2013 Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing

PER-MARE: Adaptive Deployment of MapReduce over Pervasive Grids

Luiz Angelo Steffenel*, Olivier Flauzac*, Andrea Schwertner Charão†, Patricia Pitthan Barcelos†,
Benhur Stein†, Sergio Nesmachnow‡, Manuele Kirsch Pinheiro§ and Daniel Diaz§
*Université de Reims Champagne-Ardenne, Reims, France
{luiz-angelo.steffenel,olivier.flauzac}@univ-reims.fr
†Universidade Federal de Santa Maria, Santa Maria, Brazil
{andrea,pitthan,benhur}@ufsm.br
‡Universidad de la República, Montevideo, Uruguay
sergion@fing.edu.uy
§Université Paris 1 Panthéon-Sorbonne, Paris, France
{manuele.kirsch-pinheiro,daniel.diaz}@univ-paris1.fr

***Abstract*—MapReduce is a parallel programming paradigm successfully used to perform computations on massive amounts of data, being widely deployed on clusters, grid, and cloud infrastructures. Interestingly, while the emergence of cloud infrastructures has opened new perspectives, several enterprises hesitate to put sensible data on the cloud and prefer to rely on internal resources. In this paper we introduce the PER-MARE initiative, which aims at proposing scalable techniques to support existent MapReduce data-intensive applications in the context of loosely coupled networks such as pervasive and desktop grids. By relying on the MapReduce programming model, PER-MARE proposes to explore the potential advantages of using free unused resources available at enterprises as pervasive grids, alone or in a hybrid environment. This paper presents the main lines that orient the PER-MARE approach and some preliminary results.**

***Keywords*-Pervasive computing; MapReduce; Big data**

I. Introduction

The MapReduce programming paradigm [1] has become a popular solution for rapid implementation of distributed data-intensive applications, being supported on both grid and cloud environments. Interestingly, while the emergence of cloud infrastructures has opened new perspectives, several enterprises hesitate to migrate their applications to the cloud, as clouds present several security issues, when compared to private or managed infrastructures such as grids or clusters. When dealing with sensitive data, these enterprises prefer therefore to rely on private infrastructures to develop their applications. By proposing a pervasive grid environment supporting a programming model such as MapReduce, we can intend to explore the potential advantages of using free unused resources structured as a pervasive grid, be it alone or in a hybrid environment.

The PER-MARE initiative aims at the adaptation of a popular MapReduce distribution for pervasive and desktop grids, proposing scalable techniques to support existent MapReduce-based, data-intensive applications, but in the context of loosely coupled networks such as pervasive and desktop grids.

We consider pervasive grids as a type of large-scale infrastructure with specific characteristics in terms of volatility, reliability, connectivity, security, etc. In the general case, pervasive grids rely on resources contributed by volunteers. Desktop grids are a particular case of pervasive grids leveraging unused processing cycles and storage space available within the enterprise. These environments present challenging deployment and context-awareness constraints, as heterogeneity, fault tolerance and resource volatility facts may strongly impact the performance of such networks. Although some works tried to address this problem before, PER-MARE initiative innovates by adopting a two-fold development approach: (*i*) on one hand, we wish to adapt a well-known MapReduce implementation (Hadoop, for instance), including context-aware elements that may allow its efficient deployment over a pervasive or desktop grid; and (*ii*) on the other hand, we shall work on the implementation of a Hadoop-compatible API over a P2P distributed computing environment originally meant for pervasive grids.

We believe that this double process will bring us better insights on the deployment of MapReduce over pervasive grids. In this paper, we introduce the PER-MARE approach and some preliminary results. We present PER-MARE vision about a context-aware MapReduce, which intends to handle high volatility of pervasive grid resources by applying context-awareness techniques on tasks distribution and scheduling. Such volatility, represented mainly by nodes churn, affects both task and data distribution. Context-awareness will allow to better adapt MapReduce applications to this dynamic environment, but an appropriate and fault tolerant infrastructure is also needed. We propose here a P2P MapReduce implementation, which intends to be an alternative implementation compatible with Hadoop API and pervasive grids. The preliminary results of such implementation are also presented in this paper. We believe that providing a context-aware behavior to MapReduce applications while keeping the compatibility with the API from Hadoop will contribute to easily deploy existent applications

978-0-7695-5094-7/13 $31.00 © 2013 IEEE
DOI 10.1109/3PGCIC.2013.10

over pervasive grids and therefore evaluate the performance and fault-tolerant issues related to such environments.

The rest of the paper is organized as follows: Section 2 presents an overview on related work. Section 3 introduces the problem statement. Section 4 introduces PER-MARE approach and proposals. Section 5 presents our preliminary results, before concluding on Section 6.

II. Background and Related Works

Data-intensive distributed computing is an active research topic. The approaches to this problem include programming paradigms and supporting infrastructures. In this section, we present the MapReduce paradigm and discuss some important issues and related works concerning the deployment of MapReduce applications over pervasive and desktop grid infrastructures.

A. About MapReduce

MapReduce [1] is a parallel programming paradigm successfully used by large Internet service providers to perform computations on massive amounts of data. After being strongly promoted by Google, it has also been implemented by the open source community through the Hadoop project, maintained by the Apache Foundation and supported by Yahoo! and even by Google itself. This model is currently getting more and more popular as a solution for rapid implementation of distributed data-intensive applications. The key strength of the MapReduce model is its inherently high degree of potential parallelism that should enable processing of petabytes of data in a couple of hours on large clusters consisting of several thousand nodes.

A MapReduce computation takes a set of input key/value pairs, and produces a set of output key/value pairs. The user of the MapReduce paradigm expresses the computation through two functions:

1) map, that processes a key/value pair to generate a set of intermediate key/value pairs; and
2) reduce, that merges all intermediate values associated with the same intermediate key. The framework takes care of splitting the input data, scheduling the jobs' component tasks, monitoring them, and re-executing the failed ones.

Furthermore, when associated with a distributed filesystem (HDFS, in the case of Hadoop), MapReduce can improve its performance by minimizing data transfers over the network.

A few typical examples of simple MapReduce applications include counting URL Access Frequency by processing web page requests, creating reverse Web-link graph or an inverted index from large set of documents.

B. Data-intensive applications on Pervasive and Desktop Grids

Although desktop grids have been very successful with projects such as Seti@Home [2], Folding@home [3] and others, data-intense computing on these environments is a still a promising area: for now, desktop grids have mostly focused on loosely coupled parallel applications with few I/O and without dependencies between the tasks. Some major achievements combining their huge storage potential with their processing capability are expected. They would impact the applications requiring an important volume of data input storage with frequent data reuse and limited volume of data output. The MapReduce programming model adapts well to this class of applications, and there is a growing interest in supporting MapReduce on desktop grids.

Since enabling MapReduce on pervasive and desktop grids raises many research issues, we can decompose this problem in two subtopics: data distribution and storage and data processing.

There are two approaches to distribute large volume of data to large number of nodes distributed on Internet. The first approach relies on P2P protocols where peers collaboratively participate to the distribution of the data by exchanging file chunks. In [4] and [5], authors investigate the use of the Bittorrent protocol with the XtremWeb and BOINC Desktop Grid in the case of data-intense bag of tasks application. [6] relies on a JXTA platform, deploying two overlay networks, M-net and S-net (master and slave, respectively), which mimics the master-slave coordination mechanism from Hadoop. Note that if the P2P approach seems efficient, it assumes that volunteers would agree that their PC connects directly to another participant's machine to exchange data. Unfortunately, this could be seen as a potential security threat. It is unlikely to be widely accepted by users. This drawback has so far prevented adoption of P2P protocol by major volunteer computing projects. The second approach is to use a content delivery approach where files are distributed by a secure network of well-known and authenticated volunteers [7] [8]. This approach is followed by the ATTICS project [9] (Peer-to-Peer Architecture for Data-Intensive Cycle Sharing). Instead of retrieving files from a centralized server, workers get their input data from a network of cache peers organized in a P2P ring.

Several systems have been proposed to aggregate unused storage of desktop workstation within a LAN. Farsite [10] builds a virtual centralized file system over a set of untrusted desktop computers. It provides file reliability and availability through cryptography, replication and file caching. Freeloader [4] fulfills similar goals but unifies data storage as a unique scratch/cache space for hosting immutable datasets and exploiting data locality, allowing to persistently store data on desktop PCs.

Lin et al. [11] 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. Their goal is somewhat similar to ours, but their solution based on dedicated nodes does not fit well to more dynamic environments as pervasive grids.

Due to the simplicity of its 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 [12], 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.

The deployment of MapReduce over pervasive and desktop grids exposes most of the challenges presented in this section with respect to data distribution, storage and processing. At the moment there is no single solution that solves all these issues together. When considering pervasive grids, where heterogeneity is a major characteristic, data processing/scheduling must be driven by contextual information (resources characteristics, node reliability, network performance, data location) in order to achieve the expected processing performance.

C. Adaption to the Computational Context

As cloud computing has leveraged the use of MapReduce, it is natural that most MapReduce distribution have been tailored to such environments. For instance, most IaaS clouds use sets of virtual machines that share similar characteristics such as computational power and memory and, in such cases, MapReduce does not require specific adaption to the computational context as all virtual machines are similar. As a consequence, most users simply rely on MapReduce default configurations such as the number of reduce tasks by machine, the maximum memory, etc. Although this behavior can be modified through property files, there is no mechanism to automatically detect and modify these parameters. When dealing with a heterogeneous environment such as a pervasive grid, MapReduce must be able to automatically tune to the nodes characteristics.

While the computational context is tightly related to the processing power of the resources, it also impacts other aspects such as fault-tolerance and data storage. Indeed, Hadoop allow a certain number of duplicated processes/data in order to circumvent fault situations. If the context on pervasive grid is not considered, tasks may be inefficiently allocated or even disappear if the node volatility is high. Similarly, HDFS tries to place data for the map and reduce phases as closes as possible to the processes/tasks that will need it, as to reduce the (slow) access over the network. In a pervasive grid, the placement policy must account also on the volatility and speed of the resources, preventing data losses. While the contextual information required for the adaption of MapReduce can be obtained from the system properties (CPU and network speed, number of cores, memory size, etc.), the diffusion and analysis of such information must be tightly integrated into the MapReduce framework to boost the platform efficiency. For this reason, context-awareness [13] and context distribution [14] are important elements to be considered.

III. Problem statement

One of the first challenges a user faces when deploying MapReduce is that its most known and popular implementation, Hadoop, requires a highly structured environment such as a grid or a cloud to be deployed. For instance, Hadoop relies on a collection of tools (Hadoop Core, HDFS, etc.) developed by different Apache subprojects, which interact through a complicate set of master and slave daemons. As a result, Hadoop installation, although well documented, requires a stable set of machines known at startup time. Please note that the installation procedure lacks of automatic context adaption, forcing the administrator to manually define the characteristics of the resources, such as the number of cores of each machine.

Together, these elements prevent a user to quickly launch MapReduce over unused internal resources (e.g. an enterprise desktop grid) or over a volunteer network where nodes join and leave the network dynamically. Some authors [6], [15] addressed this problem by developing P2P frameworks compatible with the MapReduce paradigm. While proposing interesting solutions for the distribution and fault tolerance issues, these frameworks have their own APIs that are not compatible with application codes written for Hadoop.

Our project is precisely addressing this point: proposing scalable techniques to support existent Hadoop applications in the context of loosely coupled networks such as pervasive grids. We consider pervasive grids as a type of large-scale infrastructure with specific characteristics in terms of volatility, reliability, connectivity, security, etc. According to Parshar and Pierson [16], pervasive grids represent the extreme generalization of the grid concept, in which the resources are pervasive. For these authors, pervasive grids seamlessly integrate pervasive sensing/actuating instruments and devices together with classical high performance systems. In the general case, pervasive grids rely on resources contributed by volunteers, but these resources are extremely volatile. They may appear and disappear from the grid, according their availability. Desktop grids are a particular case of pervasive grids leveraging unused processing cycles and storage space available within the enterprise.

However, contrarily to simple desktop grids, general pervasive grids have to deal with a more dynamic environment in a transparent way. Indeed, mobile devices should be able to come into the environment in a natural way, as their owner moves [17]. Devices from different natures, from the desktop and laptop PCs until the last generation tablets, should be integrated in seamlessly way. It results an environment characterized by: (*i*) the volatility of its components, whose participation on the grid is notably a matter of opportunity and availability; and (*ii*) by the heterogeneity of these components, whose capabilities may vary on different aspects (platform, OS, memory and storage capacity, network connection, etc.). Besides, the internal status of these devices may also vary during their participation into the grid environment. For instance, during the execution of a job, a mobile device integrated to a pervasive grid may change its network connection, passing from a fixed connection to a wireless one. The same can be observed with the available memory: after starting a job, device's owner may start new applications that modify device memory status.

Pervasive grids environments have to deal with such additional constraints related to the heterogeneity and volatility of the resources. In such environments, it is essential to adapt the application to the network variable behavior and to coordinate the resources (task scheduling, data placement, etc.). According to Coronato & De Pietro [17], pervasive grid environments should be able to self-adapt and self-configure in order to incoming mobile devices. We strongly believe that context-awareness is needed in order to support such self-adaption. Context-awareness can be defined as the ability of a system to adapt its operations to the current context, aiming at increasing usability and effectiveness by taking environmental context into account [18]. In order to support environments changes, context-awareness becomes a critical aspect to boost the efficiency of the applications over pervasive grids.

Besides, considering the extreme development of mobile devices inside organizations nowadays, the opportunistic use of such resources as an internal pervasive grid appears as an interesting alternative for those who hesitate to distribute sensible data over cloud infrastructures. These still suffer from security issues that prevent their application in some cases. Nevertheless, in order to be fully operational, pervasive grids environment have to first tackle problems related to its dynamic nature.

IV. The PER-MARE approach

Given the problems presented above, we propose to address the lack of context adaptation of MapReduce applications over pervasive grids all while keeping the compatibility with MapReduce most popular implementation, Hadoop. To meet this global goal, several aspects need to be investigated.

Our approach is to improve the behavior of MapReduce-based applications on pervasive grids using a two-fold investigation method. Hence, to better understand the elements that may impact the deployment of MapReduce over pervasive grids, our teams investigate the problem through two different approaches: on the one hand, we shall modify Hadoop as to implement automatic tuning of the nodes, simplifying therefore the deployment procedure over pervasive nodes. On the other hand, the second approach relies on the adaptation of a distributed computing platform (CONFIIT [19]) to make it compatible with the MapReduce paradigm and, more specifically, with Hadoop's API.

We believe that this double approach is essential to understand and cover all the facets of the problem. By comparing these two approaches "side by side" we can propose effective solutions for pervasive grids.

In order to validate our developments, we target a scalable execution across multiple sites with real-life workloads from existing applications. A distributed desktop grid among our institutions shall provide important insights on the adaptability to the heterogeneity of resources and the dynamic nature of the networks. In addition, the Grid5000 platform (https://www.grid5000.fr) will provide the additional experimental infrastructure to explore the scalability issues of our work.

A. Towards context-aware MapReduce

In order to cope with pervasive grids, MapReduce implementations have to deal with issues mainly related to nodes volatility and heterogeneity of such environments. Indeed, dynamic nature of pervasive environments demands adapting to changing operating conditions and to enable more efficient and effective operation while avoiding system failure [20]. Context-awareness is then needed in order to cope with such changing environments.

Unfortunately, as discussed previously, Hadoop was not designed to support such changing environment, with resources that may appear and disappear from the environment or new resources that can be integrated opportunistically, neither with active resources whose capabilities may also vary during the execution. Although several MapReduce implementations can handle nodes disconnection, most of them cannot support a smooth re-connection of a node after a disconnection period or the addition of a new node. Indeed, Hadoop only allow new nodes to join the network through the restart of the entire daemon network. This means that they cannot fully handle mobile devices, which are often characterized by frequent disconnection and re-connection. They cannot either discover for new nodes that were not previously declared in the configured cluster. Such behavior limits the use of such implementations, and mainly Hadoop, over pervasive grids.

In order to deal with such limitation, PER-MARE intends to propose a context-aware implementation of Hadoop. Our proposal is to introduce into Hadoop a lightweight context middleware, allowing to dynamically feed Hadoop platform

and applications with context information. Such information could then be used for data and task scheduling, in order to fully benefit from pervasive grids.

Nevertheless, to reach this goal, several challenges should be tackled. First, nodes availability should be observed. New nodes should be integrated on the grid on the fly, as well as disconnected nodes should be able to reintegrate the grid once available again. This means that nodes should be able to dynamically join and leave pervasive grid. A P2P behavior is needed, in which nodes discover other nodes and inform them about their availability. Such P2P behavior supposes to extend heartbeat mechanism traditionally used by Hadoop to identify nodes failures in order to allow nodes to join the grid during execution.

Context information about the nodes themselves should also be observed, and this without a significant impact on the nodes performance. Different context information can be observed: node location, available memory, storage, network latency, etc. Such context information set must be extensible. New context information should be easily integrated on it. A plugin-based mechanism, such as [21], can be used in order easily plug new sensor capabilities on the platform.

Besides, context information about a node should be relayed to the others nodes in order to allow an efficient data and task schedule inside the pervasive grid. In other words, a context distribution mechanism, similar to [14], should be considered. Such mechanism should integrate heartbeat mechanism in order to couple context information with nodes-alive signal, forming a lightweight P2P context distribution mechanism.

In order to handle such challenges, we need first to study the feasibility of such proposals on Hadoop platform. Hadoop is composed by a complex set of daemons and components involved on the multiple aspects of HDFS and MapReduce execution. Modify such ecosystem can reveal itself a delicate task. Thus, an initial study about Hadoop internal infrastructure has already started [22], [23]. By using failure injection techniques, this study has demonstrated Hadoop vulnerabilities concerning nodes failures, notably when such failures concern master data nodes. This confirms our opinion that more flexible mechanisms are necessary in order to allow Hadoop applications to fully benefit from pervasive grid environments. Our feasibility study is still in process, and we expect to publish results soon.

B. MapReduce on a P2P distributed computing environment

Due to its simple task model, MapReduce can be easily implemented in a distributed computing environment. In our project, we rely on the P2P distributed computing middleware like CONFIIT (Computation Over Network for FIIT) [19]. In CONFIIT, the programmer needs to decide how to divide the problem into a finite number of independent tasks, and how to compute each individual task. This is the same principle of MapReduce *map* and *reduce* steps, which can be considered as a sequences of Finite number of Independent and Irregular Tasks (FIIT [24]) problems.

The CONFIIT framework is structured around collaborative nodes connected over a logical oriented ring overlay network. Communication between nodes is achieved using a token-like mechanism, which carries the state of computation around the ring. Task status (and partial results) are broadcasted among the nodes, which contributes to the coordination of the computing tasks and form a global view of the calculus.

A node owns the different parameters of the current computations (a list of tasks and associated results). It 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. If later a node reintegrates a community, it is able to share the results from the tasks it completed and re-synchronize its task's list. A simple scheduling mechanism randomly rearrange the list of tasks at each node, which helps the computation of tasks in parallel without requiring additional communication between nodes.

1) Programming models:

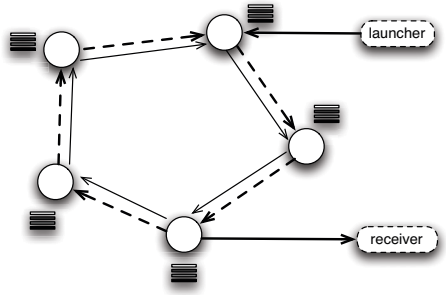
Since constraints of a given application could be different and sometimes in contradiction (fault tolerance, efficiency, etc.), CONFIIT offers two main programming models: distributed and centralized mode.

The **distributed mode** allows a high fault tolerance level in the computation since task results are distributed to each node in the community. Thus, a broken computation can be re-launched using already computed tasks. Figure 1(a) shows information exchanges in the community for a distributed application. At first, the launcher sends the computing request to a node. The request is propagated along the community. During computation, results of individual tasks are propagated across the community such that each node could locally store all individual results (data blocks). Concurrently to the computations, information on the global computation is also exchanged among nodes.Besides, it is worth noting that the launcher only needs to be connected during the initiation phase. At the end of the computation, the global result can be retrieved from any node in the community.

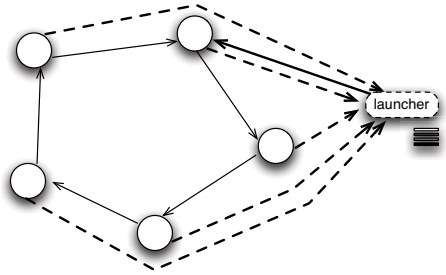
The **centralized mode** concentrates the storage on a single node (the job launcher), which reduces the global load of storage space, but reduces fault tolerance. As this mode is too restrictive for a pervasive environment, we prefer to rely on the Distributed mode on the remaining of this paper.

2) Using CONFIIT *for MapReduce:*

CONFIIT offers an interesting platform for building a P2P MapReduce implementation. Its fault tolerant characteristics make it particularly appropriate for pervasive grid. Thus, we are introducing into CONFIIT a MapReduce implementation inspired by Hadoop API. This first implementation intends allowing comparing traditional Hadoop implementation with a P2P implementation of MapReduce, leveraging advantages and inconvenients that shall be further explored in our future



(a) Distributed mode



(b) Centralized mode

Figure 1. CONFIIT Programming modes

works.

When using CONFIIT, a MapReduce job can be expressed as a two rounds execution, one handling Map tasks and another handling Reduce tasks. Similarly to Hadoop, during Map phase, several tasks are launched according the number of input files. The number of tasks during the Reduce phase is calculated based on the number of available nodes. Once a round starts, each node starts a task from the shared task list, and broadcasts its results at the end of the task's computation.

When using the distributed mode, MapReduce implementation over CONFIIT supports nodes failures as well as nodes volatility, allowing nodes to dynamically leave and join the grid. Indeed, as long as a task is not completed, other nodes on the grid may pick it up. In this way, when a node fails or leaves the grid, other nodes may recover tasks originally taken by the crashed node. Inversely, when a node joins the CONFIIT community, it receives a copy of the working data and may pick up available (incomplete) tasks on the shared task list. Such P2P-like behavior offers a better support for more dynamic environments such as pervasive grids, and also allows the joining of additional nodes, a feature that usually lacks on Hadoop.

In the next session, we present the first results obtained with a MapReduce prototype over CONFIIT. Besides, we are also working on supplying on CONFIIT a fully compatible Hadoop API. Even if it is already possible to build MapReduce-like applications over CONFIIT, porting Hadoop applications to CONFIIT still requires additional adaptations. We believe that by offering a fully compatible API, migrating application from Hadoop API to CONFIIT will be easier, offering an alternative P2P execution platform for existent applications and also as a cheap testing platform.

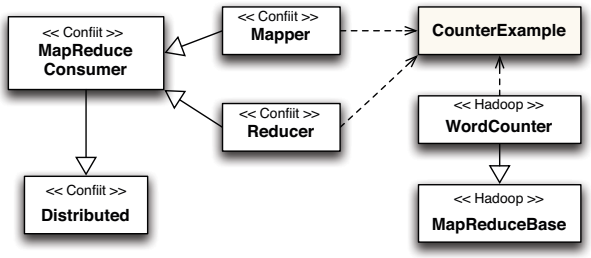


Figure 2. Experimental application shared between Hadoop and Confiit implementations

V. PRELIMINARY RESULTS

In a first step to deploy MapReduce applications over a pervasive network, we implemented a prototype version of the classic `WordCount` application over CONFIIT. This prototype uses a core application, independent from CONFIIT or Hadoop, as show in Figure 2. This core application could then be used on both prototypes, allowing a clear comparison between both solutions. Besides, as illustrated in Figure 2, in this first tentative, we rely on the Distributed mode for both Map and Reduce parts, as it has interesting fault tolerance properties that can be useful in a volatile environment, such as full replication of partial results.

Our prototype reproduces the Map and Reduce phases with two CONFIIT instances, one for each phase. The computation of each task calls the `map()` or `reduce()` methods from the core classes shared with the Hadoop implementation. In the case of the Map job, each task returns a list of <K', V'> elements. For the Reduce job, task solvers access the results from the previous job and can therefore compute a word count <K', V">. Because the Distributed mode replicates all results over the entire CONFIIT community, the Reduce tasks can read the results from the Map instance directly from their hard drives.

While most parts of a MapReduce application can be directly mapped to CONFIIT methods, one single difference resides on the need to indicate the number of computing tasks. Indeed, this behavior is automatized on Hadoop, which tries to guess the required number of Map and Reduce processes. In our prototype, this parameter was defined as to mimic the behavior of Hadoop, i.e., by setting a number of Map tasks to roughly correspond to the number of input files and the number of Reduce tasks to correspond to the number of computing cores in the CONFIIT nodes at the time Reduce starts (this number may vary later, due to nodes volatility).

The experiments were conducted over 16 machines on a Grid'5000 cluster [1]. Each machine is composed by 2 AMD

[1] http://www.grid5000.fr

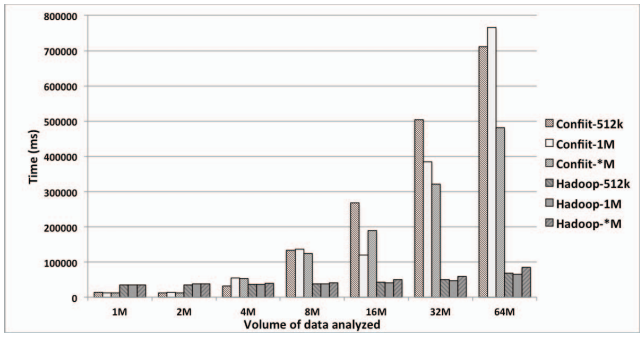


Figure 3. Comparison between Confiit and Hadoop over 16 nodes

Opteron 275 2.2GHz CPUs, totalizing 4 cores per node. A Gigabit Ethernet interconnects the nodes.

For the experiments, we evaluate the performance of both CONFIIT and Hadoop solutions when varying the total amount of data and the number/size of input files. For each data size, we measure 3 different input splits: one single file, 1MB splits and 512kB splits. The reason for such approach is to analyse the impact of the input files on the map step from both solutions. For the input data, we chose the Gutenberg Project *Science Fiction Bookshelf CD*[2], which contains more than 200 books in text format. The results presented on Figure 3 represent the median of the performed measures.

When analyzing the measures, two scenarios arise: for small data volumes, our prototype outperforms Hadoop, while Hadoop performs much better for large data sets. In the first scenario, this is mostly due to CONFIIT lightweight middleware. However, when the data volume augments, we observe a huge performance slowdown. Investigation shows that this is due to the task update pattern used on the Distributed mode, which overloads the token passing mechanisms and creates a bottleneck.

More specifically, the current implementation of CONFIIT Distributed mode (see Figure 1(a)) spreads results using "service" layer messages: the same messages that are used to keep nodes updated about the tasks completion are used to transmit the tasks results. For small data sets this procedure poses no problem but when the amount of data grows, the service layer becomes overloaded. As a consequence, nodes finish by computing most of the tasks locally because few updates are able to reach to the other nodes.

Future works will focus on correcting these problems, improving massive data exchanges among CONFIIT nodes. This can be achieved through the use of specific data exchange channels (created on-demand by nodes that wish to complete their data sets after receiving an update message), or through a third-part P2P filesystem. Indeed, the use of a DHT with controlled data replication is also an interesting solution to ensure fault tolerance without relying on a full replication of all data (an especial concern in the case of BigData). The development of a context-awareness scheduling shall also help on this effort, as data distribution and data locality are successful elements of the Hadoop framework.

[2]http://www.gutenberg.org/wiki/Gutenberg:The_CD_and_DVD_Project

VI. CONCLUSION

The MapReduce programming paradigm and its most known implementation, Hadoop, are becoming increasingly popular. Particularly designed for distributed data-intensive applications, MapReduce is being largely supported on both grid and cloud environments. Nevertheless, Hadoop, similarly to other MapReduce implementations, is designed for dedicate environments. It does not support dynamic environments, which are subject to nodes volatility and changes on internal node state. As a consequence, Hadoop does not fit pervasive grid environments, which are characterized by such dynamism and by an opportunistic use of heterogeneous nodes.

In this paper, we presented the PER-MARE initiative, which aims at adapting MapReduce to pervasive grids. PER-MARE proposes an innovative two-fold approach, aiming at, on the one hand, adapting Hadoop implementation by adding on it a context-aware behavior, and on the other hand, proposing a P2P Hadoop compatible implementation, based on CONFIIT platform. This paper presented the bases of this two-fold approach and some preliminary results. Indeed, although in an early stage, PER-MARE has already some stimulating results. A first prototype of P2P MapReduce over CONFIIT is already available at or project website[3] and we are currently working on a Hadoop compatible API for CONFIIT. Besides, we are currently analyzing Hadoop internal infrastructure in order to allow Hadoop to support nodes volatility, and especially dynamic joining of new nodes.

Handling nodes volatility is a first step towards a context-aware implementation of Hadoop. This next step intends to allow a smart task and data schedule, based on node context information such as location, available memory and storage, network latency, etc. This context-aware effort will be extended also to CONFIIT and our P2P MapReduce implementation. In this case, context information will be mainly used for adapting P2P data distribution over available nodes, according execution context of each node.

ACKNOWLEDGMENTS

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

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

REFERENCES

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

[2] D. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "Seti@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, pp. 56–61, Nov. 2002. [Online]. Available: http://doi.acm.org/10.1145/581571.581573

[3] A. Beberg, D. Ensign, G. Jayachandran, S. Khaliq, and V. Pande, "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, pp. 1–8.

[4] S. Vazhkudai, V. Freeh, X. Ma, J. Strickland, N. Tammineedi, and S. Scott, "FreeLoader: Scavenging desktop storage resources for scientific data," in *Proceedings of Supercomputing 2005 (SC'05)*, Seattle, USA, 2005.

[5] F. Costa, L. Silva, G. Fedak, and I. Kelley, "Optimizing data distribution in desktop grid platforms," *Parallel Processing Letters*, vol. 18, no. 3, pp. 391–410, Sep. 2008.

[6] F. Marozzo, D. Talia, and P. Trunfio, "A peer-to-peer framework for supporting mapreduce applications in dynamic cloud environments," in *Cloud Computing*, ser. Computer Communications and Networks, N. Antonopoulos and L. Gillam, Eds. Springer London, 2010, pp. 113–125.

[7] C. Mastroianni, P. Cozza, D. Talia, I. Kelley, and I. Taylor, "A scalable super-peer approach for public scientific computation," *Future Gener. Comput. Syst.*, vol. 25, no. 3, pp. 213–223, 2009.

[8] I. Kelley and I. Taylor, *Bridging the Data Management Gap Between Service and Desktop Grids*. Springer, 2008.

[9] ——, "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, pp. 65–72.

[10] A. Adya, W. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer, "Farsite: federated, available, and reliable storage for an incompletely trusted environment," *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 1–14, 2002.

[11] H. Lin, X. Ma, J. Archuleta, W. Feng, M. Gardner, and Z. Zhang, "Moon: Mapreduce on opportunistic environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10)*, 2010, pp. 95–106.

[12] E. Cesario, C. Mastroianni, N. De Caria, and D. Talia, "Distributed data mining using a public resource computing framework," *Grids, P2P and Service Computing*, 2010.

[13] D. Preuveneers, K. Victor, Y. Vanrompay, P. Rigole, and M. Kirsch-Pinheiro, *Context-Aware Adaptation in an Ecology of Applications*. IGI Global, 2009, ch. 1, pp. 1–25.

[14] M. Kirsch-Pinheiro, Y. Vanrompay, K. Victor, Y. Berbers, M. Valla, C. Fra, A. Mamelli, P. Barone, X. Hu, A. Devlic, and G. Panagiotou, "Context grouping mechanism for context distribution in ubiquitous environments," in *OTM Conferences (1)*, ser. Lecture Notes in Computer Science, R. Meersman and Z. Tari, Eds., vol. 5331. Springer, 2008, pp. 571–588.

[15] DiscoProject. http://discoproject.org/.

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

[17] A. Coronato and G. D. Pietro, "Mipeg: A middleware infrastructure for pervasive grids," *Future Generation Computer Systems*, vol. 24, no. 1, pp. 17 – 29, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X07000593

[18] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 2, no. 4, pp. 263–277, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1504/IJAHUC.2007.014070

[19] O. Flauzac, M. Krajecki, and L. Steffenel, "Confiit: a middleware for peer-to-peer computing," *Journal of Supercomputing*, vol. 53, no. 1, pp. 86–102, July 2010.

[20] A. Ferscha, Ed., *Pervasive Adaptation : Next generation pervasive computing research agenda*. Institute for Pervasive Computing, Johannes Kepler University Linz, 2011. [Online]. Available: http://www.perada.eu/research-agenda/

[21] N. Paspallis, "Middleware-based development of context-aware applications with reusable components," Ph.D. dissertation, University of Cyprus, 2009.

[22] E. Gondim, B. Prates, P. P. Barcelos, and A. Charão, "Análise de alternativas para injeção de falhas no Apache Hadoop," in *Proceedings of the XII Simpósio em Sistemas Computacionais*, Vitória, ES, Brazil, 2011.

[23] E. Gondim, P. P. Barcelos, and A. S. Charão, "Explorando o framework de injeção de falhas do Apache Hadoop," in *Proceedings of the XIII Simpósio em Sistemas Computacionais*, Petrópolis, RJ, Brazil, 2012.

[24] M. Krajecki, "An object oriented environment to manage the parallelism of the FIIT applications," in *Parallel Computing Technologies, 5th International Conference, PaCT-99*, ser. Lecture Notes in Computer Science, V. Malyshkin, Ed. St. Petersburg, Russia: Springer-Verlag, Sep. 1999, vol. 1662, pp. 229–234.