Bringing Context to Apache Hadoop

Guilherme W. Cassales and Andrea S. Charão
Laboratório de Sistemas de Computação
Universidade Federal de Santa Maria
Santa Maria, RS, Brazil
{cassales,andrea}@inf.ufsm.br

Manuele Kirsch-Pinheiro and Carine Souveyet
Centre de Recherche en Informatique
Université de Paris 1 - Panthéon Sorbonne
Paris, France
{manuele.kirsch-pinheiro, carine.souveyet}@univ-paris1.fr

Luiz Angelo Steffenel
Laboratoire CReSTIC - Équipe SysCom
Université de Reims Champagne-Ardenne
Reims, France
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 [2], 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 drawbacks 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 very time consuming and error-prone and ill-configured nodes will harm the overall performance.

Besides, this assumption of a homogeneous environment limits the deployment of Hadoop over desktop and pervasive grids. Pervasive grids [3][4] 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 Schadt et al. [5], cloud computing solutions present important drawbacks when considering data transfer (transferring gigabytes of data across 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 [6] 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. For instance, when running Hadoop in homogeneous environments, the configuration files represent indeed an easy way to configure a cluster, since one needs only to discover the capacity of a node and to replicate the files to every other node in the environment. However, when using a heterogeneous environment, one will have to discover and edit XML files for each node in a cluster. This behavior often limits the Hadoop scalability in heterogeneous clusters, since the configuration files will not follow real nodes characteristics, and consequently, nodes will be limited to default values.

In order to overcome this drawback, we propose 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 [7]. Context information has been used for adapting application behavior during execution time [8][9], adapting content [10] or components deployment [11][12], for instance. 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.

Our proposal of context-awareness focuses therefore on discovering the real node capacity and providing a scheduler based on true observed information. In this paper, we conducted experiments with a basic set of context information (CPU, memory), which in our tests proved to be significantly different from default values. The experiments demonstrate that the applications performance can be widely improved through the use of context information, which encourages us to develop further the context-aware scheduling with additional parameters such as data locality, CPU speed and even task re-splitting to better explore idle resources and speculative execution.

 ISBN: 978-1-61208-353-7

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.

Apache Hadoop has two main components (see Fig. 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 (Fig. 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. The CapacityScheduler has the role of centralizing the information about NodeManager's capacities on the "master" node, keeping track of them in a global pool of resources and distributing them according to the ApplicationMasters requests.

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 Work

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

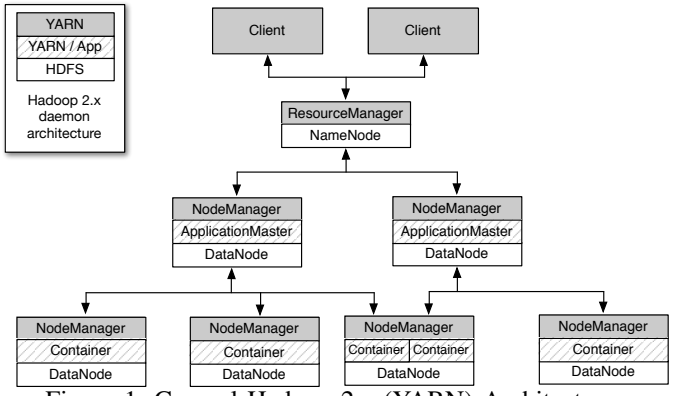

Figure 1: General Hadoop 2.x (YARN) Architecture.

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 Kumar et al. [6], Tian et al. [13] or Rasooli [14]. 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, Kumar et al. [6] and Tian et al. [13] classified both jobs and nodes in a scale of I/O and CPU potential, while Rasooli et al. [14] 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, Isard et al. [15] 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 Zaharia et al. [16] and Chen et al. [17] 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, Chen et al. [17] 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 Xie et al. [18] 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

 ISBN: 978-1-61208-353-7

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. On the top of YARN daemons, as shown in Fig. 1, we found the ResourceManager (RM), which is in charge of managing the resources from the entire cluster and of assigning applications to the underlying computing resources. These resources belong to the NodeManagers (NM), and each NM will inform the RM the amount of available resources upon start.

When a new job is submitted to the cluster, the ResourceManager registers the start of the job and then delegates the supervision to an ApplicationMaster (AM). The ApplicationMaster is the manager of the application, which asks for resources for the CapacityScheduler (a component of ResourceManager). The CapacityScheduler tracks the free/used resources on the cluster and grants them to the AM based on the global (cluster) and local (application) limits. The granted resources are presented as Containers, a processing instance 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 memory and number of cores for applications and containers, and are composed by the minimum and maximum limit. If no value is given for a precise node or application, default values from the XML files are assumed, which can substantially differ from real node characteristics.

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.

B. Experimenting Resource Allocation

To better understand the impact of configuration parameters on the resource allocation mechanism, we present in this section an experiment where we compare different memory requests against the minimum and maximum memory parameters. 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.

Table I: RESULTS FOR RM MEMORY ALLOCATION EXPERIMENT.

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

The results from these scenarios can be seen in Table I. The columns represent each scenario, while the first two lines (Minimum/Maximum Memory) refer to configuration parameters. The third and fourth lines (Map/Reduce Memory Request) refer to the application request parameters. Finally, the last two rows (Allocated Map/Reduce Memory) present the resources effectively allocated to the job based on the other parameters.

From these scenarios, we observe 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 in the minimum-maximum range, Hadoop performs a small set of calculations to determine how much memory will be granted. Whenever the minimum allocation does not 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.

This experiment demonstrates 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 [7]. Quite often, context information is used for adaptation purposes [9]. Context

 ISBN: 978-1-61208-353-7

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 [8][9]. Different context information can be observed for adaptation purposes, varying from user's profile, location and activities till characteristics of the used device and execution environment [9][10][19]. Indeed, several works in the literature [11][12][20][21] propose observing device execution conditions (including available memory, CPU, network connection, battery consumption, etc.) in order to adapt application execution to them, by selecting or deploying appropriate components. 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. Among such information, we can include CPU and memory capacities, node's current charge, but also network speed and data locality (related to HDFS replication) to improve task allocation.

Nonetheless, in order to be useful, context information should be acquired and modeled appropriately, with a minimal impact to the overall performance of the running applications. This is particularly true for Hadoop, whose goal is precisely to improve performance of MapReduce applications [22]. A lightweight mechanism, in the opposite to traditional context management systems [8][9], is then needed.

Thus, to include context information on Hadoop, we integrated a lightweight collector module, using the Java Monitoring API (Application Programming Interface) [23], which allows to easily access the real characteristics of a node, with no additional libraries required. The collector module, illustrated by Fig. 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 Kirsch et al. [19], 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 a 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 limits to the real cluster resource availability. This scaling affects the containers allocation as a function of the available

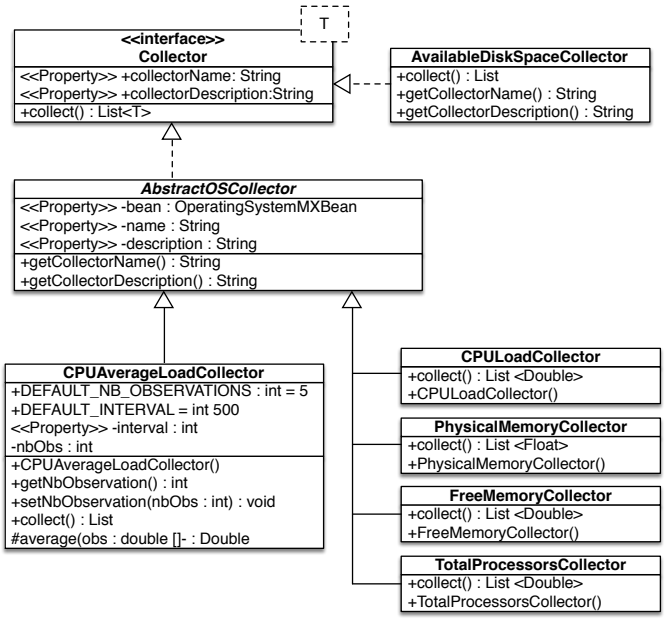

Figure 2: Elements of the context collector for Hadoop.

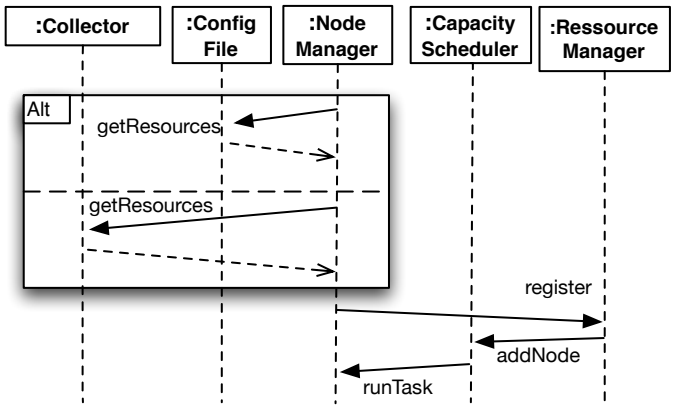

Figure 3: Simplified sequence diagram for resource registering and granting.

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. Fig. 3 illustrates this process. It is worth noting that, if the collector is unavailable, NM will keep using traditional configuration files.

 ISBN: 978-1-61208-353-7

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 provides information about the experiments we conducted to improve the Hadoop scheduling behavior, as well as the results achieved. These experiments consisted in deploying Hadoop services in a cluster with original CapacityScheduler and comparing it against a context-aware CapacityScheduler. Theses experiments focus on two basic parameters, available memory and number of cores, whose injection in the CapacityScheduler is straightforward and require no modifications to the CapacityScheduler algorithm. Additional context information can be included through the collector module, as described in the Section V-A.

The experiments were performed in a cluster from the Grid'5000 [24] computing environment. We used five nodes (a master and 4 slaves), each having the following configuration: 2 AMD CPUs 1.7GHz, 12 cores/CPU and 48 GB of RAM. All nodes run Ubuntu-x64-12.04, with JDK 1.7 installed, and the Hadoop distribution was the 2.2.0 YARN version. As The TeraSort benchmark was used as application subject.

A. 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.

Our first experiment compares the allocated node memory when using the default implementation or our context collector. Thanks to the context collector, we could be able to detect real node characteristics, which are significantly different from the default values, as one can see in Table II. This discrepancy in capacity collected/used is due the utilization of default XML parameters by the default scheduler. As stated before, the Hadoop configuration is heavily dependent on XML files, making it hard to extract the full potential of the cluster without a delicate and time-consuming configuration. The default XML files have the node memory set to 8 GB and the node number of cores set to 8, that are reasonable numbers when using a cluster of personal computers, but when deployed in a larger cluster these values will, more often than not, waste potential.

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

The second experiment compares 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 5 GB data to sort, therefore requesting enough 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, maximum allocation of 8 GB and 32 cores per node for a total resource 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 (48 GB and 24 cores from each node), with a minimum allocation of 4 GB and 2 cores and maximum allocation of 24 GB and 12 cores per node.

Figs. 4 and 5 present a chart with tasks execution (actually the containers) and the nodes 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. When a segment ends, it means that at least one task has finished or a new task has been started on that NodeManager. If a number was on a segment and disappears on the next, that task has finished. If a number wasn't on the first segment and suddenly appears on the next, that task has started. Because the default configuration uses one single Reduce task, we simplify the diagram by representing only the Map tasks.

Fig. 4 portraits the execution of the TeraSort algorithm with the 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 Fig. 4, where tasks 32-39 represent the second execution wave).

Fig. 5 portraits the execution of the TeraSort algorithm 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. These extra tasks are what Hadoop calls speculative tasks, when one or more tasks are advancing slower than the rest, Hadoop creates new copy tasks in order to prevent bigger losses in throughput. This means that if the original tasks were indeed experiencing issues, in the event they fail to complete, another task is already processing the faulty task data, otherwise, if the task eventually finishes its processing before the speculative, the copies will be disregarded.

B. 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.

 ISBN: 978-1-61208-353-7

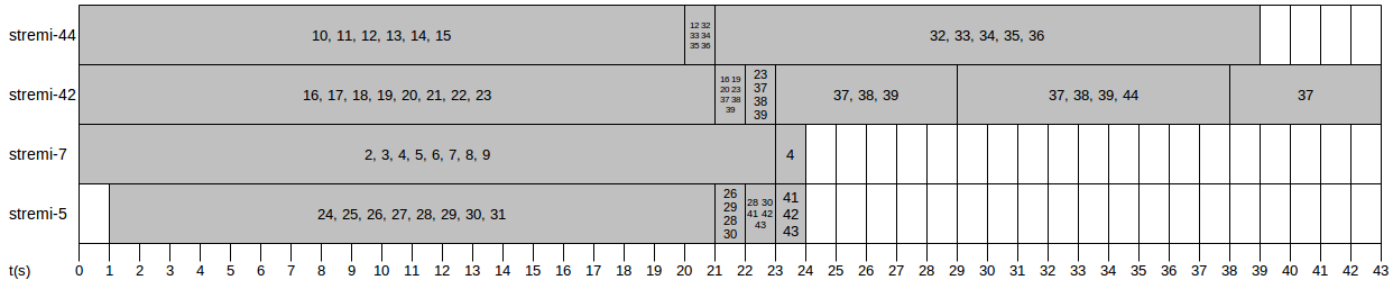

Figure 4: Container assignment with the original CapacityScheduler.

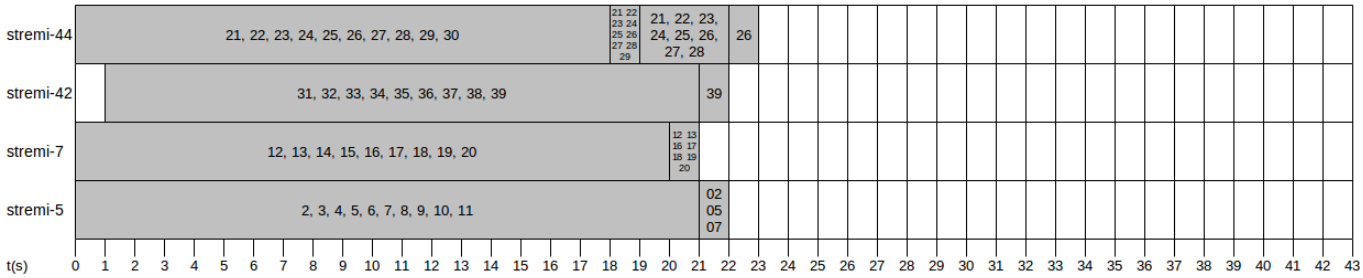

Figure 5: Container assignment with the context-aware CapacityScheduler.

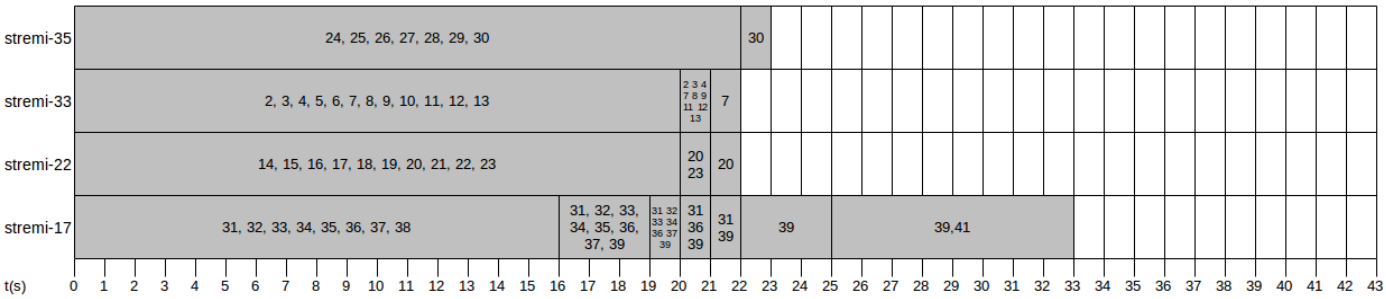

Figure 6: Container assignment in the simulated heterogeneous environment.

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.

Fig. 6 portraits the execution of TeraSort 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 and Future Work

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 [25][26]. 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.

 ISBN: 978-1-61208-353-7

ACKNOWLEDGMENT

The authors would like to thank their partners in the PER-MARE project [26] 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 [24] 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] The Apache Software Foundation, "Apache hadoop," 2014, [retrieved: Mar. 2014]. [Online]. Available: http://hadoop.apache.org/

[3] 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.

[4] 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–, [retrieved: Mar. 2014]. [Online]. Available: http://ebiquity.umbc.edu/_file_directory_/papers/623.pdf

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] M. Kirsch-Pinheiro, M. Villanova-Oliver, J. Gensel, Y. Berbers, and H. Martin, "Personalizing web-based information systems through context-aware user profiles," International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2008), 2008, pp. 231–238.

[11] D. Preuveneers and Y. Berbers, "Context-driven migration and diffusion of pervasive services on the osgi framework," International Journal of Autonomous and Adaptive Communications Systems (IJAACS), vol. 3, no. 1, Dec. 2010, pp. 3–22.

[12] C. Louberry, P. Roose, and M. Dalmau, "Kalimucho: Contextual deployment for qos management," in Distributed Applications and Interoperable Systems, ser. Lecture Notes in Computer Science, P. Felber and R. Rouvoy, Eds. Springer, 2011, vol. 6723, pp. 43–56.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010, pp. 1–9.

[19] 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.

[20] M. Baldauf and P. Musialski, "A device-aware spatial 3d visualization platform for mobile urban exploration," Fourth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2010), 2010, pp. 47–52. [Online]. Available: http://www.thinkmind.org/index.php?view=article&articleid=ubicomm_2010_3_20_10127

[21] J. Floch, C. Frà, R. Fricke, K. Geihs, M. Wagner, J. L. Gallardo, E. S. Cantero, S. Mehlhase, N. Paspallis, H. Rahnama, P. A. Ruiz, and U. Scholz, "Playing music - building context-aware and self-adaptive mobile applications," Software: Practice and Experience, vol. 43, no. 3, 2013, pp. 359–388.

[22] M. Kirsch-Pinheiro, "Requirements for context-aware mapreduce on pervasive grids," PER-MARE Deliverable D3.1, Deliverable D3.1, 2013, [retrieved: Mar. 2014]. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00858310

[23] Oracle, "Monitoring and management for the java platform," 2014, [retrieved: Mar. 2014]. [Online]. Available: http://docs.oracle.com/javase/7/docs/technotes/guides/management/

[24] Grid'5000, "Grid'5000," 2014, [retrieved: Mar. 2014]. [Online]. Available: https://www.grid5000.fr

[25] L. Steffenel, O. Flauzac, A. S. Charao, P. P. Barcelos, B. Stein, S. Nesmachnow, M. K. Pinheiro, and D. Diaz, "PER-MARE: Adaptive deployment of mapreduce over pervasive grids," in 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), Oct 2013, pp. 17–24.

[26] STIC-AmSud, "PER-MARE project," 2014, [retrieved: Mar. 2014]. [Online]. Available: http://cosy.univ-reims.fr/PER-MARE

 ISBN: 978-1-61208-353-7