Bringing Context to Apache Hadoop

Guilherme W. Cassales*, Andrea S. Charão*, Manuele Kirsch-Pinheiro†, Carine Souveyet† and Luiz Angelo Steffenel‡
*Laboratório de Sistemas de Computação, Universidade Federal de Santa Maria, Santa Maria, RS, Brazil
Email: {cassales,andrea}@inf.ufsm.br
†Centre de Recherche en Informatique, Université de Paris 1 - Panthéon Sorbonne, Paris, France
Email: {manuele.kirsch-pinheiro, carine.souveyet}@univ-paris1.fr
‡CReSTIC - SysCom, Université de Reims Champagne-Ardenne, Reims, France
Email: luiz-angelo.steffenel@univ-reims.fr

***Abstract*—One of the first challenges when deploying MapReduce over pervasive grids is that Apache Hadoop, the most known MapReduce distribution, requires a highly structured environment such as a dedicated cluster or a cloud infrastructure. In pervasive environments, context-awareness becomes essential to coordinate the resources (task scheduling, data placement, etc.) and to adapt them to the environment variable behavior. In this paper we present our first efforts to improve Hadoop by introducing context-awareness on its scheduling algorithms. The experiments demonstrate that context-awareness allows Hadoop to better scale based on actual resource availability, therefore improving the task allocation pattern and rationalizing resource usage in a heterogeneous dynamic network.**

Keywords–Context-awareness; MapReduce; Apache Hadoop; job scheduling.

I. Introduction

Given today's high volume of available data, new methods of processing this huge volume are being researched. Recently one of these new methods, MapReduce [1] and its most known implementation Apache Hadoop[1], is gaining space among both users and developers. MapReduce [1] is a programming model for parallel data processing, while Hadoop is a software platform implementing MapReduce. Thanks to Hadoop, it is possible to easily process large data sets in a computer cluster.

Designed to work with homogeneous cluster environments, Apache Hadoop is currently used not only on dedicated clusters, but also over cloud computing infrastructures. Despite its design, Hadoop has some liabilities that negatively affect its performance. One of those is the assumption that every node has the same resource capacity, and that this capacity is set in a default XML file. As the cluster size scales, this task becomes quickly time consuming and error-prone. This limitation has an important impact on Hadoop performance, since ill-configured nodes will harm the overall performance.

Besides, this assumption of an homogeneous environment limits the deployment of Hadoop over desktop and pervasive grids. Pervasive grids[2][3] are characterized by their heterogeneity, integrating nodes with quite different capabilities. Such heterogeneous environments represent an interesting alternative to cloud computing infrastructures. Indeed, as underlined by [4], cloud computing solutions present important drawbacks when considering data transfer (transferring gigabytes of data accross the network can be costly) and data security/privacy (putting sensitive data on the cloud may represent an important issue for some application).

In order to extract the best performance from Apache Hadoop on heterogeneous environments, it is necessary to reconsider how tasks are scheduled on the cluster. Indeed, MapReduce performance in Hadoop is tightly tied to the scheduler [5] and to its capability of observing the environment characteristics. Currently, Hadoop scheduler considers only information from the XML configuration file, ignoring the actual state of the nodes. In order to overcome this drawback, we propose in this paper to improve Hadoop scheduling through a context-aware approach. Context can be defined as any information that can be used to characterize the situation of an entity (a person, place or object) that is considered relevant to the interaction between a user and an application [6]. We advocate that being aware of context in which a job is executed may contribute to a better use of resources in heterogeneous environments. In this paper, we propose to open Hadoop scheduler to the job execution context, observing real node conditions instead of a static (potentially mismatching) configuration file. By collecting the job execution context, we allow a better utilization of cluster's resources and also a better adaptation to heterogeneous environments.

The rest of the paper is organized as follows: Section II introduces MapReduce model and the basis of Apache Hadoop framework. Section III discusses related works, focusing on context-awareness and on other improved Hadoop schedulers. Section IV analyzes and evaluates Hadoop scheduling mechanism. Section V presents our proposal of context-aware scheduling, while Section VI presents experiments and first results. We conclude in Section VII.

II. About Hadoop

The Apache Hadoop is a framework that has the purpose of facilitating distributed processing through the MapReduce model. MapReduce [1] divides computation into two phases: map and reduce. During map, input data is split into smaller slices of data, whose analysis is distributed over the participating nodes. Each participant computes one (or more) slice of data, generating intermediary key/values results. During reduce phase, intermediary values concerning a given key are put together and analyzed, generating final key/values results. Hadoop framework is in charge of distributing data and map/reduce tasks over the available nodes. As a result, programmers need only to focus on map and reduce functions, since data and task distribution becomes transparent.

[1] http://hadoop.apache.org

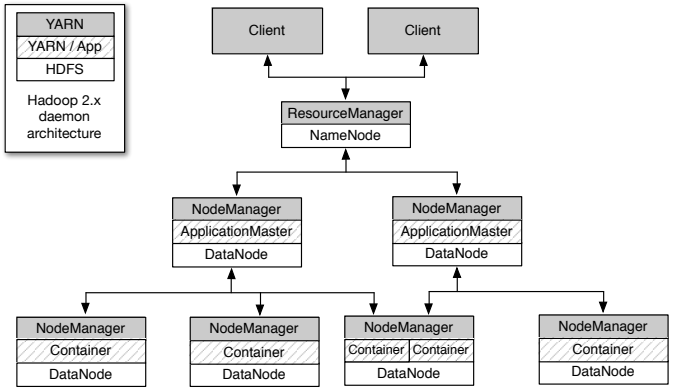

Figure 1. General Hadoop 2.x (YARN) Architecture

Apache Hadoop has two main components (see Figure 1), which are Hadoop Distributed FileSystem (HDFS) and Yet Another Resource Negotiator (YARN). These components are respectively responsible for the data management on a distributed file system and for MapReduce tasks and job processing. YARN manages tasks and jobs distribution over the available nodes and it is in charge of scheduling jobs according to nodes capacity. Each node information is controlled by an individual NodeManager, while overall cluster information is centralized by the ResourceManager.

YARN integrates a scheduler that is responsible for distributing tasks over the available nodes. Current YARN structure (Figure 1) aims at acquiring and using each NodeManager resource information to improve the performance of its default scheduler, the CapacityScheduler. Basically, each NodeManager consults configuration files in order to discover node declared capacity and inform ResourceManager about its existence. This information is transferred to the scheduler that uses it for deciding an appropriate job scheduling.

In order to deploy Hadoop on a cluster, every node must have some XML configuration files available in their local Hadoop installation. In fact, given Hadoop huge dependence on XML files, even the nodes resources are set by these files. This peculiarity makes a more adaptive environment something hard to achieve with the default Hadoop distribution.

III. Related Works

Because Hadoop performance is tightly dependent on the computing environment, but also on the application characteristics, several researchers focused on bringing context-awareness to Hadoop. Their works can be roughly classified as: (i) job or task schedulers, whose purpose is changing the Hadoop scheduling, and (ii) resource placement facilitators.

In the first case, we find works like [5][7] or [8]. Those assume that most jobs are periodic and demand similar CPU, network and disk usage characteristics. As a consequence, these works propose classification mechanisms that first analyze both jobs and nodes with respect to its CPU or I/O potential, allowing an optimized matching of applications and resources when a job is submitted. For instance, works [5] and [7] classified both jobs and nodes in a scale of I/O and CPU potential, while [8] go beyond I/O and CPU potential and propose a full classification of jobs in order to match these jobs with nodes belonging to the same classification. Similarly, [9] proposes a capacity-demand graph that helps in the calculation of the optimum scheduling from a global cost function.

While the previous works focus on the improvement of the overall cluster performance through an offline knowledge about the applications and the resources, other works focus on individual tasks in order to ensure a smooth operation. For instance, works like [10] and [11] focus on improving tasks deployment inside a job, as a way to reduce the response time in large clusters, executing many jobs of short duration. These works rely on heuristics to infer the job estimated progress and decide whether to launch a speculative task on another possibly faster machine. Similarly, [11] propose using historical execution traces to improve its predictions. They propose a re-balancing of data across the nodes, leaving more data to faster nodes and less data on slower nodes.

Finally, works like [12] aim at providing better performance on jobs through better data placement, using mainly the data locality as decision making information. The performance gain is achieved by the data re-balancing in nodes, feeding faster nodes with more data. This lowers the cost of speculative tasks and also of data transfers through the network.

We may observe that most of these works rely on the categorization of jobs and nodes, which is hard in a dynamic environment like pervasive grids. Even when runtime parameters such as elapsed time or data placement are considered, they assume a controlled and well known environment. Because of these assumptions, these works fail on responding to the requirements of pervasive grids. Indeed, previous works focus on the reduction of response time or improvement of overall performance, which is a goal slightly different from ours, which is to adapt Hadoop to heterogeneous environments.

IV. Hadoop scheduling

In order to improve Hadoop, it is important to understand current mechanisms that would be influenced and whose behavior should be altered. Thus, before presenting how we extend current Hadoop behavior with context information, we shall introduce the Hadoop resource allocation pattern.

A. Understanding Hadoop Allocation Pattern

Apache Hadoop operates in a master/slave hierarchy on both components (YARN and HDFS), each component being subdivided in numerous sub-components (see Figure 1). On the top of YARN daemons, as shown in Figure 1, we found the ResourceManager (RM), which is in charge of managing the entire cluster and of assigning applications to the underlying compute resources. These resources are controlled by the NodeManagers (NM). When a new job is submitted, the ResourceManager delegates the job supervision to an ApplicationMaster (AM). The ApplicationMaster is the manager of the application, every application has one AM. It is the ApplicationMaster that asks for resources for the CapacityScheduler, which tracks the free/used resources and grant them to the AM. These granted resources are actually Containers, a form to represent reservations, the component in which all the processing takes place.

Resource allocation is based on a set of parameters defined in the XML configuration files. These parameters concern both

Table I. RESULTS FOR RM MEMORY ALLOCATION EXPERIMENT

	Default	Higher	Smaller	In Range
Map Memory Request (MB)	1024	1024	1024	3456
Reduce Memory Request (MB)	1024	1024	1024	3712
Minimum Memory (MB)	1024	512	2048	512
Maximum Memory (MB)	8192	768	8192	8192
Map Memory Allocation (MB)	1024	ERROR	2048	3584
Reduce Memory Allocation (MB)	1024	ERROR	2048	4096

memory and number of cores for applications and containers. If no value is given for a precise node, default values from the XML files are assumed, which can substantially differ from real node characteristics.

B. Experimenting Resource Allocation

We have performed a set of experiments in order to evaluate the impact of configuration parameters on resource allocation pattern. We focused on memory allocation, as there are few parameters related to minimum and maximum memory allocations for a single request.

We considered four different scenarios: (*i*) default allocation, (*ii*) request higher than maximum allowed, (*iii*) request smaller than minimum allowed and (*iv*) request inside the range. The results from these scenarios can be seen in Table I. First two rows correspond to the job request, while the following two rows show minimum and maximum memory indicated in the configuration files, and the last rows present what was effectively allocated to the job.

Table I demonstrates that a request with a value higher than the maximum will cause an error that aborts the job. For a request of a value smaller than the minimum, the cluster grants the minimum allowed. The fourth scenario shows a request inside the valid range. Although the requests were similar, the resources granted were different. Indeed, when the request is inside the minimum and maximum range, Hadoop performs a small set of calculations to determine how much memory will be granted. Whenever the minimum allocation is not enough to satisfy the request, the granted value is incremented by the minimum allocation until it matches one of the following cases: *(a)* the value is equal to the request; *(b)* the value is higher than the request and lower than the maximum allocation; or *(c)* the value exceeds maximum allocation.

Results in Table I demonstrate that the default scheduling is closely related to the resource availability. Having a wrong information could ruin the performance of the algorithm. Since there is no mechanism to automatically detect and modify resource parameters, dealing with a heterogeneous environment, such as a pervasive grid, quickly becomes a challenging task. It appears then clear that, in order to support heterogeneous environments, Hadoop must be aware of its real (and not supposed) execution environment.

V. CONTEXT-AWARE SCHEDULING

In order to detect the node real capacity, we chose to integrate a context collector into Hadoop. This collector is charged of observing the execution environment, allowing an automatic detection of each node capacity. Thanks to this context collector, context information representing real memory and CPU conditions of each node can be observed, allowing the proposal of improved scheduling mechanisms.

A. Collecting Context Information

Context information corresponds to a large concept, often related to the observation of a user, a device or the execution environment. Commonly, it is defined as any information that may characterize the situation of an entity. This entity can be a user, a device or the environment itself[6]. Quite often, context information is used for adaptation purposes[13]. Context awareness can then be seen as the capability a system has of observing and reacting to the environment in order to adapt its own behavior to context changes [13][14]. In our case, we are interested on the execution context of a job, which is composed by the nodes executing it. We believe that Hadoop must be aware of this execution context in order to schedule appropriately submitted jobs.

In order to be useful, context information should be acquired and modeled appropriately. Nonetheless, such monitoring should not impact the overall performance of the running applications. This is particularly true for Hadoop, whose goal is precisely to improve performance of MapReduce applications [15]. A lightweight mechanism, in the opposite to traditional context management systems (e.g.[14][13]), is then needed.

Thus, to include context information on Hadoop, we integrated a lightweight collector module, using the Java monitoring API[2], which allows to easily access the real characteristics of a node, with no additional libraries required. The collector module, illustrated by Figure 2, allows observing different context information, such as the number of processors (cores) and the system memory, using a set of interface and abstract classes that generalize the collecting process. Due to its design, it is easy to integrate new collectors and improve available context information for the scheduling process, providing data about the CPU load or disk usage, for example.

Context information is described by using a predefined name and a description. Such name corresponds to a concept identified in a context ontology. This model, inspired from [16], considers each context information as a context element, for which multiple values can be observed. Context ontology allows then to semantically describe each element, while the description gives an human readable definition for it.

This collector module was integrated to the NodeManager, since it is in charge of processing tasks and managing node definition. In this first prototype, we collect node capacity (available memory and number of cores), and this information is then sent to the ResourceManager. As a consequence, the information from the context collector module allowed us to improve the Hadoop scheduler operation without having to modify its implementation. This is especially interesting as further works will be able to compare other schedulers from the literature without having to modify their implementation.

B. Integrating Context Information

Context information detected using the context collector is transferred to Hadoop scheduler, which can scale the allocation

[2]http://docs.oracle.com/javase/7/docs/technotes/guides/management/

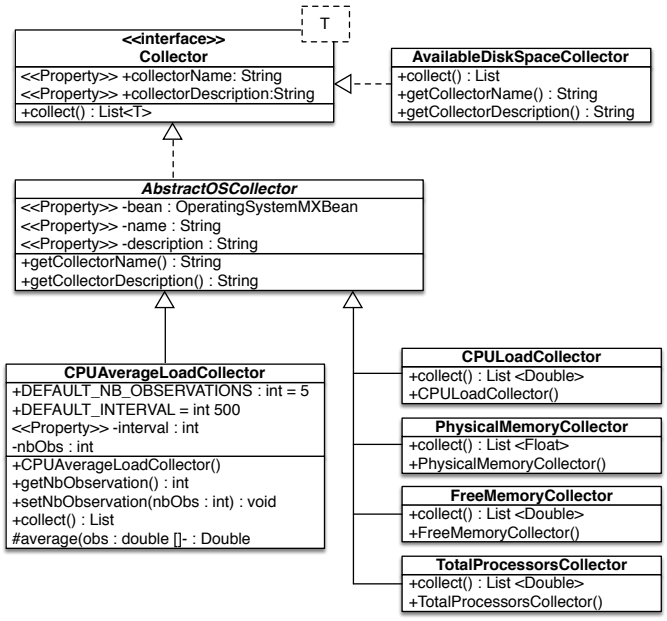

Figure 2. Elements of the context collector for Hadoop.

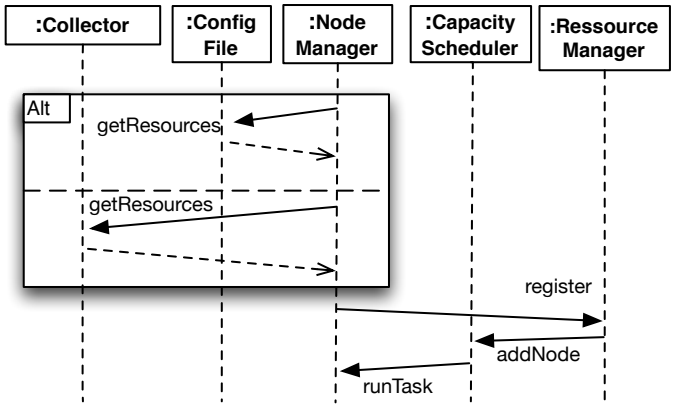

Figure 3. Simplified sequence diagram for resource registering and granting

Table II. RESOURCES AVAILABLE ON ORIGINAL AND CONTEXT-AWARE CAPACITYSCHEDULER

	Original CapacityScheduler	Context-aware CapacityScheduler
Node Memory	8 GB	48 GB
Node Vcores	8	24

limits to the real cluster resource availability. This scaling affects the containers allocation as a function of the available memory and computing cores, impacting therefore on the choice of tasks placement and how speculative task are started. As a result, we could obtain a better usage of the resources, minimizing the need for speculative tasks too. By adapting the capacity to the cluster real resource, no resource would be wasted or left inactive while the scheduler is making tasks wait due to wrong information being received.

In order to integrate the collector, we identified the NodeManager (NM) as the best entry point, since this is the service responsible for processing tasks. Collected context information about available memory and number of cores is sent to the ResourceManager (RM). When each NM registers to the RM, it tries to obtain this information from the context collector, which supplies NM with observed values. This information is sent and provided to the CapacityScheduler that uses it to dispatch tasks. Figure 3 illustrates this process. It is worth noting that, if the collector is unavailable, NM will keep using traditional configuration files.

Information from the context collector module allows us to improve the behavior of the Hadoop scheduler without having to modify its implementation. This is especially interesting as further works will be able to compare other schedulers from the literature without having to modify their implementation.

VI. EXPERIMENTS AND RESULTS

This section provide information about the experiments that were capable of changing the Hadoop scheduling behavior and the results achieved. The experiment consisted in deploying Hadoop services in a cluster with original CapacityScheduler and with the context-aware CapacityScheduler in order to compare the change in total resources availability. Through these experiments, using the TeraSort benchmark, we could observe in a first moment the impact of the default configuration values and the values from our context-aware collector on the tasks allocation. Then, we compare the allocation pattern adopted by the original CapacityScheduler and our context-aware CapacityScheduler. This allows us to compare how the higher resource availability and higher allocation limits impacted the scheduling.

A. Hardware and Software configuration

The experiment was performed in a cluster deployed on Grid'5000[3] computing environment. We used five up to 32 nodes, with a single master, having the following configuration: 2 CPUs AMD@1.7GHz, 12 cores/CPU and 48GB RAM. All nodes run Ubuntu-x64-1204, with JDK 1.7 installed. The Hadoop distribution was the 2.2.0 YARN version.

B. Results and interpretation

With an experiment running on nodes with the same configuration, it would be easy to discover the true capacity of a node, change the values on a XML file and replicate it to all other nodes inside the environment. The problem becomes evident once the environment is not homogeneous, as it would require the discovery of the true capacity of each node and the creation of separated XML files for each node. Indeed, quite often a cluster configuration does not follow real nodes characteristics, being limited to default values. Thus, our first experiment consists in comparing node memory obtained from default implementation and from our context collector.Thanks to the context collector, we could be able to detect real node characteristics, which is significantly different from the default values, as we can see in Table II.

The second experiment we performed compared the behavior of the original CapacityScheduler with our own context-aware CapacityScheduler. This experiment used the same configuration from the previous experiment. We launched a TeraSort job with 5GB data to sort, therefore requesting enough

[3]https://www.grid5000.fr

containers and providing enough data to stress the cluster. The original CapacityScheduler uses the default configuration, with a minimum allocation of 1 GB and 1 core, and maximum of 8 GB and 32 cores per node (summing up 32 GB and 32 cores for the cluster). For the context-aware CapacityScheduler, the collector detects all 192 GB and 96 cores on the cluster, with a minimum allocation of 4 GB and 2 cores and maximum of 24 GB and 12 cores per node.

Gantt charts on Figs. 4 and 5 present tasks (actually the containers) and the resources they are tied to. The different segments indicate the tasks that have been allocated to a given NodeManager, and the numbers inside the segment indicates which containers are running at that moment. Because the default configuration uses one single Reduce task, we only represent the Map tasks.

Figure 4 portraits the Gantt Chart of the TeraSort with original CapacityScheduler. One can notice that some containers had to wait for the completion of others in order to start processing their tasks. Indeed, Hadoop splits the work in 38 Map tasks (numbered 2-39), which are distributed to the nodes according to the known resource capabilities. When the first tasks are completed, new tasks are provided to the nodes, if any available (as illustrated in Figure Fig. 4, where tasks 32-39 represent the second execution wave).

Figure 5 portraits the Gantt Chart of the TeraSort with context-aware CapacityScheduler. In this case the overall completion time was reduced, due to the fact that all containers could be started right after the arrival of the request, thanks to the higher resource availability.

After an analysis and comparison of both charts, it is possible to notice that the default chart has containers 41-43 started on node stremi-5 and container 44 started on node stremi-42, while the context-aware chart has only the standard containers, which are numbered 2-39. This happens because these extra containers are, in reality, speculative tasks launched because other tasks were taking to long to finish.

C. Heterogeneity Simulation

A third experiment was performed to simulate a heterogeneous environment and test how well the context-aware would adapt. Once again, the experiment consisted in executing the TeraSort algorithm in the cluster with the simulated heterogeneous environment using context-aware CapacityScheduler.

This experiment used the same configuration from the previous experiment. The only difference is that the nodes are purposely given false capacities when being added to the RM, simulating the following heterogeneous cluster:

- stremi-17: 28 GB of memory and 14 cores.
- stremi-22: 32 GB of memory and 18 cores.
- stremi-33: 48 GB of memory and 24 cores.
- stremi-35: 24 GB of memory and 12 cores.

Figure 6 portraits the Gantt Chart of the TeraSort execution within the simulated heterogeneous environment, also using context-aware CapacityScheduler. Compared to the default case, the heterogeneous execution shows an improvement, but due to lower cluster capacity, it is slightly worse than the context-aware scheduler on homogeneous environment.

On this experiment a speculative task was launched, the container 41. It is also noteworthy that the scheduler did not change nodes to launch the speculative task, because the node had spare capacity when the request for the speculative arrived.

This experiment shows that it is possible to use this context-aware scheduler in a heterogeneous environment. Indeed, the allocations were adapted to a slightly smaller cluster if compared to the real environment. As a future work, it is possible to set the allocation limits in function not only of total cluster resources but also of each individual node resource capacity.

VII. Conclusion

In this paper, we have proposed to improve Apache Hadoop behavior with context information, thanks to a new context-aware scheduler. These changes allowed Hadoop to be aware of its execution context, and particularly of the real capacity of the nodes composing the cluster. The context-aware CapacityScheduler we have proposed here is capable of receiving the real capacity from each NodeManager, thanks to a lightweight context collector plugged on NodeManager. This provides the cluster a better scaling potential while also using every node's full capacity. Experimental results demonstrate that the context-aware CapacityScheduler could better scale up improving containers management, and consequently the overall Hadoop scheduling behavior.

This context-aware scheduling represents a first step of a further vision, proposed by the PER-MARE project[17]. Indeed, we intend to go further in this direction, considering not only nodes capabilities, but also current state (current available memory, CPU load or network bandwidth, for instance). We strongly believe that such a context-aware behavior is essential for supporting MapReduce application over pervasive grids.

Acknowledgment

The authors would like to thank their partners in the PER-MARE project[4] and acknowledge the financial support given to this research by the CAPES/MAEE/ANII STIC-AmSud collaboration program (project number 13STIC07). Experiments presented in this paper were carried out on Grid'5000 experimental testbed.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, 2008, pp. 107–113.

[2] M. Parashar and J.-M. Pierson, "Pervasive grids: Challenges and opportunities," in Handbook of Research on Scalable Computing Technologies, K.-C. Li, C.-H. Hsu, L. T. Yang, J. Dongarra, and H. Zima, Eds. IGI Global, 2010, pp. 14–30.

[3] V. Hingne, A. Joshi, T. Finin, H. Kargupta, and E. Houstis, "Towards a pervasive grid," in Proceedings of the International Parallel and Distributed Processing Symposium, ser. IPDPS'03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 207.2–. [Online]. Available: http://ebiquity.umbc.edu/_file_directory_/papers/623.pdf

[4] E. E. Schadt, M. D. Linderman, J. Sorenson, L. Lee, and G. P. Nolan, "Computational solutions to large-scale data management and analysis," Nature Reviews Genetics, vol. 11, no. 9, Sept 2010, pp. 647–657.

[5] K. A. Kumar, V. K. Konishetty, K. Voruganti, and G. V. P. Rao, "Cash: context aware scheduler for hadoop," in International Conference on Advances in Computing, Communications and Informatics, ser. ICACCI '12. New York, NY, USA: ACM, 2012, pp. 52–61.

[4] http://cosy.univ-reims.fr/PER-MARE

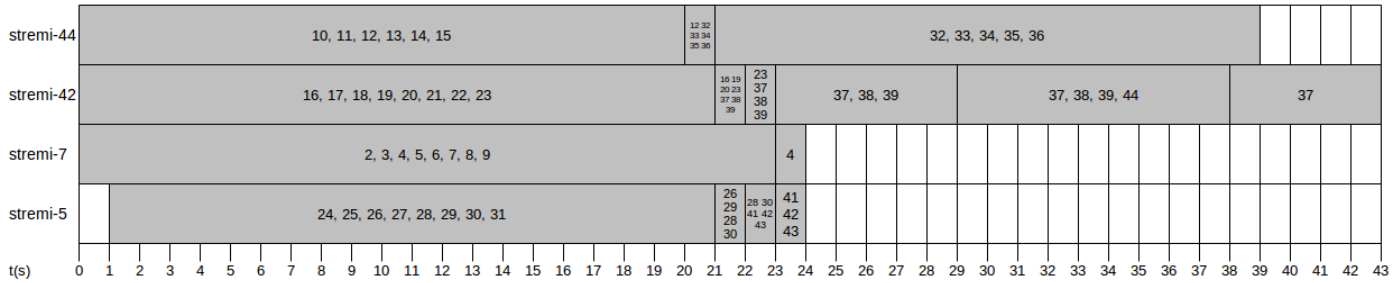

Figure 4. Container assignment with the original CapacityScheduler

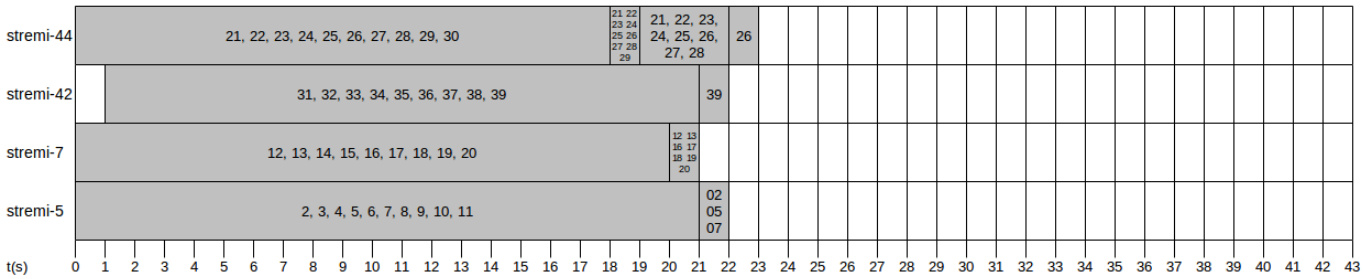

Figure 5. Container assignment with the context-aware CapacityScheduler

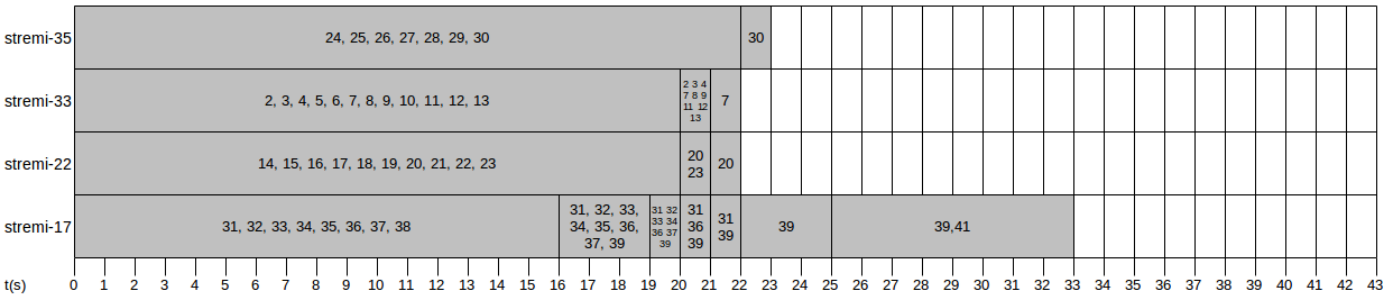

Figure 6. Container assignment in the simulated heterogeneous environment

[6] A. Dey, "Understanding and using context," Personal and Ubiquitous Computing, vol. 5, no. 1, 2001, pp. 4–7.

[7] C. Tian, H. Zhou, Y. He, and L. Zha, "A dynamic mapreduce scheduler for heterogeneous workloads," in 8th International Conference on Grid and Cooperative Computing, ser. GCC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 218–224.

[8] A. Rasooli and D. G. Down, "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, ser. SCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1284–1291.

[9] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in ACM SIGOPS 22nd symposium on Operating systems principles, ser. SOSP '09. ACM, 2009, pp. 261–276.

[10] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in Proceedings of the 8th USENIX conference on Operating systems design and implementation, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 29–42.

[11] Q. Chen, D. Zhang, M. Guo, Q. Deng, and S. Guo, "Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment," in Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, ser. CIT '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 2736–2743.

[12] J. Xie, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, S. Yin, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010.

[13] D. Preuveneers, K. Victor, Y. Vanrompay, P. Rigole, M. Kirsch-Pinheiro, and Y. Berbers, Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications. IGI Global, 2009, ch. Context-Aware Adaptation in an Ecology of Applications, pp. 1–25.

[14] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," International Journal of Ad Hoc and Ubiquitous Computing, vol. 2, no. 4, 2007, pp. 263–277.

[15] M. K. Pinheiro, "Requirements for context-aware mapreduce on pervasive grids," PER-MARE Deliverable D3.1, Deliverable D3.1, 2013. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00858310

[16] M. Kirsch-Pinheiro, J. Gensel, and H. Martin, "Representing context for an adaptative awareness mechanism," in Groupware: Design, Implementation, and Use, ser. LNCS, G.-J. Vreede, L. Guerrero, and G. Marín Raventós, Eds., vol. 3198. Springer, 2004, pp. 339–348.

[17] L. Steffenel, O. Flauzac, A. Schwertner Charao, P. Pitthan Barcelos, B. Stein, S. Nesmachnow, M. Kirsch Pinheiro, and D. Diaz, "Permare: Adaptive deployment of mapreduce over pervasive grids," in P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on, Oct 2013, pp. 17–24.