bin with the smaller load among the bins, in a way that the bin capacity does not exceed a predefined value for C;
3. Steps 1 and 2 are repeated until all tasks are in a bin.

In this scenario, aiming at finding the optimal bin-packing, C would be the optimal makespan (Coffman et al., 1978). To find a good (suboptimal) value for C in a polynomial time, iterative search methods are applied choosing an initial value for C and refining this value during the computation. Thus, for each iteration, FFD technique will be executed considering the current C. The search will end when k iterations were executed. Multifit algorithm uses the binary search method.

A value for C is obtained through the average of a lower limit (C_{low}) and an upper limit (C_{up}). These limits correspond to the best and the worst possible cases to the task packing and are defined as follows (Coffman et al., 1978):

$$C_{low} = \max\left(\frac{L}{m}, LT\right), \tag{1}$$

$$C_{up} = \max\left(\frac{2 * L}{m}, LT\right), \tag{2}$$

where L is the sum of all task loads in the queue, LT is the load of the largest task, and m is the amount of bins in which tasks were grouped into.

Fig. 9 shows an application of the Multifit algorithm. Once the initial value C is defined, the binary search for the smallest C value proceeds until k iterations are reached. In this sense, an iteration corresponds to an execution of the FFD algorithm considering the current C value. When executing each iteration, if the result of FFD algorithm is more than m bins, C_{low} will receive the current C and C_{up} will be maintained. If FFD result is up to m bins, the upper limit is refined: C_{up} will receive the current C, and the lower limit is maintained. The next iteration will proceed calculating the average between C_{low} and C_{up}. When k iterations are reached, C_{up} will contain the smaller value to C found in this binary search. If the binary search does not find any C, such that the corresponding FFD generates up to m bins, C will be the initial upper limit. In both cases through FFD application and the predetermined C, it is possible to generate up to m bins. As a final result, Multifit algorithm will present m bins with similar workloads, which could be scheduled to m machines.

In our approach, when Multifit algorithm is used in Step 5, tasks of δ are packaged in bins, through the use of FFD algorithm resulting in the creation of b bins (being $1 \leqslant b \leqslant m$). Therewith, these b

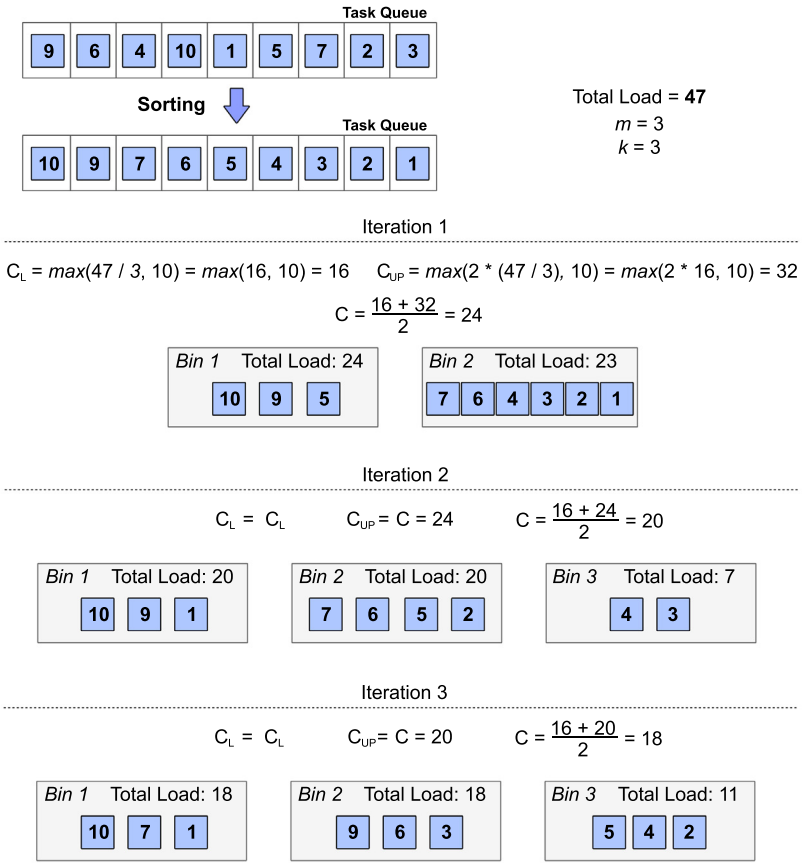

Fig. 9. Multifit algorithm.

bins are transmitted to b machines (Step 6), each machine receiving one bin.

3.3.2. Largest processing time first

Graham in the 60s (Graham, 1966) introduced an algorithm called List Scheduling (LS). This algorithm specifies that considering a queue of tasks arranged in any order, when a machine is idle, the task in the first position of the queue should be assigned to this machine. It became the basis for the development of other scheduling algorithms. Aiming at improving the competitive ratio of LS algorithm, Graham has proposed the construction of a new algorithm known as Largest Processing Time First (LPT). Graham focused on trying to avoid the LS algorithm worst case to happen, which occurs when the last task in the queue is the one with the largest processing time.

Based on that, LPT algorithm introduces the idea of sorting the tasks in descending order, making those with larger processing time to be among the first positions. After that, LS algorithm is applied with no other changes. Thus, the processing of large jobs at the end of the scheduling is avoided, obtaining a fair load distribution among the machines and reducing the performance loss (Fig. 10).

LPT algorithm is based on the idea of executing the largest tasks first. Thus, in our approach the task queue (δ) is organized in descending order (Step 5), according to the cost of each task (T_l) of the queue. So, tasks that have a larger computational cost, according to the estimated metrics, will be in the first positions of the queue. After that, in Step 6, tasks at the beginning of the queue will be transmitted to each available machine in such a way that all RIPs receive at least one task. As a machine becomes idle, the first task of the queue will be assigned to it. Meanwhile, new jobs are continuously inserted in the job queue (ψ) and, for each new job, Steps 2–4 will be performed. It means that this new job will be broken in tasks and each one will be inserted in δ. However, this insertion will be carried out in a way that keeps the queue in descending order of workload, according to the cost of each task. Thus, it will be avoided the necessity of reorganize the queue each time new tasks have to be inserted in δ.

3.3.3. Optimized LPT

The biggest disadvantage of the LPT algorithm is the need for evaluating the computational task cost to insert them in the queue and make them available to the RIPs. This evaluation is performed by the PDF Profiler. Despite the fact that it does not necessarily represent a large computational cost, it may not allow the immediate transfer of tasks to idle RIPs. The Scheduler may be busy with the profiling of tasks at the time that there are free RIPs waiting to receive more tasks. Such situation will compromise the performance gain obtained by the use of LPT. Thus, we proposed a slight modification on LPT in order to reduce this overhead, creating a new algorithm: the Optimized LPT.

The main goal of Optimized LPT is to reduce the Scheduler response time. Thus, it was established that the Scheduler should apply the metrics on the tasks concurrently with the scheduling of tasks to free RIPs. To accomplish that, as soon as the Scheduler needs to apply the metric on a particular task, a new thread responsible for that function is created. Another important modification is that before the creation of this thread, the tasks are inserted at the end of the task queue. As the threads finish their executions, the computational cost of the corresponding tasks will

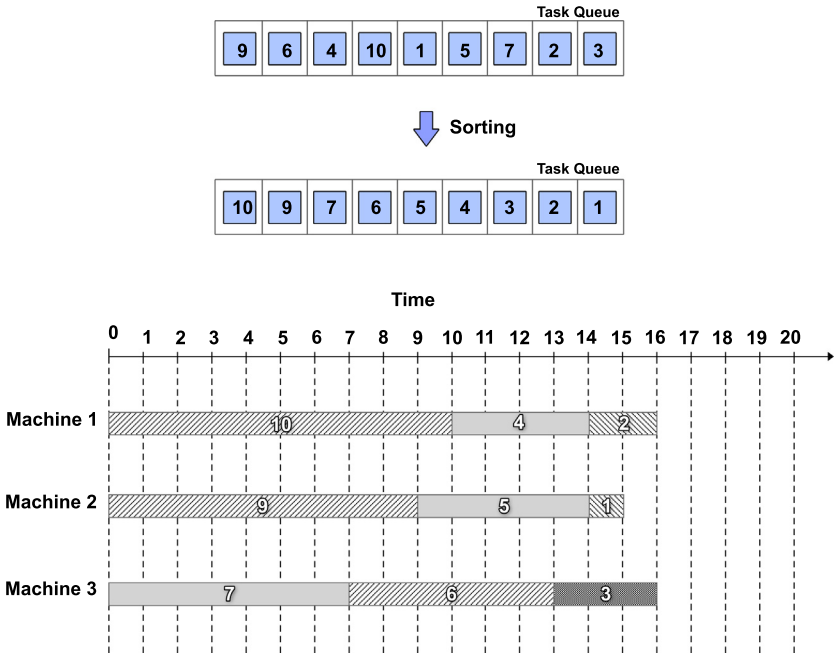

Fig. 10. LPT algorithm.

be updated by removing them from the queue and inserting them back in the correct position according to its cost. Through this strategy, if a RIP is free and there are no more tasks in the queue except those for which the computational cost has not been computed, one of these tasks will be sent to the machine anyway and the machine will not be idle.

4. Scenario configuration

This section presents two main aspects of the scenario configuration used to test our new architecture: input documents and job queue configurations. Section 4.1 illustrates some input documents that are handled in PSPs environments. These documents were used to populate the queues in our test cases. Section 4.2 describes three configurations for the job queue, helping us evaluating the proposed scheduling algorithms.

4.1. Input documents

The jobs used in our experiments are composed of 8 different input document types, as presented by Fernandes et al. (2012). These documents are strongly based on real cases and commonly used within PSPs. An example of each document is shown in Fig. 11 and a brief description of their characteristics is presented as follows:

1. **Letter:** contains one or more pages with a large amount of texts and fewer pictures.
2. **Newsletter:** unlike Letter, a newsletter contains several images, which are used for advertisements and news on products and services.
3. **Cards:** it contains background images and little text.
4. **Postcard:** it is used for the exhibition of a place, product or service. Thus, texts and images are used to compose each page.
5. **Flyer:** used to present and advertise products, ideas, events, among others. In this case, images and texts are used.
6. **Brochure:** usually consists of a large number of pages with large amount of images and text.
7. **Newspaper:** contains information and news headlines. This document presents both texts and images.
8. **Poster:** typically, it consists of a large size page. A poster can contain only images, only texts or both images and text.

For our experiments, 20 customer jobs used in PSPs were created through a combination of these 8 types of input documents. These jobs contain text, images and other PDF graphical objects. The full list of jobs is presented in Table 1 along with the documents most significant characteristics (like number of pages, number of images and computational cost, for example).

As it can be observed, the chosen documents have a large variability in number of pages, text and images. The goal is to make experiments with documents within a wide range of ripping costs. This high variability reflects the reality of most small to mid sized PSPs in which jobs come from various customers.

4.2. Job queue

The jobs order in the queue may have an important impact on performance. Thus, it is interesting to perform executions of different strategies on different queue configurations. Using this approach, it will be possible to evaluate how the scheduling algorithm will impact on the architecture general performance.

In a real PSP scenario, jobs may be available at different moments. Aiming at reproducing this scenario, it was established that some jobs are initially available in the queue (called initial jobs) and others will be included after x seconds (late jobs), where $10 \leqslant x \leqslant 60$. Using this small range, new tasks will be available after few seconds, making it possible to simulate how each strategy addresses the profile analysis, organization and distribution of jobs concurrently. Thus, three different queue configuration were defined as follows:

1. **Queue 1:** job queue randomly organized. In this case, jobs computational cost may vary from the first to the last job of the queue (Table 2(a)).

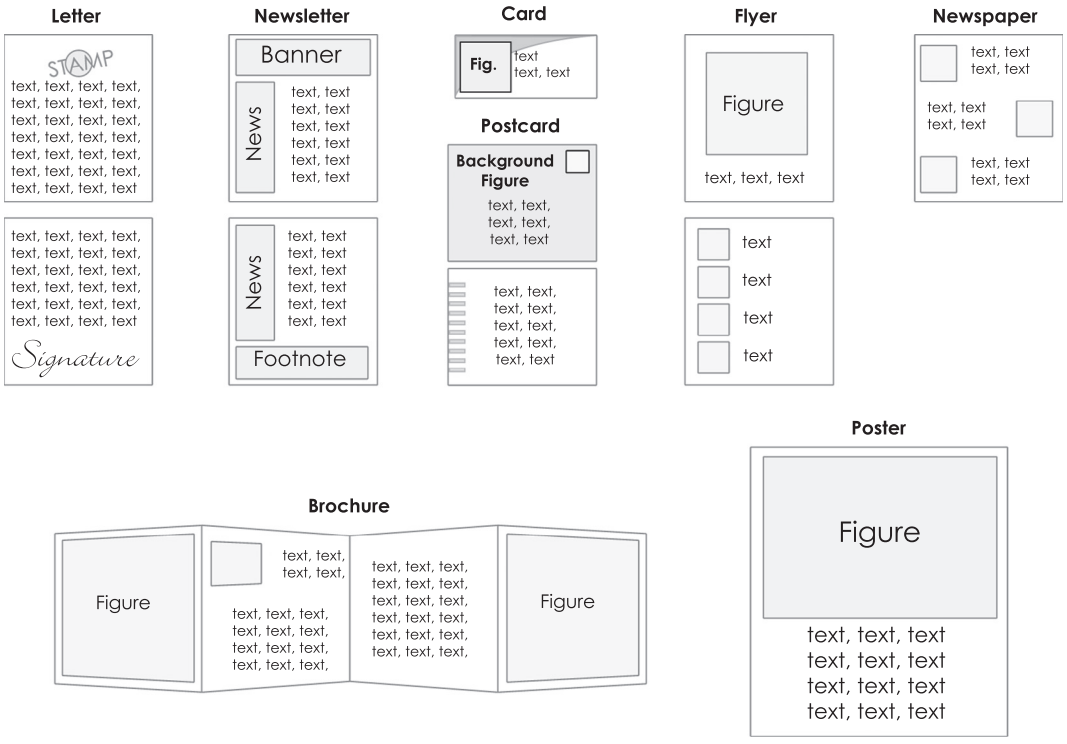

Fig. 11. Examples of input documents.

Table 1
Job composition.

Job	Documents	Pages	Images	Pages with text	Computational cost
Letter 1	130	390	130	390	9.4743
Letter 2	400	800	800	800	21.055
Letter 3	90	450	0	450	9.1054
Card 1	400	200	800	200	9.4547
Card 2	40	40	40	40	0.5681
Newsletter 1	220	440	3740	440	18.7140
Newsletter 2	170	340	680	340	9.1560
Postcard 1	500	250	4000	250	10.405
Postcard 2	20	40	120	40	0.6183
Flyer 1	80	160	320	160	4.2587
Flyer 2	100	200	400	200	4.4444
Flyer 3	250	500	750	500	10.2770
Flyer 4	75	150	150	150	6.6435
Brochure 1	9	108	162	108	1.7586
Brochure 2	200	1000	1800	1000	23.4190
Brochure 3	300	600	2700	600	13.3400
Newspaper 1	400	400	2000	400	12.8990
Newspaper 2	150	900	2700	900	23.3320
Poster 1	500	500	2000	500	10.2560
Poster 2	70	70	140	70	1.7707

2. **Queue 2:** job queue sorted in descending order, from the largest to the smallest computational cost (Table 2(b)).
3. **Queue 3:** job queue sorted in ascending order, from the smallest to the largest computational cost (Table 2(c)).

These configurations were defined in order to evaluate our Scheduler regarding the jobs loads variations. Moreover, the queues aim at verifying whether the proposed architecture is able to handle this type of irregularity on load balance among the available RIPs. Furthermore, it became possible to estimate the impact of these variations in the scheduling strategies that already exist. It is important to highlight that queues 2 and 3 are extreme situations which will not be common in the context of PSPs.

5. Experimental results

This section presents experimental results obtained through the use of our proposed architecture. Section 5.1 specifies the environment used for performing the tests. A detailed performance analysis will be presented in Section 5.2.

5.1. Test environment

The environment is composed of a cluster of blades, which is a multicomputer architecture classified as a COW (Cluster Of Workstations). The used cluster contains 5 blades and each one has an Intel Xeon 3.0 GHz (64 bits) quad-core processor, with 8 GB of RAM memory and 400 GB of disk. Each core is considered as an active processing unit of the architecture, thus the architecture supports up to 20 processes simultaneously (four by blade). Moreover, these blades are connected via a high speed Gigabit Ethernet network. The operating system used on the machines is Windows Server 2003 R2 Standard x64 Edition, with Service Pack 2, a system commonly used in printing environments due to its compatibility with various versions of industrial RIPs.

Java language was chosen to implement the previously described scheduling strategies. Despite the fact that the use of Java is not usual in high performance area, such language has been adopted by some researchers for their implementations (Getov, Hummel, & Mintchev, 1998, for instance). Furthermore, this language offers advantages in portability, extensibility and high abstraction level. In special, it offers compatibility with the PDFBox Java library, used in the PDF Profiler. For the implementation, Java JDK 1.6 was used. Thus, due to the multicomputer architecture previously mentioned, communication among processes needs to be carried out through the message passing paradigm.

Table 2
Job queue configurations.

Position	Job	Type	Delay (s)
(a) Queue 1			
1	Brochure 1	Initial	–
2	Brochure 2	Initial	–
3	Brochure 3	Initial	–
4	Card 1	Initial	–
5	Card 2	Late	60
6	Flyer 1	Late	50
7	Flyer 2	Late	40
8	Flyer 3	Late	30
9	Flyer 4	Late	20
10	Letter 1	Late	10
11	Letter 2	Late	20
12	Letter 3	Late	30
13	Newsletter 1	Late	40
14	Newsletter 2	Late	50
15	Newspaper 1	Late	60
16	Newspaper 2	Late	30
17	Postcard 1	Late	40
18	Postcard 2	Late	20
19	Poster 1	Late	50
20	Poster 2	Late	60
(b) Queue 2			
1	Brochure 2	Initial	–
2	Newspaper 2	Initial	–
3	Letter 2	Initial	–
4	Newsletter 1	Late	40
5	Brochure 3	Late	10
6	Newspaper 1	Late	60
7	Postcard 1	Late	40
8	Flyer 3	Late	30
9	Poster 1	Late	60
10	Letter 1	Late	10
11	Card 1	Late	30
12	Newsletter 2	Late	50
13	Letter 3	Late	30
14	Flyer 4	Late	20
15	Flyer 2	Late	40
16	Flyer 1	Late	50
17	Poster 2	Late	50
18	Brochure 1	Late	10
19	Postcard 2	Late	20
20	Card 2	Late	60
(c) Queue 3			
1	Card 2	Initial	–
2	Postcard 2	Initial	–
3	Brochure 1	Initial	–
4	Poster 2	Initial	–
5	Flyer 1	Initial	–
6	Flyer 2	Initial	–
7	Flyer 4	Initial	–
8	Letter 3	Late	30
9	Newsletter 2	Late	50
10	Card 1	Late	30
11	Letter 1	Late	10
12	Poster 1	Late	60
13	Flyer 3	Late	30
14	Postcard 1	Late	40
15	Newspaper 1	Late	60
16	Brochure 3	Late	10
17	Newsletter 1	Late	40
18	Letter 2	Late	20
19	Newspaper 2	Late	50
20	Brochure 2	Late	10

In order to implement communication, the MPJ-Express (MPJ-Express, 2015) library was used. This library is an implementation of the MPI (Message Passing Interface) standard (Snir, Otto, Huss-Lederman, Walker, & Dongarra, 1996) using Java language. Due to the use of Java, this library provides high-level concepts of abstraction and portability, which are very important characteristic for PSPs. In terms of performance, MPJ-Express can obtain similar results when compared to the libraries like MPICH (MPI Chameleon) (MPICH, 2015) and LAM/MPI (Local Area Multicomputer/MPI) (LAM/MPI, 2015).

5.2. Scheduler architecture performance analysis

This section discusses performance measurements for our architecture using the three algorithms previously described (Section 3.3), comparing them to the three traditional scheduling strategies for ripping (Section 2.3), resulting in six test cases. For each test case 10 executions were performed over 3 different queue settings. In this context, an execution means to rip all jobs (documents) in a specific queue according to a strategy. From these results, the larger and the smaller execution time for each test battery were excluded and an average of the other 8 remaining values was computed.

It is important to emphasize that for all strategies, a processor should be reserved to run the Scheduler. This processor will be responsible for dealing with the whole scheduling procedure. Thus, the existence of parallel RIPs depends on a minimum number of 3 processes (2 RIPs and one Scheduler). Hence, the amount of processes in the tests executions ranged from 3 to 20, except for N RIPs per job existing strategy. In this case, four distinct values were assigned to *N*: 2, 3, 4 and 5 RIPs per job. In this sense, the number of processes used to test the N RIPs per job strategy was stipulated to avoid idle processes. For example, using 2 RIPs per job over 3 processes, one process is the Scheduler and the other are a group of 2 RIP engines. However, if 2 RIPs per job strategy were performed over 4 processes, one process would be the Scheduler, two processes would be a group of 2 RIP engines and the third one would be idle. Therefore, the *N* RIPs per job strategy was only tested for certain number of processes.

Initially, speed-up values for the six test cases were obtained using Queue 1. In this scenario, the queue is randomly organized, meaning that jobs with different loads are mixed. Table 3 describes acceleration factors obtained specifically for N RIPs per job strategy varying the number of processes. Fig. 12 presents the other strategies speed-ups in a comparative graph also varying the number of processes (from 3 to 20).

As can be seen in Fig. 12, the 1 RIP per job strategy speed-ups does not scale well. In fact, the speed-up remains around a factor 3 for different number of processes. This behavior can be explained since we use more ripping processing units, each RIP receives few jobs to compute (at least one). In this scenario, one of the jobs assigned to a single RIP will be the last one to finish processing, while several RIPs are idle. Thus, the addition of more processes will not represent an effective gain in this strategy, since several processing units will remain idle waiting for more non-existent jobs to come.

Table 3
Queue 1 speed-ups: *N* RIPs per job.

RIPs per job	Number of processes						
		3	4	5	6	7	9
2		1.42	–	2.32	–	2.57	2.77
3		–	2.66	–	–	4.72	–
4		–	–	3.74	–	–	4.15
5		–	–	–	4.26	–	–
	10	11	13	15	16	17	19
2	–	3.17	2.84	2.97	–	2.77	2.98
3	4.96	–	4.99	–	5.20	–	4.99
4	–	–	3.91	–	–	3.47	–
5	–	3.72	–	–	3.54	–	–

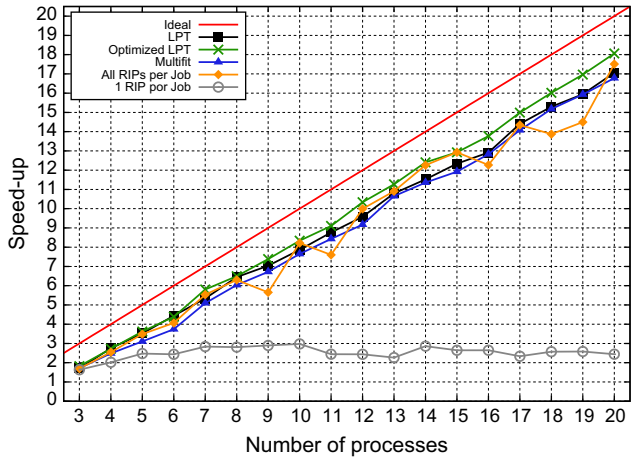

Fig. 12. Speed-ups for Queue 1.

The obtained speed-ups for *N* RIPs per job strategy are also not satisfactory. In Table 3 it can be seen that there is a speed-up fluctuation in the measurements. It is possible to notice that the best speed-up is 5.2 over 16 processes (corresponding to 3 RIPs per job). These situation can be explained by the fact that this strategy leads to an under use of RIPs and an unfair load balance, because RIPs of the same group may receive tasks with very different sizes, so that those who did not finish their process will prevent others to receive new tasks. Thus, the execution time is slowed down since some jobs remain waiting in the queue even when some ripping units are idle.

Among the traditionally used ripping strategies, the all RIPs per job strategy presented the best results. Using this approach, a speed-up of 17.51 was obtained using 20 processes. However, in its performance curve, fluctuations generating unpredictable situations and unsatisfactory acceleration factors for certain numbers of processes can be seen. This strategy breaks the jobs into fragments (tasks), reducing the size of the grain to be computed for each RIP. Each task is then transmitted to idle RIPs as they become available. However, no consideration about the computation cost of the task that is being transmitted is taken. Thus, this distribution becomes susceptible to overload and underload of the RIPs, generating the presented behavior.

Analyzing the results for our proposed architecture, we can notice that the large fluctuations in the results were eliminated. Our Scheduler considers the tasks load before distribute them in order to balance the computational effort to be spent by each available processing units. The speed-ups obtained by *LPT* and *Multifit* algorithms although constant, are in many cases lower than the one obtained by all RIPs per job strategy. Possibly this situation occurs since these scheduling algorithms need to perform a profile analysis before including the jobs in the queue. Thus, if there are idle RIPs during the analysis of one or more jobs, they should wait until the profile analysis is completed to receive a new task. The overhead generated by this situation affects the speed-up curve. Moreover, in most cases *Multifit* speed-up is below the *LPT* speed-up. This may occur because the *Multifit* algorithm adds even more overhead when packaging tasks in bins.

Finally, the best results were obtained by the *Optimized LPT* algorithm. It presents less fluctuations and reaches speed-ups that overcome all test cases. The main advantage offered by the use of this algorithm in our architecture is to allow the Scheduler to transmit immediately tasks to idle RIPs concurrently to the analysis of the jobs profile. It reduces the *LPT* inherent overhead, because it always analyzes tasks before the Scheduler sending them to be ripped.

The use of different approaches affects not only the speed-up, but also the percentage of utilization of each processing unit. To analyze this measurement, the efficiency obtained for each test case is shown in Table 4. Observing these results, it is possible to notice that the efficiency is also affected by the number of processes. The 1 RIP per job strategy clearly presents a loss in efficiency as more processes are used since some RIPs are idle during the execution. The same behavior can be noticed for the N RIPs per job strategy, where the addition of more processes does not represent an effective performance gain consequently reducing the efficiency. Efficiency for all RIPs per job strategy presents a non-uniform behavior. Sometimes reaching a very high efficiency (around 90%) and sometimes presenting a regular efficiency (70–85%). Our architecture, on the other hand, presents uniform high efficiencies, even with a larger number of processes. The algorithms used with our scheduling architecture presented an average efficiency of 84% using *Multifit*, 88% using *LPT* and 92% using *Optimized LPT*.

Table 5 and Fig. 13 present speed-up results for Queue 2, in which job loads are sorted from the largest to the smallest. The *N* RIPs per job strategy does not present any significant changes in

Table 5
Queue 2 speed-ups: *N* RIPs per job.

RIPs per job	Number of processes					
	3	4	5	6	7	9
2	1.38	–	2.66	–	3.07	3.05
3	–	2.64	–	–	4.50	–
4	–	–	3.54	–	–	3.66
5	–	–	–	4.38	–	–
10	11	13	15	16	17	19
2 –	2.98	3.40	3.13	–	3.04	3.04
3 5.12	–	5.01	–	3.82	–	4.29
4 –	–	3.59	–	–	3.63	–
5 –	3.98	–	–	3.73	–	–

Table 4
Efficiency for Queue 1.

Strategy	Number of processes																	
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1 RIP per 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 per job	0.71	–	0.58	–	0.42	–	0.34	–	0.31	–	0.23	–	0.21	–	0.17	–	0.16	–
3 RIPs per job	–	0.88	–	–	0.78	–	–	0.55	–	–	0.41	–	–	0.34	–	–	0.26	–
4 RIPs per job	–	–	0.93	–	–	–	0.51	–	–	–	0.32	–	–	–	0.21	–	–	–
5 RIPs per job	–	–	–	0.85	–	–	–	–	0.37	–	–	–	–	0.23	–	–	–	–
All RIPs per 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
Optimized LPT	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

Table 6
Efficiency for Queue 2.

Strategy	Number of processes																	
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1 RIP per 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 per job	0.69	–	0.66	–	0.51	–	0.38	–	0.29	–	0.28	–	0.22	–	0.19	–	0.16	–
3 RIPs per job	–	0.88	–	–	0.75	–	–	0.56	–	–	0.41	–	–	0.25	–	–	0.23	–
4 RIPs per job	–	–	0.88	–	–	–	0.45	–	–	–	0.29	–	–	–	0.22	–	–	–
5 RIPs per job	–	–	–	0.87	–	–	–	–	0.39	–	–	–	–	0.24	–	–	–	–
All RIPs per 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
Optimized LPT	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

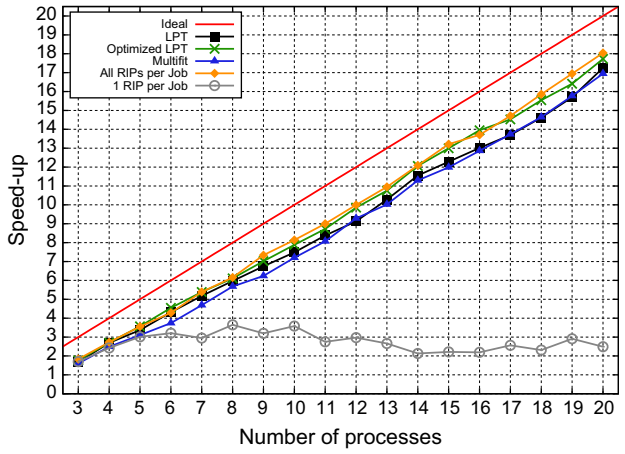

Fig. 13. Speed-ups for Queue 2.

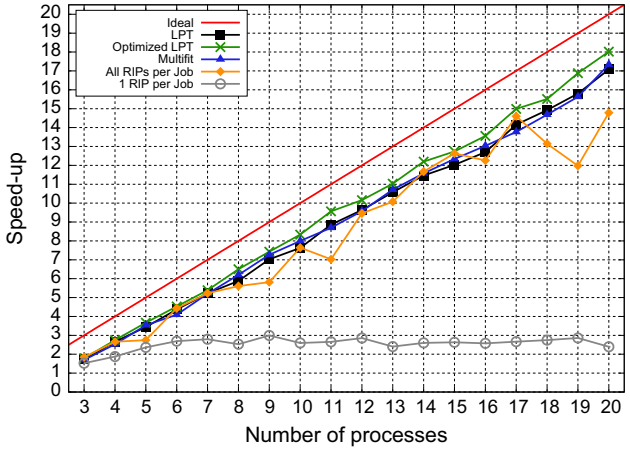

Fig. 14. Speed-ups for Queue 3.

its performance. The existence of idle RIPs remains a problem slowing down the performance, resulting in low acceleration factors. This behavior can be observed for all analyzed group sizes. For the 1 RIP per job strategy, results presented a slight increase in the acceleration factor compared to the results obtained for Queue 1. However, the speed-up remains very low for a larger number of processes. This might happen because some RIPs are idle during the ripping process. Another important aspect is that the results present a fluctuation, probably due to a large grain size that can generate overload and underload of the RIPs.

Table 7
Queue 3 speed-ups: *N* RIPs per job.

RIPs per job	Number of processes						
		3	4	5	6	7	9
2		1.61	–	2.54	–	3.09	3.36
3		–	2.57	–	–	4.43	–
4		–	–	3.47	–	–	3.82
5		–	–	–	4.21	–	–
	10	11	13	15	16	17	19
2	–	3.04	2.96	3.04	–	2.78	2.99
3	4.51	–	5.04	–	4.60	–	4.56
4	–	–	3.44	–	–	4.25	–
5	–	3.95	–	–	3.50	–	–

Observing all RIPs per job strategy speed-up, no significant variations can be seen. Using Queue 2, it was possible to distribute first the heavier tasks, reducing the RIPs overload factor at the end of the processing. As the queue in this case is already sorted, LPT algorithm would be supposed to produce the best results among all others. However, the acceleration factors for *LPT* and *Multifit* algorithms are smaller than the one achieved when using all RIPs per job strategy. The possible explanation for that is the need to analyze each task even when the task queue is organized in descending order of ripping cost, condition that generates overhead. Furthermore, analyzing the Optimized LPT algorithm, obtained results do not deviate much from those obtained by all RIPs per job strategy, due to its ability of immediately transmitting tasks to idle RIPs. Anyway, a small overhead is observed possibly by the thread that analyzes the job profile.

In the context of Queue 2, efficiency values are presented in Table 6. As expected, the efficiency of 1 RIP per job and N RIPs per job strategies did not increase as more processing units were added, due to the low scalability of the application. On the other hand, the efficiency of all RIPs per job strategy was high, with an average of 92%. Considering the algorithms we used in our architecture, average efficiency was 82%, 86% and 90% to *LPT*, *Multifit* and *Optimized LPT* respectively. It is possible to notice that *LPT* and *Multifit* efficiency are not as good as the one obtained by all RIPs per job strategy. However, the efficiency obtained using the *Optimized LPT* algorithm is very similar to that obtained by all RIPs per job strategy, since this approach does not suffer too much with the impact of unnecessary job analysis overload.

Fig. 14 and Table 7 present the results for Queue 3. In this queue, jobs are sorted in ascending order of workload. As can be noticed, the speed-ups obtained using our architecture keep close to the ideal regardless the algorithm used. In this context, Optimized LPT presents the best speed-up among all test cases. LPT and Multifit results were mainly affected by the job profiling

Table 8
Efficiency for Queue 3.

Strategy	Number of processes																	
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1 RIP per 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 per job	0.80	–	0.63	–	0.51	–	0.42	–	0.30	–	0.24	–	0.21	–	0.17	–	0.16	–
3 RIPs per job	–	0.85	–	–	0.73	–	–	0.50	–	–	0.42	–	–	0.30	–	–	0.25	–
4 RIPs per job	–	–	0.86	–	–	–	0.47	–	–	–	0.28	–	–	–	0.26	–	–	–
5 RIPs per job	–	–	–	0.84	–	–	–	–	0.39	–	–	–	–	0.23	–	–	–	–
All RIPs per 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
Optimized LPT	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

overhead, that retards the task distribution. Nevertheless, in most cases, these two algorithms match or surpass the traditional strategies speed-ups.

Considering 1 RIP per job and N RIPs per job strategies, no changes in previous behavior can be observed. These strategies present a low scalability, with no significant gain as more processes are added. Moreover, the all RIPs per job strategy presented a peculiar behavior with non-uniform performance. This may be explained by the order that the jobs appear at the queue. Large jobs will be inserted in the end, increasing the overload and underload among RIPs. This effect can be seen when dealing with a queue that has some small jobs and a special large job (that will be the last one to be ripped). In this case, an important loss in performance will occur since many RIPs will be idle waiting for one RIP to finish the last task. On the other hand, when dealing with jobs containing similar sizes, a performance loss will not occur since the load will be balanced among RIPs.

Finally, Table 8 shows efficiency values for the third queue configuration. As expected, 1 RIP per job and N RIPs per job strategies present lower efficiencies when more processes are included. All RIPs per job strategy presents a wide variation in the achieved efficiencies, starting from 66% (19 processes) up to 92% (3 processes). Due to this situation, the average efficiency of this strategy was around 82%. For the algorithms used in our architecture, high average efficiencies can be observed, such as 87% for *LPT* and *Multifit*, and 92% for *Optimized LPT*, which outperforms all other strategies.

6. Conclusion and future work

This paper presented a job profile oriented scheduling architecture for improving the throughput of the ripping process on industrial printing environments. Through the use of specific metrics that consider each specific job characteristics, we estimate the computational cost of the ripping process using the PDF Profiler tool. Since we are able to use the information about the computational cost in the scheduling process, any makespan scheduling algorithm can be used along with our architecture. Therefore we tested different scheduling algorithms, including those that consider the computational time. In this context, the LPT algorithm was adapted to obtain better results based on the particularities of the ripping process. Our architecture was then used to distribute the load among the available RIP engines in a clever way aiming at an optimized utilization of the available resources.

The results obtained using our proposed architecture presented better load balancing, with a better performance than the ones provided by traditional algorithms. Among traditional algorithms, that do not consider the computational time, the all rips per job strategy presented the best results, though depending on the queue configuration. Three algorithms (Multifit, LPT and Optimized LPT) which use computational cost information were used to test our architecture and presented satisfactory results regardless the input queue configuration. Optimized LPT presented the best performance, with an average efficiency of 91.3% and speed-ups around 18 when using 20 processors. In some cases it is possible to notice a similar behavior between the all RIPs per job strategy and Optimized LPT. However, while Optimized LPT keeps the efficiency and speed-up for the 3 tested queues, the all rips per job strategy presented a poor performance when the queue is sorted in ascending order.

Despite the performance gain obtained by the job profile oriented scheduling architecture using the tested algorithms, it is also important to highlight that this is an adaptive environment, which allows the use of any scheduling algorithm and enables the creation of new ones. As future work, the study of other scheduling strategies appears as an interesting path to follow, allowing the optimization of the ripping process in different ways whenever jobs details are previously known. For example, a RIP engine can be configured to deal with specific characteristics (e.g., reusability) that are presented in a given job. Based on that, RIP engines could be configured to deal with jobs that present those specific characteristics. This specialization may improve the performance and throughput of ripped jobs.

A job characteristic that might have an important influence on the computational cost is the amount of reusable objects. If it is known that a job presents many reusable objects, we may configure the RIP engine to prioritize the storage of the rasterized version of these objects in their cache and in some auxiliary storage (such as a disk or larger memory) in such a way that they can be used later. However, if RIPs are configured to process jobs with high reusability and the job objects are used only once, store operations become expensive and may compromise the performance of the ripping process in general.

Since jobs usually present a mixed of reusable and non-reusable objects, it is hard to establish when this option should be activated. Due to this situation, it would be interesting to split jobs in an intelligent way, so that pages containing the same reusable objects can be assigned to the same RIP, which will be configured to deal with high reusability. In this case, pages that do not have reusable objects in common with other pages can be assigned to RIPs that do not use reusability optimization.

Another possible optimization in the scheduling algorithm is related to job transparency. As well as reusable objects, RIPs can identify if a PDF document contains several transparent objects applying then a specific process for ripping those objects. However, if this optimization is used and the PDF document has too many opaque objects, the performance of the ripping process is compromised. Therefore, if this optimization is activated, processing opaque objects as transparent objects will result in an additional processing cost. In this sense, pages with transparent objects should be sent to RIPs that have the transparency optimization activated.

Based on the promising performance obtained with our job profile oriented scheduling architecture and the idea to optimize the

ripping process depending on specific document characteristics (such as reusability and transparency), we believe that the next step of our research is to incorporate those characteristics in our scheduling architecture. For that, we should (i) modify the PDF Profiler to identify transparent and reusable objects in documents, (ii) adapt the PDF Splitter to perform a more clever separation of the jobs pages considering the new information and (iii) create new scheduling algorithms that would benefit from this information as it becomes available.

References

Adobe Systems (2003). *PDF reference* (4th ed.). San Jose: Adobe Systems Incorporated.

Albers, S. (2013). Recent advances for a classical scheduling problem. *ICALP'13: Proceedings of the 40th international conference on automata, languages, and programming* (Vol. 2, pp. 4–14). Riga, Latvia: Springer.

Book, R. V. (1975). Reducibility among combinatorial problems. *The Journal of Symbolic Logic, 40*(4), 618–619.

Coffman, E. G., Garey, M. R., & Johnson, D. S. (1978). An application of bin-packing to multiprocessor scheduling. *SIAM Journal of Computing, 7*(7), 1–17.

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.

Dell'Amico, M., & Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing, 7*(2), 191–200.

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

Friesen, D. K. (1984). Tighter bounds for the multifit processor scheduling algorithm. *SIAM Journal of Computing, 13*(1), 170–181.

Getov, V., Hummel, S. F., & Mintchev, S. (1998). High-performance parallel programming in Java: Exploiting native libraries. *Concurrency: Practice and Experience, 10*(11), 863–872.

Giannetti, F. (2007). A multi-format variable data template wrapper extending Podis PPML-T standard. In *DocEng'07: Proceedings of the 7th ACM symposium on document engineering* (pp. 37–43). Winnipeg, Canada: ACM.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal, 45*(9), 1563–1581.

Graham, R. L. (1969). Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics, 17*(2), 416–429.

Hochbaum, D. S., & Shmoys, D. B. (1987). Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of ACM, 34*(1), 144–162.

Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., & Graham, R. L. (1974). Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing, 3*(4), 299–325.

Kruatrachue, B., & Lewis, T. (1988). Grain size determination for parallel processing. *IEEE Software, 5*(1), 23–32.

LAM/MPI (2015). Lam/mpi home page <http://www.dcs.ed.ac.uk/home/trollius/www.osc.edu/lam.html> Last access 16.07.2015.

Leung, J. (Ed.). (2004). *Handbook of scheduling: Algorithms, models, and performance analysis*. Boca Raton, USA: CRC Press, Inc..

MPICH (2015). Mpich home page <https://www.mpich.org/> Last access 16.07.2015.

MPJ-Express (2015). Mpj-express home page <http://sourceforge.net/projects/mpjexpress/files/>. Last access 16.07.2015.

Nunes, T., Fernandes, L. G., Giannetti, F., Cabeda, A., Raeder, M., & Bedin, G. (2007). An improved parallel XSL-FO rendering for personalized documents. In *14th Euro PVM/MPI – European PVM/MPI users group meeting* (pp. 56–63). Berlin/Heidelberg, Paris, France: Springer.

Nunes, T., Giannetti, F., Fernandes, L. G., Timmers, R., Raeder, M., & Castro, M. (2006). High performance XSL-FO rendering for variable data printing. In *SAC '06: Proceedings of the 2006 ACM symposium on applied computing* (pp. 811–817). Dijon, France: ACM.

Nunes, T., Giannetti, F., Kolberg, M., Nemetz, R., Cabeda, A., & Fernandes, L. G. (2009). Job profiling in high performance printing. In *DocEng'09: Proceedings of the 9th ACM symposium on document engineering* (pp. 109–118). New York, NY, USA: ACM.

Nunes, T., Raeder, M., Kolberg, M. L., Fernandes, L. G., Cabeda, A., & Giannetti, F. (2009). High performance printing: Increasing personalized documents rendering through PPML jobs profiling and scheduling. In *CSE'09: Proceedings of the 12th IEEE international conference on computational science and engineering* (pp. 285–291). IEEE Computer Society.

PDFBox (2015). PDFBox home page <http://pdfbox.apache.org> Last access 16.07.2015.

Purvis, L., Harrington, S., O'Sullivan, B., & Freuder, E. C. (2003). Creating personalized documents: an optimization approach. In *Doc- Eng '03: Proceedings of the 2003 ACM symposium on document engineering* (pp. 68–77). Grenoble, France: ACM.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., & Dongarra, J. (1996). *MPI: The complete reference*. Cambridge, USA: MIT Press.