Context-Aware Scheduling for Apache Hadoop over Pervasive Environments

Guilherme Cassales, Andrea Schwertner Charão, Manuele Kirsch Pinheiro, Carine Souveyet, Luiz Angelo Steffenel

To cite this version:

Guilherme Cassales, Andrea Schwertner Charão, Manuele Kirsch Pinheiro, Carine Souveyet, Luiz Angelo Steffenel. Context-Aware Scheduling for Apache Hadoop over Pervasive Environments. 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015), Jun 2015, Londres, United Kingdom. pp.202-209, 10.1016/j.procs.2015.05.058 . hal-01196986

HAL Id: hal-01196986

https://hal-paris1.archives-ouvertes.fr/hal-01196986

Submitted on 15 Sep 2015

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 202 – 209

Procedia Computer Science

The 6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

Context-Aware Scheduling for Apache Hadoop over Pervasive Environments

Guilherme W. Cassales[a], Andrea S. Charão[a], Manuele Kirsch Pinheiro[b], Carine Souveyet[b], Luiz A. Steffenel[c]

[a]*Laboratório de Sistemas de Computação, Universidade Federal de Santa Maria, Santa Maria, Brazil*
[b]*Centre de Recherche en Informatique, Université Paris 1 Panthéon Sorbonne, Paris, France*
[c]*Laboratoire CReSTIC - Équipe SysCom, Université de Reims Champagne-Ardenne, Reims, France*

Abstract

This article proposes to improve Apache Hadoop scheduling through the usage of context-awareness. Apache Hadoop is the most popular implementation of the MapReduce paradigm for distributed computing, but its design doesn't adapt automatically to computing nodes' context and capabilities. By introducing context-awareness into Hadoop, we intent to dynamically adapt its scheduling to the execution environment. This is a necessary feature in the context of pervasive grids, which are heterogeneous, dynamic and shared environments. The solution has been incorporated into Hadoop and evaluated through controlled experiments. The experiments demonstrate that context-awareness provides comparative performance gains, especially when part of the resources disappear during execution.

Keywords: Context-Awareness; Map Reduce; Apache Hadoop; Task Scheduling; Pervasive Grid; Big Data

1. Introduction

Apache Hadoop is a framework for distributed and parallel computing, implementing the MapReduce programming paradigm, which aims at processing big data sets[1]. Hadoop is designed to scale up from a single server to thousands of machines, each offering local computation and storage.

Without specific configuration by the administrator, Apache Hadoop supposes the use of dedicated homogeneous clusters for executing MapReduce applications. As the overall performance depends on the task scheduling, Hadoop performance may be seriously impacted when running on heterogeneous and dynamic environments, for which it was not designed for.

* Guilherme Cassales. Tel.: +55-55-3220-8849 ; fax: +55-55-3220-8523.
E-mail address: cassales@inf.ufsm.br

This is an especial concern when deploying Hadoop over pervasive grids. Pervasive grids are an interesting alternative to costly dedicated clusters, as the acquisition and maintenance of a dedicated cluster remain high and dissuasive for many organizations. According to Parashar and Pierson[2], pervasive grids represent the extreme generalization of the grid concept, in which the resources are pervasive. Pervasive grids propose using resources embedded in pervasive environments in order to perform computing tasks in a distributed way. Concretely, they can be seen as computing grids formed by existing resources (desktop machines, spare servers, etc.) that occasionally contribute to the computing grid power. These resources are inherently heterogeneous and potentially mobiles, coming in and out the grid dynamically. Knowing that, in essence, pervasive grids are heterogeneous, dynamic, shared and distributed environments, its efficient management becomes a very complex task[3]. Task scheduling is thus severely affected by the management of the environment complexity.

Many works have proposed to improve the adaptability of the Hadoop framework on environments that diverge from the initial supposition, each having their own proposal and objectives[4,5,6,7]. The PER-MARE project[8], in which this work was developed, aims at adapting Hadoop to pervasive environments[9].

Indeed, Hadoop is based on static configuration files and the current versions do not adapt well to resources variations over the time. In addition, the installation procedures force the administrator to manually define the characteristics of each potential resource, such as the memory and the number of cores of each machine, which is a hard task in a heterogeneous environment. All these factors prevent deploying Hadoop on more volatile environments. The PER-MARE project aims at the improvement of Hadoop so that it could adapt itself to the execution context and therefore be deployed over pervasive grids.

In order to adapt Hadoop to a pervasive grid environment, supporting context-awareness is essential. Context-aware is the capacity of an application or software to detect and respond to environment changes[10]. A context-aware system is able to adapt its operations to current state without human intervention, therefore improving the system's usability and efficiency[11]. In pervasive grids, the scheduling is a task that may be benefited in context-aware systems, collecting data about the grid resources and making decisions based on the data collected.

This work focuses on our developments to introduce context-awareness capabilities on Hadoop task scheduling mechanisms. Through a context collection procedure and minimal changes on Hadoop's resource manager, we are able to update the information about the availability of resources in each node of the grid and then influence the scheduler tasks assignments.

The rest of the paper is organized as follows: Section 2 presents Apache Hadoop architecture and scheduling mechanisms. Section 3 discusses related work, focusing on context-awareness and on other works that try to improve Hadoop schedulers. Section 4 presents our proposal of context-aware scheduling, while Section 5 presents the experiments conducted and the achieved results. We finally conclude this paper in Section 6.

2. About Hadoop Scheduling

The Apache Hadoop framework is organized in a master and slave architecture, with two main services: storage (HDFS) and processing (YARN). Both services have their own master and slave components, as presented on Fig. 1. It is possible to see the *NameNode* and *ResourceManager* services, which are the masters of the HDFS and YARN respectively, and their slave counterparts, the *DataNode* and *NodeManager*. It is also possible to note the *ApplicationMaster*, the component responsible for internal application (job) management, or simply task scheduling. While *ResourceManager* is the component responsible for job scheduling. Each node also runs a set of *Containers*, where the execution of Map and Reduce tasks takes place.

2.1. Hadoop Schedulers

Concerning job scheduling, Hadoop offers several options. The simplest scheduler, called Hadoop Internal Scheduler, processes all jobs in arrival order (FIFO). This scheduler has a good performance in dedicated clusters where the competition for resources is not a problem. Another scheduler available is the Fair Scheduler, mainly used to compute batches of small jobs. It uses a two level scheduling to fairly distribute the resources[12].

The third scheduler available is the Capacity Scheduler. The CapacityScheduler is designed to run Hadoop MapReduce as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the uti-

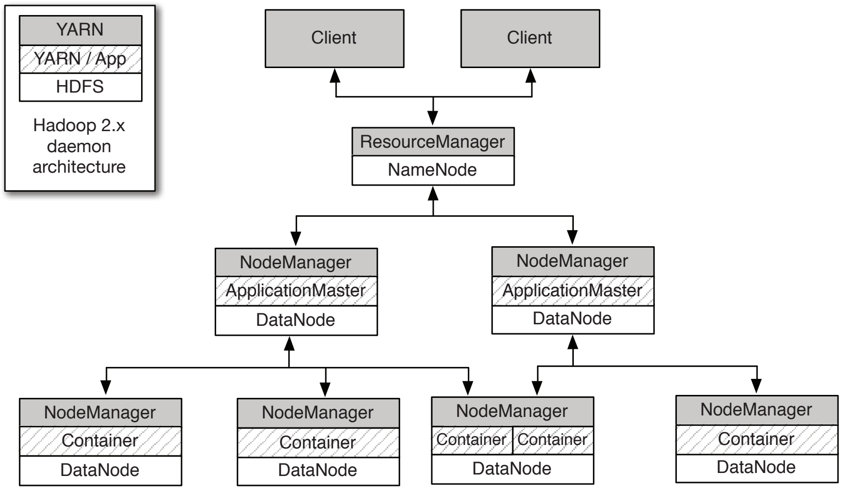

Fig. 1. General Apache Hadoop architecture

lization of the cluster while running Map-Reduce applications. The CapacityScheduler is designed to allow sharing a large cluster while giving each organization a minimum capacity guarantee. The central idea is that the available resources in the Hadoop MapReduce cluster are partitioned among multiple organizations that collectively fund the cluster based on computing needs. There is an added benefit that an organization can access any excess capacity not being used by the others users. This provides elasticity for the organizations in a cost-effective manner[12].

The existence of these schedulers allows a flexible management of the framework. Despite that, the available schedulers neither detect nor react to the dynamicity and heterogeneity of the computing environment, a typical concern on pervasive grids.

3. Related Work

Over the years, different works proposed improvements to the scheduler mechanisms from Hadoop in order to respond to specific needs. These contributions may be divided as proposals of new scheduling methods or proposals of improvement for the resource distribution.

Works like[4], [13] and[6] assume that most applications are periodic and demand similar resources regarding CPU, memory, network and hard disk load. These assumptions allow the applications and nodes to be analyzed regarding the CPU and I/O potential, enabling the optimization of execution through matching of nodes and applications with the same characteristics. Another work that focuses on a new scheduling method is[14], where the authors propose the usage of a capacity-demand graph that assists the calculation of optimal scheduling based on an overall cost function.

While previously works focus on performance improvement using static information about resources and applications, other works sought to incorporate task specific information on their proposals. For example, works like[5] and[15] attempted to better distribute the tasks of an application as a way to reduce its response time large clusters. The authors of[5] use heuristics to infer the estimated task progress and to make a decision about the launching of speculative tasks. Speculative tasks are copies of tasks launched when there is a possibility that the original task is on a faulty or too slow node. Another work[15] proposes the usage of historical execution data to improve decision making.

The final result of both methods – new scheduling mechanics and improvement of resource distribution – is a load rebalancing, forcing faster nodes to process more data and slower nodes to process less data. The work[7] tries to achieve that through a system based on resource supply and demand, allowing each user to directly influence scheduling through spending rates. The main objective is to allow a dynamic resource sharing based on preferences set by each user.

There are also works like[16], which attempt to provide a performance boost in jobs through better data placement, mainly using data location as information to decision making. The performance gain is achieved through data rebalancing on nodes, raising the load on faster nodes. This proposal reduces the number of speculative tasks and data transfers through the network.

The work[17] uses a P2P structure to arrange the cluster. On this approach, nodes can change their function (master/slave) over time and can have both functions at the same time, since the functions are tied to applications and not the cluster. The objective of this work was the adaptation of MapReduce paradigm to a P2P environment, which given the natural volatility of P2P environments, would offer support to pervasive grids. However, this proposal focuses on providing a resilient infrastructure and does not explore the scheduling of jobs and tasks.

Indeed, most of previously cited works does not actually consider current state of the available resources. Resources are described, not observed. However, context-aware computing[11] has demonstrated that this observation is possible and that the execution environment may influence application behavior. A question then raises: can we improve MapReduce scheduling by observing current execution environment? Next sections will try to answer this question.

4. Context-Aware Scheduling

The main goal of this work is to improve the scheduling of Hadoop by adding support to dynamic changes in the availability of resources, like those occurring in a pervasive environment. Similar to works on Section 3 we try to improve the resource distribution, since faster and more robust nodes would have more data to process. Different from these works, we opted to modify the Hadoop code through insertion of dynamic context information using, as far as possible, an existing scheduler (*Capacity Scheduler*). In order to detect dynamic changes, the scheduler must collect context information that, in this case, refers to available resources on the nodes. Slaves must communicate periodically with the master in order to keep information updated and let the scheduler adapt to the new context. On the following section we present a more detailed explanation of the changes implemented in Apache Hadoop.

4.1. Context collector

By default, Hadoop reads information about the nodes from XML configuration files. These files contain many Hadoop configuration parameters, including the resource capacity of each node. Once loaded, the information will not be updated until the next time the service is started. As pervasive environments may face performance changes during the execution of an application, we need a mechanism that updates contextual information during runtime according to the environmental conditions.

To solve this problem, we integrate a collector module into Hadoop, allowing the collection of contextual information about the available resources. The collector was developed for the PER-MARE project[8], and its class diagram is presented in Fig. 2. The collector module is based on standard Java monitoring API[18], which allows to easily access the real characteristics of a node, with no additional libraries required. It allows collecting different context information, such as the number of processors (cores) and the system memory, using a set of interface and abstract/concrete classes that generalize the collecting process. Due to its design, it is easy to integrate new collectors and improve the resources available for the scheduling process, providing data about the CPU load or disk usage, for example.

4.2. Communication

Gathering the context information required to feed the Hadoop scheduler requires transmitting this information through the network from slave nodes (NodeManager) till master node (ResourceManager), which is responsible for the scheduling. Instead of relying on a separate service, we chose to use the ZooKeeper API[19], a tool initially developed inside Hadoop that becomes a full project as its usage was extended to other applications. Zookeeper provides efficient, reliable, and fault-tolerant tools for the coordination of distributed systems. In our case, we use ZooKeeper services to distributed context information.

As illustrated in Fig. 3, all slaves (NodeManager) run an instance of the NodeStatusUpdater thread, that collect data about the real resource availability of the node every 30 seconds. If the amount of available resources changes, the DHT on ZooKeeper will be updated. Similarly, the master (ResourceManager) also creates a thread to watch ZooKeeper. If the Zookeeper node detects a DHT change, the master will be notified and update the scheduler information based on the new information. This solution extends a previous one we proposed in[20] by offering a real

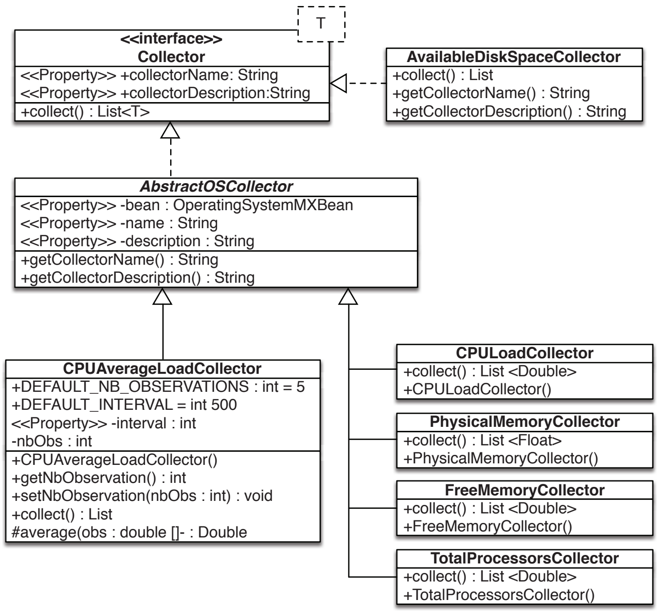
T
<<interface>>
Collector
<<Property>> +collectorName: String
<<Property>> +collectorDescription:String
+collect() : List<T>
AvailableDiskSpaceCollector
+collect() : List
+getCollectorName() : String
+getCollectorDescription() : String
AbstractOSCollector
<<Property>> -bean : OperatingSystemMXBean
<<Property>> -name : String
<<Property>> -description : String
+getCollectorName() : String
+getCollectorDescription() : String
CPUAverageLoadCollector
+DEFAULT_NB_OBSERVATIONS : int = 5
+DEFAULT_INTERVAL = int 500
<<Property>> -interval : int
-nbObs : int
+CPUAverageLoadCollector()
+getNbObservation() : int
+setNbObservation(nbObs : int) : void
+collect() : List
#average(obs : double []- : Double
CPULoadCollector
+collect() : List <Double>
+CPULoadCollector()
PhysicalMemoryCollector
+collect() : List <Float>
+PhysicalMemoryCollector()
FreeMemoryCollector
+collect() : List <Double>
+FreeMemoryCollector()
TotalProcessorsCollector
+collect() : List <Double>
+TotalProcessorsCollector()

Fig. 2. Context collector structure

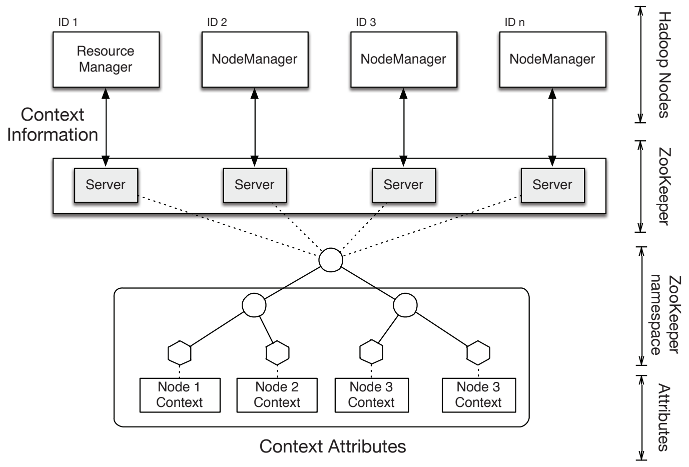

Fig. 3. Using ZooKeeper to distribute context information

time observation of available nodes. Indeed, our previous solution[20] only updated information regarding the resources on service initialization, replacing the XML configuration file, while this one updates resource information whenever the availability changes. As a result, scheduling is performed based on the current resource state.

5. Experiments and Results

To facilitate understanding, the description of the experiments was divided into three subsections, a subsection on the setting and environment preparation, other intended for results and another containing the analysis of the results.

5.1. Preparations and configuration

In order to test the new behavior of the framework, we conducted experiments with the Grid'5000[21] in cluster *genepi*. We configured a cluster with 4 slaves, each having the following configuration: 2 Intel(R) Xeon(R) CPU E5420 @ 2.50 GHz (totalizing 8 cores per node) and 8 GB of RAM. All nodes run Ubuntu-x64-12.04, with JDK 1.7 installed, and the Hadoop distribution was the 2.5.1 YARN version.

As benchmark we used the application TeraSort in a data set of 15 GB. The resources considered in the experiments were the memory and number of cores, which have a direct impact on the amount of tasks Map allocated. We had due care to not run any other application that could influence the results. The information about containers' execution is obtained from the analysis of Hadoop logs. After the modification in scheduling was implemented, the following scenarios have been configured for the experiments:

Scenario A: in this scenario we simulate a dedicated Hadoop cluster, so that the reported memory will always correspond to the available memory. We consider reported memory as the information that the scheduler will use in the scheduling process, while available memory is the free memory of the node or cluster. Using a direct notation, the reported memory is 100 % and the available memory is also 100% all through the execution.

Scenario B: in this case, nodes can be used for other purposes than running Hadoop, so the reported memory may, at some point, differ from the available memory initially configured for Hadoop usage. This case corresponds to the default behavior of Hadoop, where memory resources are provided through a XML configuration property *yarn.nodemanager.resource.memory-mb*. Using a direct notation, the reported memory is 100%, but the available memory is 50%. To simulate this scenario we actually reduce the number of resources (by reducing the number of nodes) while reporting the same amount of available memory from Scenario A.

Scenario C: here, nodes are also shared with other applications and, indeed, a new application starts running after Hadoop has been launched, and the context awareness collector performs an information update every 30 seconds. Hadoop is started after the first update has taken place, so the execution context corresponds to the real available resources. Using a direct notation, the reported memory and the available memory correspond to 50% of the values reported on Scenario A.

Scenario D: finally, this scenario presents an extension of Scenario C where Hadoop is started before the occurrence of the update, so the scheduler starts with wrong available resource information and must adapt during the execution with the help of the context collector. Using a direct notation, the reported memory at the beginning of the execution is 100% of the resources from Scenario A (wrong information), but at runtime this information is updated to 50%.

5.2. Results

The results of the experiments are resumed on Table 1 and Fig. 4. On Table 1, the first column represents the scenarios explained above. The second column represents the total time used by all map tasks. The third column represents the average execution time of map tasks. The fourth column represents the standard deviation on average time for each case. The last column represents the number of speculative tasks launched.

As stated before, every task is processed on Containers, and some containers are not affected by scheduling, as the ApplicationMaster or the Reduce tasks. For this reason we ignore these tasks and concentrate the analysis on the elements that can be affected by the context-aware scheduling.

Table 1. Resume of results expressed in seconds.

Case	Total Map Time (s)	Average Map Time (s)	Map tasks Standard Deviation	Number of speculative tasks
A	149	39.47	15.73%	2
B	788	222.97	59.86%	1
C	348	38.38	18.09%	3
D	477	68.42	29.91%	1

In addition, we generated Gantt diagrams for all scenarios, illustrated in Fig. 4, in which each case presents a line for each node in the cluster. As stated in the description of the scenarios, Scenarios B, C and D run on half of the nodes from Scenario A to simulate a reduced amount of resources.

Each line portraits the resources consolidated by that node represented on a color scale, where the darker the tone, the more containers are executing simultaneously. White means no container executing, and black means 16 containers. Additionally, the lines are segmented along the chart to indicate that a container has either finished or started at that moment. The chart is scaled in seconds, and all charts go from 0 to 780 seconds.

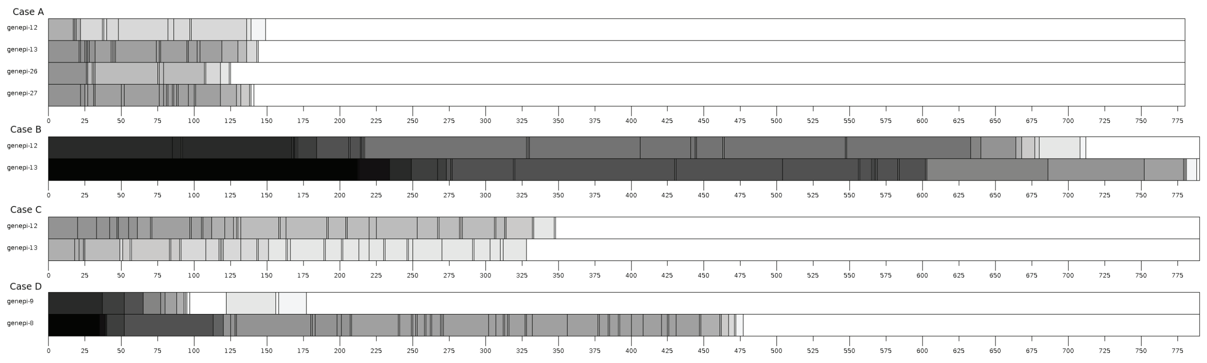

Fig. 4. Gantt charts of the experiments

By analyzing Table 1, it is possible to note that cases A and C, where the real resources were known before the start of the job, had the smallest average map time and standard deviation. This is due to the fact that the nodes were never overloaded since the scheduler had the right information. This can also be seen on charts, where cases A and C have similar tones. Indeed, case C had half the resources of case A and took twice the time to complete, which is the expected behavior. Table 1 also indicates that the amount of speculative tasks launched were among 1 to 3 on all cases, which might be contrary to expected on cases B and D. However speculative tasks are launched based on a progress scale from 0 to 1, this progress is then compared to all other tasks, which would also be very slow.

On the other hand, both cases B and D have a dark tone at the beginning, meaning that 16 containers are executing (twice the real capacity). Also, the first containers took about 20 seconds to execute in cases A and C (cf. the first segment line), while on case B it requires about 70 seconds, evidencing an overload on the nodes. Although both (B and D) had the same initial conditions (50% available resources and 100% reported resources), case D took less time to complete. The reason for this is that the context collector updates the reported resources on D, allowing the scheduler to reorganize tasks after the first container set completes. Indeed, case D had high concentration of executing containers only in the first moments, unlike case B where nodes keep overloaded until the end due to the absence of updated information. Although the scheduler does not preempt excess containers, it is possible to note an improvement on performance of around 40% based solely on the fact that the scheduler avoids overloading the nodes.

Scenarios C and D show that regular context updates contribute to reduce the execution time on a dynamic Hadoop cluster. We showed that even when starting with the same circumstances of the worst case (Scenario B), updating the information helps the scheduler to minimize the execution time. Our solution contributes both to provide correct information before the execution starts (Scenario C) or to adapt the execution to resources changes (Scenario D).

6. Conclusion

On this work, we aimed at developing the ability of detecting and adapting to resource availability changes on the environment of Apache Hadoop Capacity Scheduler. The improvements were implemented with a lightweight context collector and communication provided by Apache ZooKeeper. These improvements go further our previous work[20] by considering a continuous observation of node capabilities. The results show that the solution can positively impact the performance, especially in the situations where the available resources drop after the beginning of the execution.

While Hadoop does not perform preemption/migration of tasks, the association of a context-aware scheduler and speculative tasks may contribute to circumvent the bottlenecks caused by the resources variability. Our future works will concentrate on modifying the scheduler algorithms, in order to consider a wider collection of context information than the current parameters (memory and cores). We believe that additional parameters such as CPU speed, network speed and even battery capacity are essential parameters on a pervasive grid. Finally, we intend to evaluate our proposal on a real pervasive grid environment, composed of heterogeneous off-the-shelf volunteer computers, in order to measure the impact of context observation on high dynamic environments.

Acknowledgment

The authors would like to thank their partners in the PER-MARE project[8] and acknowledge the financial support given to this research by the CAPES/MAEE/ANII STIC-AmSud collaboration program (project number 13STIC07).

References

1. Dean, J., Ghemawat, S.. Mapreduce: Simplified data processing on large clusters. *Commun ACM* 2008;**51**(1):107–113.
2. Parashar, M., Pierson, J.M.. Pervasive grids: Challenges and opportunities. In: Li, K., Hsu, C., Yang, L., Dongarra, J., Zima, H., editors. *Handbook of Research on Scalable Computing Technologies*. IGI Global. ISBN 978-160566661-7; 2010, p. 14–30. doi:`10.4018/978-1-60566-661-7.ch002`.
3. Nascimento, A.P., Boeres, C., Rebello, V.E.F.. Dynamic self-scheduling for parallel applications with task dependencies. In: *Proceedings of the 6th International Workshop on MGC*; MGC '08. New York, NY, USA. ISBN 978-1-60558-365-5; 2008, p. 1:1–1:6.
4. Kumar, K.A., Konishetty, V.K., Voruganti, K., Rao, G.V.P.. Cash: context aware scheduler for hadoop. In: *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*; ICACCI '12. New York, NY, USA. ISBN 978-1-4503-1196-0; 2012, p. 52–61.
5. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.. Improving mapreduce performance in heterogeneous environments. In: *Proceedings of the 8th USENIX conference on Operating systems design and implementation*; OSDI'08. Berkeley, CA, USA: USENIX Association; 2008, p. 29–42.
6. Rasooli, A., Down, D.G.. Coshh: A classification and optimization based scheduler for heterogeneous hadoop systems. In: *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*; SCC '12. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-4956-9; 2012, p. 1284–1291.
7. Sandholm, T., Lai, K.. Dynamic proportional share scheduling in hadoop. In: *Proceedings of the 15th International Conference on Job Scheduling Strategies for Parallel Processing*; JSSPP'10. Berlin, Heidelberg. ISBN 3-642-16504-4, 978-3-642-16504-7; 2010, p. 110–131.
8. STIC-AmSud, . PER-MARE project. 2014. `http://cosy.univ-reims.fr/PER-MARE`, Last access: July 2014.
9. Steffenel, L.A., Flauzac, O., Charão, A.S., Barcelos, P.P., Stein, B., Nesmachnow, S., et al. Per-mare: Adaptive deployment of mapreduce over pervasive grids. In: *Proceedings of the 2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*; 3PGCIC '13. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-5094-7; 2013, p. 17–24.
10. Maamar, Z., Benslimane, D., Narendra, N.C.. What can context do for web services? *Commun ACM* 2006;**49**(12):98–103.
11. Baldauf, M., Dustdar, S., Rosenberg, F.. A survey on context-aware systems. *Int J Ad Hoc Ubiquitous Comput* 2007;**2**(4):263–277.
12. Apache, . Apache hadoop. 2014. `http://hadoop.apache.org/docs/r2.6.0/index.html`. Last access: November 2014.
13. Tian, C., Zhou, H., He, Y., Zha, L.. A dynamic mapreduce scheduler for heterogeneous workloads. In: *Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*; GCC '09. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-3766-5; 2009, p. 218–224.
14. Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., Goldberg, A.. Quincy: fair scheduling for distributed computing clusters. In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*; SOSP '09. New York, NY, USA: ACM. ISBN 978-1-60558-752-3; 2009, p. 261–276.
15. Chen, Q., Zhang, D., Guo, M., Deng, Q., Guo, S.. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In: *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*; CIT '10. Washington, DC, USA: IEEE Computer Society. ISBN 978-0-7695-4108-2; 2010, p. 2736–2743.
16. Xie, J., Ruan, X., Ding, Z., Tian, Y., Majors, J., Manzanares, A., et al. Improving mapreduce performance through data placement in heterogeneous hadoop clusters. In: *Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW)*. 2010, .
17. Marozzo, F., Talia, D., Trunfio, P.. P2p-mapreduce: Parallel data processing in dynamic cloud environments. *J Comput Syst Sci* 2012;**78**(5):1382–1402.
18. Oracle, . Overview of java se monitoring and management. 2014. `http://docs.oracle.com/javase/7/docs/technotes/guides/management/overview.html`, Last access: July 2014.
19. Hunt, P., Konar, M., Junqueira, F.P., Reed, B.. Zookeeper: Wait-free coordination for internet-scale systems. In: *Proceedings of the USENIX Annual Technical Conference*. Boston, MA: USENIX Association; 2010, p. 11–11. URL: `http://dl.acm.org/citation.cfm?id=1855840.1855851`.
20. Cassales, G., Charao, A., Kirsch-Pinheiro, M., Souveyet, C., Steffenel, L.. Bringing context to apache hadoop. In: *8th International Conference on Mobile Ubiquitous Computing*. Rome, Italy; 2014, .
21. Grid 5000, . Grid 5000. 2013. `https://www.grid5000.fr/`, Last access: July 2014.