View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 52 (2015) 1034 – 1039

Procedia Computer Science

International Workshop on Big Data and Data Mining Challenges on IoT and Pervasive Systems (BigD2M 2015)

Leveraging Data Intensive Applications on a Pervasive Computing Platform: the case of MapReduce

Luiz Angelo Steffenel[a,*], Manuele Kirch Pinheiro[b]

[a] *CReSTIC Laboratory, SysCom team, Université de Reims Champagne-Ardenne, France*
[b] *Centre de Recherche en Informatique, Université Paris 1 Panthéon-Sorbonne, Paris, France*

Abstract

Pervasive grids represent an important step towards the establishment of mobiquitous systems, but at the same time these environments are especially challenging when considering data distribution and data intensive processing. In this work we present CloudFIT, a middleware designed to support the volatility of pervasive environments, and discuss the main challenges related to efficiently deploy MapReduce applications on pervasive environments.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Peer-review under responsibility of the Conference Program Chairs

Keywords: Pervasive Grid; MapReduce; Hadoop; Middleware

1. Introduction

Pervasive grids can be defined as large-scale infrastructures with specific characteristics in terms of volatility, reliability, connectivity, security, etc. According to[1], pervasive grids represent the extreme generalization of the grid concept, seamlessly integrating pervasive sensing/actuating instruments and devices together with classical high performance systems. In the general case, pervasive grids rely on volatile resources that may appear and disappear from the grid, according their availability. Indeed, mobile devices should be able to come into the environment in a natural way as their owner moves,[2] and devices from different natures, from the desktop and laptop PCs until the last generation tablets, should be integrated in seamlessly way. These environments are therefore characterized by three main requirements:

- The volatility of its components, whose participation is a matter of opportunity and availability;
- The heterogeneity of these components, whose capabilities may vary on different aspects (platform, OS, memory and storage capacity, network connection, etc.);
- The dynamic management of available resources, since the internal status of these devices may vary during their participation into the grid environment.

* Corresponding author. Tel.: +33-326-913-218 ; fax: +33-326-913-397.
E-mail address: Luiz-Angelo.Steffenel@univ-reims.fr

1877-0509 © 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Peer-review under responsibility of the Conference Program Chairs
doi:10.1016/j.procs.2015.05.102

Such dynamic nature of pervasive grids represents an important challenge for executing data intensive applications. Context-awareness and nodes volatility become key aspects for successfully executing such applications over pervasive grids, but also for the handling and transmission of voluminous datasets.

In this work we present CloudFIT, a distributed computing middleware designed to support the volatility of pervasive environments. We discuss the main issues and challenges associated with the handling of data intensive applications on pervasive environments, and present an implementation of MapReduce over CloudFIT, comparing its performance against the well known Hadoop middleware[1].

The paper is structured as follows: Section 2 presents the MapReduce paradigm and discusses the main challenges for its deployment over pervasive grids, analyzing some related works. Section 3 presents the architecture of CloudFIT and its characteristics related to fault tolerance and volatility support. Section 4 introduces our implementation of a MapReduce application over CloudFIT, discussing both implementation issues and performance evaluations. Finally, Section 5 concludes this paper and sets the lines of our next development efforts.

2. MapReduce and Pervasive Grids

2.1. MapReduce

MapReduce[3] is a parallel programming paradigm successfully used by large Internet service providers to perform computations on massive amounts of data. The key strength of the MapReduce model is its inherently high degree of parallelism that should enable processing of petabytes of data in a couple of hours on large clusters.

Computations on Map-Reduce deal with pairs of key-values (k, V), and a Map-Reduce algorithm (a job) follows a two-step procedure:

1. map: from a set of key/value pairs from the input, the map function generates a set of intermediate pairs $(k_1; V_1) \rightarrow \{(k_2; V_2)\}$;
2. reduce: from the set of intermediate pairs, the reduce function merges all intermediate values associated with the same intermediate key, so that $(k_2; \{V_2\}) \rightarrow \{V_3\}$.

When implemented on a distributed system, the intermediate pairs for a given key k_2 may be scattered among several nodes. The implementation must therefore gather all pairs for each key k_2 so that the reduce function can merge them into the final result. Additional features that may be granted by the Map-Reduce implementation include the splitting of the input data among the nodes, the scheduling of the job tasks, and the recovery of tasks hold by failed nodes.

Hadoop, one of the most popular implementations of MapReduce, provides these services through a dual layered architecture where tasks scheduling and monitoring are accomplished through a master-slave platform, while the data management is accomplished by a second master-slave platform on top of the hierarchical HDFS file-system. Such master-slave architecture is not adapted to dynamic environments like pervasive grids[1], where the failure of the master may prevents the system operation.

2.2. Data-intensive applications on Pervasive Grids

In spite of a wide tradition on distributed computing environments such as Seti@Home[4], Folding@home[5] and others, we observe that most of these platforms have focused on computing-intensive parallel applications with few I/O and loose dependencies between the tasks. Enabling these environments to support data-intense applications is still a challenge, both in performance and reliability. We believe that MapReduce is an interesting paradigm for data-intensive applications on pervasive grids as it presents a simple task distribution mechanism, easily implemented on a pervasive environment, but also a challenging data distribution pattern. Enabling MapReduce on pervasive grids raises many research issues, which we can decompose in two subtopics: data distribution and data processing.

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

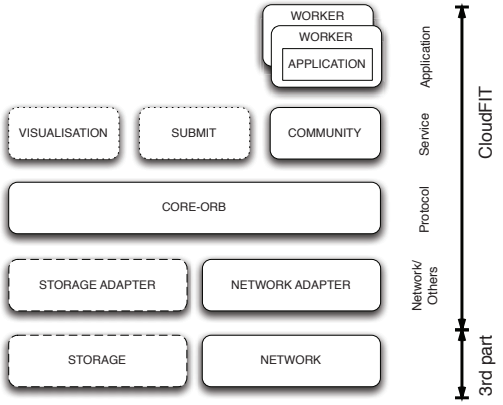

Fig. 1. CloudFIT architecture stack

There are two approaches to distribute large volume of data to large number of distributed nodes. The first approach relies on P2P protocols where peers collaboratively participate to the distribution of the data by exchanging file chunks. In [6,7], authors investigate the use of the Bittorrent protocol with the XtremWeb and BOINC Desktop Grid in the case of data-intense "bag of tasks" applications. In [8], authors propose a Hadoop-compatible middleware that relies on JXTA, deploying two overlay networks, M-net and S-net (master and slave, respectively), which mimics the master-slave coordination mechanism from Hadoop. The second approach is to use a content delivery service where files are distributed by a secure network of well-known and authenticated volunteers [9,10]. This approach is followed by [10], in which workers get their input data from a network of cache peers organized in a P2P ring.

Concerning data processing on pervasive grids, some authors have tried to improve the processing capabilities of Hadoop to take into account the volatility of the nodes. Indeed, Zaharia et al. [11] deals with heterogeneity of the supporting infrastructure, proposing a new scheduling algorithm that can improve Hadoop response time. Other authors [12,13] take similar approaches while addressing heterogeneous environments, but propose different optimizations and scheduling strategies to improve Hadoop performance.

Lin et al. [14] discuss limitations of MapReduce implementations over volatile, non-dedicated resources. They propose a system called MOON (MapReduce On Opportunistic eNvironment), which extends Hadoop in order to efficiently deal with the high unavailability of resources in desktop-based volunteer computing environments. MOON relies on a hybrid architecture, where a small set of dedicated nodes are used to provide resources with high reliability, in contrast to volatile nodes which may become inaccessible during computations.

Due to the simplicity of MapReduce processing model (map and reduce phases), data processing can be easily adapted to a given distributed middleware, which can coordinate tasks through different techniques (centralized task server, work-stealing/bag of tasks, speculative execution, etc.). Nevertheless, good performances can only be achieved through the minimization of data transfers over the network, which is one of the key aspects of Hadoop HDFS filesystem. Only few initiatives associate data-intense computing with large-scale distributed storage on volatile resources. In [15], the authors present an architecture following the super-peer approach where the super-peers serve as cache data server, handle jobs submissions and coordinate execution of parallel computations.

3. CloudFIT

In this work we present our efforts to enable MapReduce applications over the P2P distributed computing middleware CloudFIT [16,17]. The CloudFIT framework (Fig 1) is structured around collaborative nodes connected over an overlay network. CloudFIT was designed to be independent of the underlying overlay, and the current version supports the Pastry [18] overlay network as well as the PAST DHT storage mechanism [19]. Pastry is one of the most known P2P overlays and is widely employed in distributed computing environments.

In CloudFIT, the programmer needs only to decide how to divide the problem and how to compute each individual task. When executing, each node owns the different parameters of the current computations (a list of tasks and associ-

ated results) and is able to locally decide which tasks still need to be computed and can carry the work autonomously if no other node can be contacted. The status of completed tasks (optionally including the partial results from these tasks) are distributed among the nodes, contributing therefore to the coordination of the computing tasks and form a global view of the calculus.

The basic scheduling mechanism simply randomly rearranges the list of tasks at each node, which helps the computation of tasks in parallel without requiring additional communication between nodes. This simple scheduler mechanism was designed to allow idle processes to speculatively execute incomplete tasks, reducing the "tail effect" when a task is computed by a slow node. The scheduling mechanism also supports task dependencies (allowing the composition of DAGs) and can be also be improved through the use of a context module [20] that provides additional information about the nodes capacities.

Finally, fault tolerance is ensured both by the overlay (network connections, etc.) and by the computing platform. Indeed, as long as a task is not completed, other nodes on the grid may pick it up for execution. In this way, when a node fails or leaves the grid, other nodes may recover tasks originally started by the crashed node. Inversely, when a node joins the CloudFIT community, it receives an update about the tasks current status and the working data, allowing it to start working on available (incomplete) tasks.

4. MapReduce over CloudFIT

Due to its simple task model, MapReduce can be easily implemented over CloudFIT as a two successive computing jobs: one handling Map tasks and another handling Reduce tasks. Indeed, implementing MapReduce over CloudFIT is quite straightforward and can easily mimic the behavior of Hadoop. The following sections present specific details on the design choices and challenges we faced.

4.1. Map, Reduce and task dependencies

In order to implement a MapReduce application on CloudFIT, tasks inside a Map or Reduce job must be independent, all while preserving a causal relation between Map and Reduce. Therefore, several tasks are launched during the Map phase, producing a set of (k_i, V_i) pairs. Each task is assigned to a single file/data block and therefore may execute independently from the other tasks in the same phase. Once completed, the results from each task can be broadcasted to all computing nodes and, by consequence, each node contains a copy of the entire set of (k_i, V_i) pairs at the end of the Map phase. At the end of the first step, a Reduce job is launched using as input parameter the results from the map phase.

In our prototype, the number of Map and Reduce tasks was defined to roughly mimic the behavior of Hadoop, which tries to guess the required number of Map and Reduce processes. For instance, we set the number of Map tasks to correspond to the number of input files, and the number of Reduce tasks depends on the size of the dataset and the transitive nature of the data. Please note that CloudFIT may optionally perform a result aggregation after each job completion, just like the intermediate Hadoop operations called *combiners*.

Because Hadoop relies on specific classes to handle data, we tried to use the same ones in CloudFIT implementation as a way to keep compatibility with the Hadoop API. However, some of these classes were too dependent on inner elements of Hadoop, forcing us to develop our own equivalents, at least for the moment (further works shall reinforce the compatibility with Hadoop API). For instance, we had to substitute the OutputCollector class with our own MultiMap class, while the rest of the application remains compatible with both Hadoop and CloudFIT.

4.2. Data management, storage and reliability

As stated before, CloudFIT was designed to broadcast the status about completed tasks to all computing nodes, and this status may include the tasks' results. By including the results, CloudFIT ensures $n - resiliency$ as all nodes will have a copy of the data and may therefore support up to $n - 1$ simultaneous failures.

This resiliency behavior was mainly designed for computing intensive tasks that produce a small amount of data as result, as for example, a combinatorial problem where we need to count the number of solutions.. On data-intensive applications, however, $k - resiliency$ may be prohibitive as not only all nodes need to hold a copy of all task's data, but also because broadcasting several megabytes/gigabytes of results over the network is a major performance issue.

In our efforts to implement MapReduce over CloudFIT we chose a different approach to ensure the scalability of the network all while preserving a good reliability. Hence, we rely on the PAST DHT to perform the storage of tasks results as $< task\_key, task\_result >$ tuples, while the task status messages broadcast the keys from each task. As PAST perform data replication among the nodes with a predefined replication factor k, we can ensure minimal fault tolerance levels all while improving the storage performance.

4.3. Performance Evaluation

In order to evaluate the performance of CloudFIT, we implemented the traditional WordCount application and compared it against Hadoop. The experiments were conducted over 8 machines from the ROMEO Computing Center[2]. ROMEO cluster nodes are composed by bi-Intel Xeon E5-2650 2.6 GHz (Ivy Bridge) 8 cores, 32GB.

In this performance experiment we evaluate the overall execution time of both CloudFIT and Hadoop implementations when varying the total amount of data (512MB to 2GB), obtained from a corpus of textbooks from the Gutenberg Project. The results from our implementation with DHT storage are presented on Fig. 2 and represent the median of 10 runs over 8 nodes.

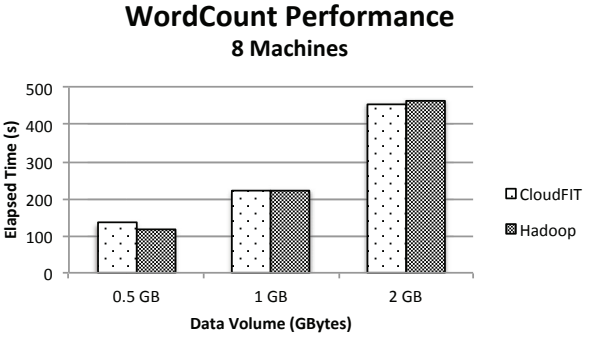

Fig. 2. WordCount MapReduce performance

When analyzing the measures, the performance of both implementations is quite similar. This is encouraging especially if we consider that CloudFIT and the access to the DHT were not optimized specifically to this kind of application. Indeed, PAST DHT is the main bottleneck for this kind of application and several actions can be performed to improve its performance.

Among the techniques to improve PAST performance we can cite the use of a mix of immutable and mutable objects. Mutable objects are useful to gather $(k; V)$ pairs from different tasks, but they force a non-negligible overhead at the DHT controller and trigger replication updates. Immutable objects don't suffer from this problem and can be used as alternative data structures in some specific situations. Another problem with P2P DHTs is that most of them do not (easily) provide information about data location. If a scheduler is able to access this information, it can select tasks that don't require data transfer over the network. Finally, the CloudFIT scheduler can be extended in order to prevent idle nodes from immediately start the execution of incomplete tasks, as this may also overload the network and the DHT with multiple requests from different nodes.

5. Conclusions

Pervasive grids represent an important step towards the establishment of mobiquitous systems in which concerns high performance computing. While the pervasive computing model has no intention to supersede classical high performance computing, there is a large domain of applications that require more flexible environments and we strongly believe that pervasive grids are especially adapted to deploy MapReduce application on enterprises, fully exploring the potential of unused (or underused) resources and therefore reinforcing the enterprises' competitiveness.

[2] `https://romeo.univ-reims.fr`

In this paper we present our efforts to implement MapReduce over a pervasive grid middleware, CloudFIT. By proposing a Hadoop-compliant API over CloudFIT, we offer MapReduce applications a transparent choice between an implementation optimized for data-intensive problems (Hadoop) and one optimized for computing intensive problems over highly dynamic environments.

From the analysis of these contributions we pointed out several elements that can contribute to the improvement of the performance of MapReduce applications over CloudFIT. For instance, our next works shall continue towards the establishment of efficient and fault-tolerant MapReduce solutions for pervasive grids.

Acknowledgment

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

References

1. 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.
2. Coronato, A., Pietro, G.D.. Mipeg: A middleware infrastructure for pervasive grids. *Future Gen Computer Systems* 2008;**24**(1):17 – 29.
3. Dean, J., Ghemawat, S.. Mapreduce: simplified data processing on large clusters. *Commun ACM* 2008;**51**(1):107–113.
4. Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.. Seti@home: an experiment in public-resource computing. *Commun ACM* 2002;**45**(11):56–61.
5. Beberg, A., Ensign, D., Jayachandran, G., Khaliq, S., Pande, V.. Folding@home: Lessons from eight years of volunteer distributed computing. In: *Proceedings of the 23rd IEEE International Symposium on Parallel Distributed Processing (IPDPS '09)*. 2009, p. 1–8.
6. Vazhkudai, S., Freeh, V., Ma, X., Strickland, J., Tammineedi, N., Scott, S.. FreeLoader: Scavenging desktop storage resources for scientific data. In: *Proceedings of Supercomputing 2005 (SC05)*. Seattle, USA; 2005, .
7. Costa, F., Silva, L., Fedak, G., Kelley, I.. Optimizing data distribution in desktop grid platforms. *Parallel Processing Letters* 2008; **18**(3):391–410.
8. Marozzo, F., Talia, D., Trunfio, P.. A peer-to-peer framework for supporting mapreduce applications in dynamic cloud environments. In: Antonopoulos, N., Gillam, L., editors. *Cloud Computing*; Computer Communications and Networks. Springer London; 2010, p. 113–125.
9. Mastroianni, C., Cozza, P., Talia, D., Kelley, I., Taylor, I.. A scalable super-peer approach for public scientific computation. *Future Gen Computer Systems* 2009;**25**(3):213–223.
10. Kelley, I., Taylor, I.. A peer-to-peer architecture for data-intensive cycle sharing. In: *Proceedings of the first international workshop on Network-aware data management (NDM '11)*. New York, NY, USA: ACM; 2011, p. 65–72.
11. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.. Improving mapreduce performance in heterogeneous environments. In: *Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation*; OSDI'08. Berkeley, CA, USA; 2008, p. 29–42.
12. 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.
13. Ahmad, F., Chakradhar, S.T., Raghunathan, A., Vijaykumar, T.N.. Tarazu: optimizing mapreduce on heterogeneous clusters. *SIGARCH Comput Archit News* 2012;**40**(1):61–74.
14. Lin, H., Ma, X., Archuleta, J., Feng, W., Gardner, M., Zhang, Z.. Moon: Mapreduce on opportunistic environments. In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*. 2010, p. 95–106.
15. Cesario, E., Mastroianni, C., De Caria, N., Talia, D.. Distributed data mining using a public resource computing framework. *Grids, P2P and Service Computing* 2010;.
16. Steffenel, L., Flauzac, O., Charao, A.S., Barcelos, P.P., Stein, B., Nesmachnow, S., et al. PER-MARE: Adaptive deployment of mapreduce over pervasive grids. In: *Proc. 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. 2013, .
17. Steffenel, L.A.. First steps on the development of a P2P middleware for map-reduce. 2013.
18. Rowstron, A., Druschel, P.. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. 2001, p. 329–350.
19. Rowstron, A., Druschel, P.. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In: *18th ACM Symposium on Operating Systems Principles (SOSP'01)*. 2001, p. 188–201.
20. Steffenel, L., Flauzac, O., Charao, A., Barcelos, P., Stein, B., Cassales, G., et al. Mapreduce challenges on pervasive grids. *J of Computer Science* 2014;**10**(11).