Scheduling algorithms for distributed cosmic ray detection using Apache Mesos

Germán Schnyder[1], Sergio Nesmachnow[1]
Gonzalo Tancredi[1], and Andrei Tchernykh[2]

[1] Universidad de la República,
Montevideo, Uruguay
{german.schnyder,sergion}@fing.edu.uy,gonzalo@fisica.edu.uy
[2] CICESE Research Center,
Ensenada, Baja California, México
chernykh@cicese.mx

Abstract. This article presents two scheduling algorithms applied to the processing of astronomical images to detect cosmic rays on distributed memory high performance computing systems. We extend our previous article that proposed a parallel approach to improve processing times on image analysis using the Image Reduction and Analysis Facility IRAF software and the Docker project over Apache Mesos. By default, Mesos introduces a simple list scheduling algorithm where the first available task is assigned to the first available processor. On this paper we propose two alternatives for reordering the tasks allocation in order to improve the computational efficiency. The main results show that it is possible to reduce the makespan getting a speedup = 4.31 by adjusting how jobs are assigned and using Uniform processors.

Keywords: Image processing, Distributed Memory, Containers, Mesos, Scheduling

1 Introduction

Hubble Space Telescope (HST) is not only an astronomical observatory but also an excellent cosmic ray detector. Because HST is above the Earth's atmosphere and therefore not protected against low-energy cosmic rays, and crosses the Van Allen radiation belts, HST is an unique particle detector. Since cosmic ray flux is affected by the strength of the magnetic field, detectors of HST detectors sample different conditions of the magnetic field, which can be used to compare to magnetic field strength, gamma ray flux and other geophysical data measured by the geomagnetic observatories. HST dark frames (or just *darks*) are suitable for cosmic ray studies because they are acquired with closed shutters so only cosmic ray events are recorded. HST results will complement that of the existing cosmic rays detectors on ground and space. Launch of the HST predate geomagnetic satellites by more than a decade; its 26 years of low altitude cosmic-ray detection provide high-resolution observations of the geomagnetic field. We propose [1] to analyze the full darks dataset to calculate the flux of cosmic rays above Earth's surface and estimate variations in the external magnetic field, thereby complementing geophysical observatory measurements. By combining HST results with measurements of solar activity, cosmic ray flux on Earth's surface, and geomagnetic data, our analysis will contribute to understating external magnetic field variations.

A two-phases workflow is needed to process the images: first, extracting the noise from the images; and second, processing the noise to understand if there are connected components that can be understood as cosmic rays impacts. For the first phase, there is a widely used scientific package named IRAF (Image Reduction and Facility [2]). IRAF includes several utilities for manipulating images in the open standard digital file format Flexible Image Transport System (FITS). For the second phase, several algorithms have been proposed to detect every stroke in the dark and determine if it is a cosmic ray or some anomaly (e.g. a damaged pixel). This step is basically about finding connected components on the image and estimating the cosmic ray inner energy. In addition, this step also determines the exact point of impact based on the instruments logs, regarding the telescope position.

In our previous article [3], we proposed an approach for improving the performance of image analysis on distributed memory High Performance Computing systems through Apache Mesos. In this work, we extend our approach to consider two scheduling algorithms for tasks-to-resource assignment in order to further improve the performance.

Originally, the scheduling of jobs and resources was not analyzed in [3], because the focus of our previous research was to design and implement a parallel approach to distribute the work with the main goal of obtaining the best horizontal scaling possible. Apache Mesos [4] is a resource scheduler that uses internal algorithms for assigning tasks to processing machines based on user specified conditions and the available resources in the computational infrastructure. In our previous proposal, tasks became available from the beginning, and so do the processes used to image processing, resulting in a straightforward workflow. In this work, we propose some tweaks to improve the performance, based on a job scheduling logic and not specifically focusing on horizontal scaling. An evolution from the Mesos default scheduling strategy, to a combination of Longest Processing Time (LPT) and Shortest Processing Time (SPT) scheduling algorithms demonstrates that the total makespan can be minimized on favor of a faster processing pipeline.

The article is organized as follows. Next section describes the problem and our previous approach for image processing. Section 3 presents a review of related works on scheduling jobs over virtualized environments. The proposed methods and scheduling algorithms are described in Section 4. After that, Section 5 presents the experimental evaluation of the proposed scheduling algorithms and reports the efficiency results. Finally, Section 6 presents the conclusions and formulates the main lines for future work.

2 Problem Description

The problem of detecting cosmic rays from HST images is complex and requires a lot of computing effort, especially taking into account that a large dataset of historical images (from the early 1990s) is available. Thus, applying a parallel model either on data or instructions domain is required to complete the processing in reasonable execution times. In this work, we follow a data-parallel approach for distributed memory system using containers, which is described in detail in our previous work [3]. All the images are supposed to be available at the same time on an external database. The processing is performed by a set of worker processes running as Docker containers, managed by Apache Mesos.

The execution environment includes a node executing Marathon [5]. Marathon is a framework for container orchestration and it is a part of the Apache Mesos ecosystem. Marathon is responsible of instantiating the containers, as described on a job configuration file. Each container can be seen as an independent processing node, where every node follows the same logic of obtaining a new image from the repository and processing it. In order to perform this task, each container uses Zookeeper [6] for synchronization: each process must know in advance which images are available to determine which image it should process.

Figure 1 presents a diagram of the proposed architecture. The dotted line indicates what is running inside the Mesos managed environment (i.e Marathon, containers and Zookeeper); DockerHub (https://hub.docker.com/) is a publicly accessible cloud registry where docker images are stored (for its later retrieval by Marathon); the images repository is where the FITS files reside (a filesystem reachable from the containers); and the output files are the results from the cosmic rays analysis.

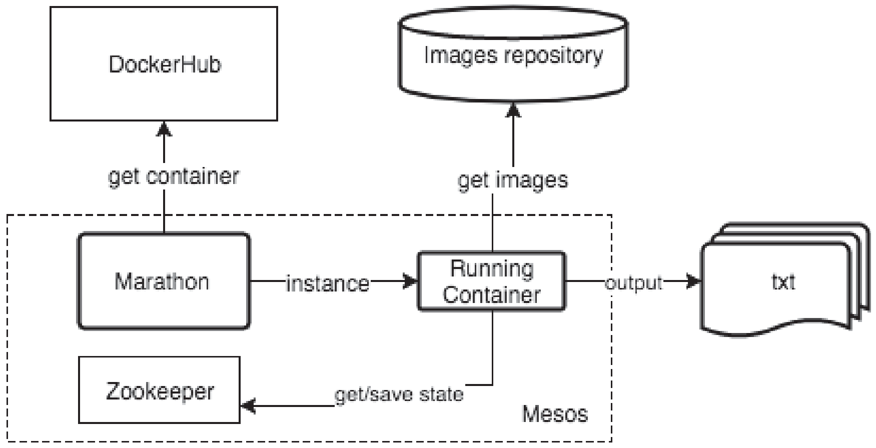

Fig. 1. Architecture for the parallel/distributed processing of HST images to detect cosmic rays using containers.

In order to achieve the goal of obtaining the best horizontal scaling (or, in other words, to perform faster as the number of processing nodes grow in a linear ratio), a solution applying virtualization is proposed. The worker code is packed and distributed through as a Docker container. Docker enables virtualization of an operating system in a lightweight container. As a consequence, Marathon can instantiate as many workers as needed on the execution environment, and the only limitation is given by the physical resources available. Taking into account this last consideration, the system can scale horizontally as much as needed for processing very large datasets (in the case of HST images, more than 10 TB of image data are available, from images taken since the Hubble started operating in 1990).

3 Related work

This section reviews some relevant and recent related works about improving jobs scheduling in distributed computing systems and virtualized environments using containers.

On the last years, as MapReduce became popular on cloud computing, many researchers started focusing on how to apply Hadoop for image processing (see for example the articles by Golpayegani and Halem [7], or Ali and Kumar [8]). Particularly on the present work, the focus will be over the already established architecture and the objective is to improve the tasks assignment strategies.

Job scheduling has been widely analyzed and developed through the last 40 years. The literature about scheduling algorithms is very large, but in this work we focus on studying simple algorithms that can be integrated easily on Mesos and provide improved execution times. Specifically, we consider the classical algorithms proposed by Adam et al. [9], who studied how to schedule parallel processors to minimize makespan assuming partially ordered tasks. This is not the case on this work, as we work with non-related tasks, but it shows a first approach on comparing execution algorithms that is relevant for our work. Almost at the same time, Coffman et al. [10] presented LPT and SPT algorithms applied to job scheduling. Their work is tightly related to the research we propose here, and some of the results presented by Coffman et al. are applied in this article. Based on the already known $O(n \log n)$ algorithm for the execution time for a system with identical processors, from Graham [11]) Coffman et al. demonstrated the efficacy of several heuristics to optimize the mean flow time or the total makespan, separately. Closer in time, Kovács [12] showed how to properly assign jobs to a setup where one processor is faster than the others. His work is also close to our proposal, but the main difference is related to the proportion between fast and slow processors. Finally, Oyetunji [13] defined and used several metrics to analyze performance on scheduling strategies that are useful to explain the algorithms proposed in our article.

Related to Map-Reduce applications for astronomical data processing, Wiley et al. [14] showed how to adapt image co-addition to the MapReduce framework. Image co-addition is a technique that has the objective of getting a high quality image as output, consuming lower resolution images as input. They realized several experiments combining Hadoop and SQL with the result of processing 100.000 images (the equivalent to 300 million pixels) in 3 minutes. Finally, Singh et al. [15] developed a custom version of MapReduce on python to process astronomical datasets. Their work developed parallel processing recipes for multicore machines for astronomical data processing. These recipes are intended to be be used by astronomers with PyIRAF/IRAF knowledge. They compared three different approaches for parallelizing the execution (Pool/Map, Process/Queue and Parallel Python) with the result of Process/Queue being the faster one. This approach involves two FIFO queues (for input of parameters and output of results) that work as pipelines to connect the nodes. This approach is similar to the one we apply in our work, because it does not block the pipeline in anyway, and enables the horizontal scaling based on the number of workers.

Our work combines distributed computing (using data parallelism) and job scheduling algorithms applied to a cosmic ray detection. This approach enables future researchers to focus on data analysis taking advantage of results presented here.

4 Scheduling algorithms for cosmic ray detection using Mesos

This section presents the two algorithms proposed for minimizing total execution time of the image processing tasks.

4.1 Architecture for the parallel execution

The diagram of the proposed architecture for the parallel execution of the IRAF image processing in our original work [3] was already presented in Fig. 1.

Regarding the scheduling considerations, in our previous work Marathon instantiates the containers as described in the configuration file, and it responsibility is just to keep them running. It does not decide how the tasks are assigned to the workers. Then, by default, tasks are assigned on a non-ordered basis. The first processor that becomes available will get the first image on the list. This can be thought as a straightforward list scheduling algorithm, where no precondition apply neither on the process or the images.

4.2 Scheduling Model

Considering the architecture described in Fig. 2, an improved job scheduling strategy can be devised. The workers have equal ready times, and the wait time for getting tasks assigned is zero (since all the images are available from the beginning).

Given the previous considerations, the scheduling problem can be defined according to the following guidelines:

- Which is the best method for tasks-to-processor assignment in order to minimize the total finishing time when processing a given set of images?
- Is it possible to maximize the throughput of the computational infrastructure used for the processing?

The problem also must take into account that: all tasks are independent, and no precedence requirements are defined; there is no need for preemption; every task runs on a separate worker and job priorities are not considered; all the processors used by workers are identical (or uniform, if needed, as it will be shown on Section 4.5).

The mathematical model for the scheduling problem considers the following elements:

- A set of processors $P = \{p_1, ..., p_m\}$.
- A set of tasks (images to process) $T = \{t_1, ..., t_n\}$.
- A speed function $s : P \rightarrow N^+$, where $s(p_j)$ gives the computing capacity of processor p_j.
- A weight function $w : T \rightarrow N^+$, where $w(t_i)$ gives the computing cost needed to process task t_i.

The objective of the scheduling problem is to find a function $f : T \rightarrow P$ that assigns tasks to processors minimizing w_i/s_j where $f(t_i) = p_j$.

According to the classification by Graham et al. [16], the resulting scheduling problem can be classified as of type $P \mid\mid C_{max}$.

4.3 Scheduling Strategies

Scheduling problems within type $P \mid\mid C_{max}$ are proved to be NP-complete [10]. However, the problem complexity can be relaxed imposing some constraints, for example assigning priorities. Thus, the order in which the tasks will be assigned can be determined by either priorities or worker processing speed.

If each image is paired with a priority number, a well-known list scheduling algorithm can be applied to obtain a performance bound below $2 - 1/m$ (being m the number of processors). Particularly, using LPT, the mean performance bound is $\frac{4}{3} - \frac{1}{3} \times m$ [11]. Specifically for this work, the image size works as a priority indicator: the bigger the image size, the bigger the priority.

4.4 LPT-CRD Scheduling Algorithm

The traditional LPT algorithm consists on sorting the tasks (images) in increasing size order and workers (processors) according to their processing power. Under this task ordering logic, tasks with larger processing times are guaranteed to start first, and they are executed by workers running on the fastest processors. We propose an adapted LPT-CRD algorithm, which is the version used for assigning images to workers following the LPT principle.

The pseudocode for the adapted LPT-CRD algorithm applied to cosmic ray detection on astronomical images is shown in Algorithm 1.

Algorithm 1: Adapted LPT-CRD algorithm for $P \mid\mid C_{max}$.

```
begin
    Order tasks $\{t_1, ..., t_n\}$, such that $w(t_1) \geq ... \geq w(t_n)$
    for $i \leftarrow 1$ to $m$ do
        $c_i := 0$
        /* proc. $p_i$ are assumed to be idle from time $c_i$=0 on        */
    end
    j := 1
    repeat
        $c_k := \min \{c_i\}$
        Assign task $t_j$ to processor $p_k$ at time $c_k$
        /* the first non-assigned task from the list is scheduled on
           the first processor that becomes free                      */
        $c_k := c_k + w(t_j)$
        j := j + 1
    until $j = n$; /* all tasks have been scheduled                   */
end
```

The schema for ordering workers (CPUs) and tasks (images) in the LPT-CRD algorithm is graphically presented in the diagram on Fig. 2.

4.5 Combined CRD Scheduling Algorithm

In Marathon, the processor speed of each worker can be adjusted to a custom value on the task definition. This procedure allows applying a different approach for scheduling, suitable for the paradigm of heterogeneous computing [17]. By adjusting the speed to a different value, the processor speed of each worker can be modified, for maintaining a set of *uniform* workers. By definition, processors are *uniform* if every job that executes on a processor of computing capacity s for t time units completes $s \times t$ units of execution. Assuming this scenario, the main focus is to minimize the *mean flowtime* for the tasks.

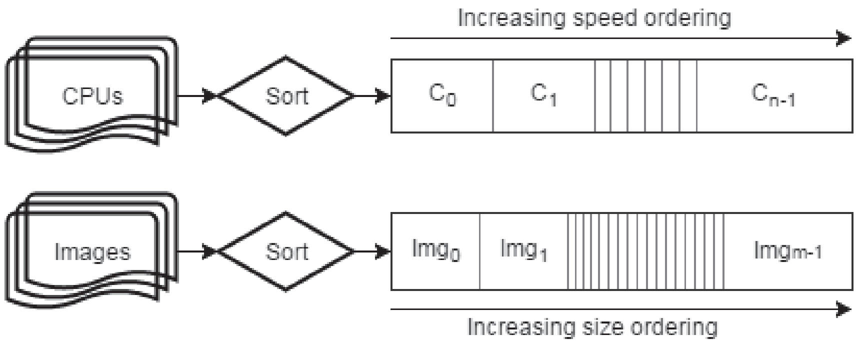

Fig. 2. In LPT-CRD, workers and images are sorted in increasing speed/size order

The *flowtime* is defined as the sum of the finishing times of all tasks (see a formal definition in Section 5.2), and the *mean flowtime* is the average flowtime value when considering the number of tasks in execution in a batch. This new version of the scheduling problem is within the class $Q \mid\mid \sum C_j$. Horowitz and Sahn [18] demonstrated that for the case of minimizing mean flowtime on a system with m processors it is possible to implement an algorithm whose complexity is $O(n \log mn)$.

Algorithm 2 presents the pseudocode of the proposed solution (combined CRD algorithm, CCRD) to solve the problem of maximizing throughput when processing astronomical images for cosmic ray detection. The main idea in CCRD consists in combining the scheduling strategies proposed by the LPT and SPT heuristics.

In CCRD, both workers and images are sorted in increasing order of power/size. At any moment where an idle worker is ready to process, the CCRD algorithm assigns an available image to it. If the worker is running on a fast processor, the algorithm selects the biggest image available. However, if the worker is running on a slow processor, the algorithm selects the smallest image available. Since the images are already sorted by size, this assignment is performed at $O(1)$.

Figure 3 describes the proposed schema for the CCRD scheduling algorithm. A worker is defined as "slow" if its processing power is 60% of available processing power, and "fast" if it is 100%. Mesos determines the available processing power based on the underlying infrastructure and the Marathon tasks requirements. These values are arbitrary, the percentage defining a slow processor can be anyone below 100%, actually. The maximum must be 100% though, because the code to execute is not parallel at instruction level and running it on more than one processor has no positive impact on performance.

Algorithm 2: Combined CRD algorithm for $Q \mid\mid \sum C_j$.

```
begin
    Order tasks $\{t_1, ..., t_n\}$, such that $w(t_1) \geq ... \geq w(t_n)$
    for $i \leftarrow 1$ to $m$ do
        $c_i := 0$
        /* proc. $q_i$ are assumed to be idle from time $c_i$=0            */
    end
    j := 1
    l := n
    repeat
        $c_k := \min \{c_i\}$
        if $q_k$ is a fast processor then
            Assign task $t_j$ to processor $q_k$ at time $c_k$
            j := j + 1
        else
            Assign task $t_{n-j}$ to processor $q_k$ at time $c_k$
            l := l - 1
        end
        /* the first or the last non-assigned task from the list is
           scheduled on the first processor that becomes free,
           depending on its type                                     */
        $c_k := c_k + w(t_j)$
    until $l = j$; /* all tasks have been scheduled                    */
end
```

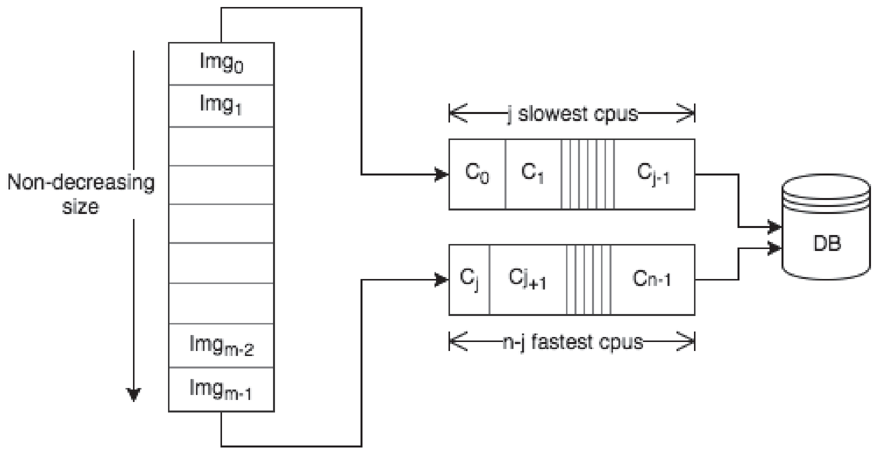

Fig. 3. Proposed schema for minimizing mean flowtime using CCRD algorithm

Finding the optimum ratio between the values for fast and slow processors speed, in terms of makespan and mean flowtime minimization, is an interesting theoretical issue, which we propose as the focus for a future line of work. This further analysis should also consider studying the optimum number of nodes of each type. In this article, for the sake of simplicity, we establish as of five of each type, making a total of 10 workers, which provides a realistic description of a parallel system in the studied architecture.

The details about how to indicate Marathon to instantiate both the slow and fast processors are presented in the codes shown in Figure 4 and Figure 5, respectively. In addition to processing speed, the configuration file must also include the instance count (i.e., in the setup used in this work, five slow processors and five fast processors) and the memory resources to be used. Both type of workers are configured with enough memory to guarantee that every image can be manipulated and the analysis can be performed without any memory issues. Finally, there are several other settings that are required by Marathon (e.g., container URL at Dockerhub, environment variables, etc). We do not report the details here because these settings are not directly related to the scheduling problem.

```
{
  "id": "slow_worker",
  "cpus": 0.6,                    //CPU percentage to be reserved
  "mem": 512.0,                   //RAM to be reserved
  "instances": 5                  //number of nodes to instantiate
}
```

Fig. 4. Sample code of a Marathon task definition for slow processors (as those using 60% of CPU)

```
{
  "id": "fast_worker",
  "cpus": 1.0,                    //CPU percentage to be reserved
  "mem": 512.0,                   //RAM to be reserved
  "instances": 5                  //number of nodes to instantiate
}
```

Fig. 5. Sample code of a Marathon task definition for fast processors (as those using 100% of CPU)

5 Experimental evaluation

This section presents the experimental evaluation of the proposed scheduling algorithms compared to the original efficiency results [3].

5.1 Computational platform

The experimental evaluation was executed using the same machine configuration, even when considering different configurations for the Marathon jobs in the proposed scheduling algorithms. The CPU processing power was capped to the same maximum value (equivalent to 8 processors). The host computer used to perform the tests is a *c4.4xlarge* instance from the EC2 Amazon Web Services (AWS) cloud, with 16 cores running Ubuntu 14.04, Intel Xeon E5-2666 v3 at 2.90 GHz, with a disk of 200 GB and 30 GB of memory.

5.2 Metrics

Performance metrics. The standard *speedup* and *computational efficiency* metrics are used for the comparison between the sequential model and the parallel execution based on containers running in Mesos. In this context, the speedup is defined as the ratio between the total processing time between the sequential execution time (T_1) and the parallel execution time (T_N) when using a specific number of N computing elements, as defined by Equation 1 below. The efficiency is the normalized value of the speedup, according to the number of computing elements used, as defined by Equation 2.

$$S_N = \frac{T_1}{T_N}, \tag{1}$$

$$E_N = \frac{S_N}{N}. \tag{2}$$

Finally, an *acceleration* metric is reported, to evaluate the relative improvement (in execution time) obtained when comparing a two algorithms as defined by

$$acceleration_{AB} = \frac{executiontime(Algorithm_A)}{executiontime(Algorithm_B)} \tag{3}$$

Scheduling evaluation. When considering the time spent to execute all tasks within a job, the most usual metrics to optimize are the *makespan* and the *flowtime* [19]. The makespan is a relevant objective to evaluate the resource utilization; it is defined as the time spent from the moment when the first task begins execution to the moment when the last task is completed. The flowtime evaluates the sum of the tasks finishing times, and it is important from the point-of-view of the users, since it reflects the response time of a computational system for a set of submitted tasks [20].

5.3 Scheduling improvements results

Table 1 reports the total makespan when evaluating the computing times for the three schedulers proposed (the default scheduler in Mesos [3], LPT-CRD for identical processors with ordering, and combined CRD for uniform processors with ordering). The table also reports the values of the acceleration, speedup, and computational efficiency metrics. The analysis is performed using different number of images in a job (100, 500, and 1000 images), following the Bag-of-Tasks (BoT) paradigm for distributed computing [21]. In our case, the BoT corresponds to a set of images. Since images are independent, they can be assigned to different workers for processing, according to the BoT model.

Table 1. Makespan results and computational efficiency analysis for the proposed scheduling algorithms over different datasets, according to the BoT model.

# images	*execution time (s)*		*acceleration*	*speedup*	*efficiency*
	default	*improved*			
		LPT-CRD Algorithm			
100	346	644	0.54	2.83	0.28
500	2382	1705	1.40	1.69	0.17
1000	5180	3438	1.51	4.17	0.42
		Combined CRD Algorithm			
100	346	533	0.65	3.41	0.34
500	2382	1710	1.39	1.69	0.17
1000	5180	3326	1.56	4.31	0.43

The results in Table 1 show that both LPT-CRD and Combined CRD algorithms performed better than the default scheduling strategy. Also, differences become more significant as the image dataset grows. For instance, on the 100 images dataset the speedups are 2.83 and 3.41 (for LPT-CRD and CCRD respectively), but goes up to 4.17 and 4.31 when the dataset size is ten times bigger. Its noticeable a minor speedup on the 500 images dataset, but this value is still better than the execution time provided by the default scheduling algorithm in Mesos. In addition, the acceleration achieved is reported as a measurement of performance improvement. On this regard, the proposed algorithms also performed better than the default scheduling, and increasing as the datasize grows, showing a good scalability behavoir. The proposed scheduling strategies allows executing faster the processing.

Table 2. Flowtime results and computational efficiency analysis for the proposed scheduling algorithms over different datasets, according to the BoT model.

# images	*execution time (s)*		*acceleration*	*speedup*	*efficiency*
	default	*improved*			
		LPT-CRD Algorithm			
100	186.26	327.44	0.57	2.83	0.28
500	996.83	864.52	1.15	1.69	0.17
1000	2127.65	1739.53	1.22	4.17	0.42
		Combined CRD Algorithm			
100	186.26	242.67	0.77	3.41	0.34
500	996.83	850.20	1.17	1.69	0.17
1000	2127.65	1708.33	1.25	4.31	0.43

Table 2 reports the total flowtime when evaluating the computing times for the three schedulers proposed, and also the computational metrics evaluated.

Similarly to the makespan evaluation, results in Table 2 indicate that the proposed algorithms computed significantly improved results when considering the mean flowtime as optimization metric. For both scheduling algorithms, the acceleration increased as the images datasets became bigger. As reported in the the results, the acceleration grew slower than the makespan acceleration. This behavior could be analyzed on a future line of work, answering why makespan acceleration grows faster than flowtime acceleration. Since the reported acceleration grows as the images dataset size increases, it can be concluded that both strategies performed better than the default scheduling in terms of flowtime.

Figure 6 graphically summarizes the main results of the computational efficiency analysis for LPT-CRD and Combined CRD on different image datasets when compared with the default Mesos scheduling algorithm.

Figure 6 indicates that both algorithms performed faster than the default scheduling strategy for all the tested datasets. The graphics show that there are slight differences between LPT-CRD and CCRD, and the speedup increases as the dataset size increases. Regarding acceleration, Figure 6 illustrates how it increased over the image dataset size, but slowing its pace. Further tests should be run to generalize this behavior for bigger image datasets as the ones proposed to analyze in the project “Geophysics using Hubble Space Telescope”.

From the obtained results, we conclude that the scheduling improvements are convenient to process the full set of images (10 TB) available from HST. Either LPT-CRD or CCRD execute much faster than the original version. A processing that would eventually take months could be computed in days or even hours, depending on the behavior of the speedup on the dataset size.

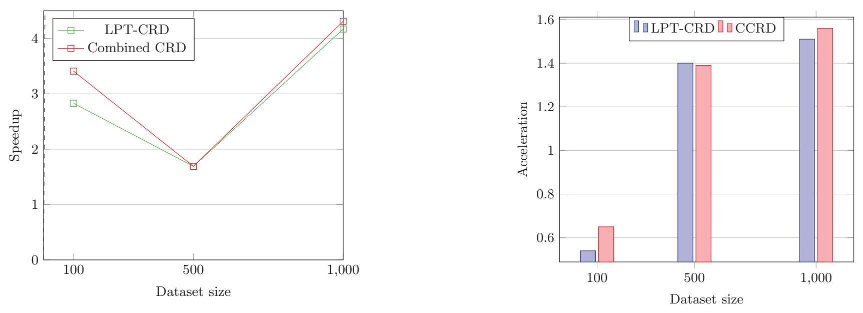

Fig. 6. Computational efficiency analysis for LPT-CRD and Combined CRD on different image datasets

6 Conclusions and future work

In this work, we improve an existent distributing computing architecture using Apache Mesos and Docker for processing images within the project "Geophysics using Hubble Space Telescope". We proposed using adapted LPT and SPT algorithms, combined with different processors, for scheduling. According to the metrics used for performance evaluation (makespan and mean flowtime), the proposed schedulers improved over the traditional one in Mesos. The main results indicate that the total makespan can be reduced up to 35% (acceleration 1.56), and the mean flowtime can be reduced 20 % (acceleration 1.25) when using five fast workers, five slow workers and a dataset of 1000 images.

The main contribution of this article is that including some adjustments on the algorithms for job scheduling, an Apache Mesos based architecture can perform better regarding makespan and flowspan metrics. This performance improvement contributes to reducing times in astronomical images processing tasks.

The main lines of current and future work are related to exploring the utilization of different CPU speeds according to the heterogeneous computing model, for example having different levels of speed and generalizing the results of the present work from two levels (slow and fast) to many levels. Finding the optimum ratio between slow and fast processing speeds, and the best configuration in terms of how many workers of each type obtains minimizes the total makespan is another line for further improving the proposed scheduling algorithms.

References

1. G. Tancredi, G. Cromwell, S. Deustua, G. Gonzalez, S. Nesmachnow, and G. Schnyder, "Geophysics using Hubble Space Telescope," 2016, Hubble Space Telescope Cycle 24 approved proposal.

2. NOAO, “IRAF Project Home Page,” July 2016. [Online]. Available: http://iraf.noao.edu/
3. G. Schnyder and S. Nesmachnow, “Improving the performance of cosmic ray detection using Apache Mesos,” International Supercomputing Conference in México, 2016.
4. The Apache Software Foundation, “Mesos,” July 2016. [Online]. Available: http://mesos.apache.org/
5. Mesosphere Inc., “Marathon: A cluster-wide init and control system for services in cgroups or Docker containers,” July 2016. [Online]. Available: https://mesosphere.github.io/marathon/
6. The Apache Software Foundation, “Apache ZooKeeper,” July 2016. [Online]. Available: http://zookeeper.apache.org/
7. N. Golpayegani and M. Halem, “Cloud computing for satellite data processing on high end compute clusters,” in *Int. Conf. on Cloud Computing*, 2009, pp. 88–92.
8. M. Ali and J. Kumar, “Implementation of image processing system using handover technique with map reduce based on big data in the cloud environment,” *International Arab Journal of Information Technology*, vol. 13, no. 2, 2016.
9. T. L. Adam, K. M. Chandy, and J. R. Dickson, “A comparison of list schedules for parallel processing systems,” *Communications of the ACM*, vol. 17, no. 12, pp. 685–690, 1974.
10. E. G. Coffman and R. Sethi, “Algorithms minimizing mean flow time: schedule-length properties,” *Acta Informatica*, vol. 6, no. 1, pp. 1–14, 1976.
11. R. L. Graham, “Bounds on multiprocessing timing anomalies,” *SIAM Journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.
12. A. Kovács, “Tighter approximation bounds for LPT scheduling in two special cases,” *Journal of Discrete Algorithms*, vol. 7, no. 3, pp. 327–340, 2009.
13. E. O. Oyetunji, “Some Common Performance Measures in Scheduling Problems : Review Article,” *Research Journal of Applied Sciences,Engineering and Technology*, vol. 1, no. 2, pp. 6–9, 2009.
14. K. Wiley, A. Connolly, J. Gardner, S. Krughoff, M. Balazinska, B. Howe, Y. Kwon, and Y. Bu, “Astronomy in the Cloud: Using MapReduce for Image Co-Addition,” *Publications of the Astronomical Society of the Pacific*, vol. 123, no. 901, pp. 366–380, 2011.
15. N. Singh, L. M. Browne, and R. Butler, “Parallel astronomical data processing with Python: Recipes for multicore machines,” *Astronomy and Computing*, vol. 2, pp. 1–10, 2013.
16. R. Graham, E. Lawler, J. Lenstra, and A. Kan, “Optimization and approximation in deterministic sequencing and scheduling: a survey,” *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.
17. M. Eshaghian, *Heterogeneous Computing*. Artech House, 1996.
18. E. Horowitz and S. Sahni, “Exact and approximate algorithms for scheduling non-identical processors,” *Journal of the ACM*, vol. 23, no. 2, pp. 317–327, 1976.
19. S. Nesmachnow, “Parallel multiobjective evolutionary algorithms for batch scheduling in heterogeneous computing and grid systems,” *Computational Optimization and Applications*, vol. 55, no. 2, pp. 515–544, 2013.
20. J. Leung, L. Kelly, and J. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., 2004.
21. W. Cirne, F. Brasileiro, J. Sauvé, N. Andrade, D. Paranhos, and E. Santos-Neto, “Grid computing for bag of tasks applications,” in *Proceedings of the 3^{rd} IFIP Conference on E-Commerce, E-Business and EGovernment*, 2003.