Requirements for Context-aware MapReduce on Pervasive Grids

Manuele Kirsch Pinheiro

PER-MARE - Pervasive Map-Reduce
Project number 13STIC07
CAPES/MAEE/ANII STIC-AmSud collaboration program

Deliverable 3.1

Requirements for Context-aware MapReduce on Pervasive Grids

Manuele Kirsch Pinheiro

Reviewer: Luiz Angelo Steffenel

Partner	Author's information
Universidade Federal de Santa Maria 1960	
– PANTHÉON - SORBONNE – UNIVERSITÉ PARIS 1	*Manuele Kirsch Pinheiro* Centre de Recherche en Informatique 90 rue Tolbiac, 75013 Paris, France Manuele.Kirsch-Pinheiro@univ-paris1.fr
UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE	*Luiz Angelo Steffenel* CReSTIC – SysCom - Université de Reims Champagne-Ardenne UFR Sciences Exactes et Naturelles Département de Mathématiques et Informatique - BP 1039 51687 REIMS CEDEX 2 Luiz-Angelo.Steffenel@univ-reims.fr
UNIVERSIDAD DE LA REPUBLICA URUGUAY	

Abstract

PER-MARE project aims at adapting MapReduce distribution to pervasive grids. Such adaptation includes implementing context-awareness capabilities into Hadoop distribution. The main purpose of this deliverable is to analyze the requirements for including such support for context-awareness on Hadoop. 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 [7]. Such environment corresponds, in the case of PER-MARE project, to a pervasive grid environment and its dynamic resources. It is then important to analyze such environment and its main characteristics, as well as the impact of observing such environment on Hadoop distribution. Indeed, performance of MapReduce applications should not be negatively impacted by context-awareness capabilities, which makes important to carefully consider context-awareness needs and implications of Hadoop distribution and on MapReduce application over pervasive grids. This deliverable intends to point out a set of requirements for context-aware support on PER-MARE project. These requirements will guide subsequent proposals on this domain.

Table of Contents

1 Introduction

Context-awareness is one of the main goals of PER-MARE project. PER-MARE intends adapting popular MapReduce distribution to pervasive grids, which are characterized by dynamicity of its resources. Context-awareness becomes then a key element for successfully deployment of applications on such pervasive environments. Nonetheless, before stating for context-support on PER-MARE project, we need to understand what is context and context-awareness.

Context-awareness can be defined as the ability of a software application to detect and respond to changes in the environment [21]. Context-aware systems are then systems capable to "*adapt their operations to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account*" [7]. The notion of context is then central to these definitions. Context can be defined as *any information that can be used to characterize the situation of an entity (a person, place, or object) considered as relevant to the interaction between a user and an application* [14]. It corresponds to a large concept standing for evolving, structured and shared information spaces, which are designed to serve a particular purpose [13]. This purpose is adaptation, in the case of context-aware systems. Several aspects of system behavior can be affected. A system may adapt, according to current context, the supplied content, the services offered to the users, the components deployment and so on. Indeed, research on context-aware computing has proposed, on the last few years, multiple solutions, with different degrees of granularity and complexity [7][14][21][27][28][34]. Choosing most appropriate solution for a given situation depends on the project needs and expectations, transforming requirements analysis on an important step towards an appropriate solution. Such requirements must be clearly identified and considered before proposing any context-awareness support.

The main purpose of this deliverable is then to study concepts behind the notions of context, context-awareness and pervasive grids in order to establish clear requirements for supporting context-awareness on PER-MARE project. These requirements will guide subsequent developments of WP3 regarding context-awareness on pervasive grids.

This document is organized as follows: section 2 introduces the general context of the project, resuming the project goals and the WP3 goals concerning context-awareness. The section 3 presents the notion of context and analyses pervasive grids characteristics, while section 4 introduces main aspects of MapReduce and Hadoop distribution. Finally, section 5 proposes a set of requirements that have been identified based on the previous analyses, before concluding in section 6.

2 Project and WP3 goals

The PER-MARE project [26] aims at the adaptation of MapReduce and its most popular distribution, Hadoop, for pervasive grids. The main goal is to propose scalable techniques to support existent MapReduce-based, data-intensive applications in the context of loosely coupled networks such as pervasive and desktop grids.

In order to reach this goal, PER-MARE project proposes a two-fold approach [26]:

(i) to adapt a well-known MapReduce implementation, Hadoop, including on it context-aware elements that may allow its efficient deployment over a pervasive or desktop grid;
(ii) to implement a Hadoop-compatible API over a P2P distributed computing environment originally meant for pervasive grids.

This double approach intends to bring us better insights on the deployment of MapReduce over pervasive grids. The first one intends handling problems related to context-awareness and task scheduling, while the second one will deal with data storage and code-compatibility issues. The ambition behind this double approach is to provide a wider vision on the problem, leading to more efficient solutions, by conjointly evaluating these approaches.

By focusing on handling such dynamic and loosely coupled environments, PER-MARE projects tackles a well-known limitation of Hadoop distribution. Hadoop is not designed to fit such volatile and heterogeneous environments, lacking of adaptation. Context-aware approach proposed by PER-MARE project intends to address this problem.

The Work Package 3 (WP3) is responsible for context adaptation on PER-MARE project. It is in charge of context adaptation and its integration within PER-MARE proposals. This work package focuses on the acquisition, analysis and distribution of context information over a pervasive grid. It has to carefully consider the nature and the pertinence of context information, as well as its impact on MapReduce applications, in order to propose appropriate solutions for supporting context-awareness on the PER-MARE ecosystem.

Context-awareness on a MapReduce application can be used to address different issues. First, context acquisition mechanism can be used in order to *automatically configure nodes* during Hadoop installation. Indeed, configuration of a Hadoop node is statically defined by XML files, which must be manually edited by the developers. It is a laborious and error-prone task that can be performed automatically by using acquired context information from the node.

Second issue concerns the nodes volatility. Context information can be used in order to track *nodes availability*, allowing MapReduce application to handle nodes volatility on pervasive grids. Even if Hadoop is able to handle nodes failure in some situations, it cannot handle the arrival of new nodes without stopping data processing and reconfiguring the cluster at hand [37]. No dynamic reconfiguration is possible in current Hadoop distribution.

A third issue that should be handled is the context-awareness, properly speaking, on tasks distribution. In fact, context information can be used for *task scheduling according to nodes availability and current execution context*. The main idea consists in distribute tasks according to

nodes current context (availability, network latency, available memory and storage, etc.) in order to take fully advantage of all available resources.

Since task scheduling and data placement are strictly coupled on Hadoop distribution, adapting task schedule should also affect (and be affected by) data placement. Co-localization in a single node of data and tasks, and notably reduce tasks, is one of the main characteristics of Hadoop and one of its success reasons. Co-locating data and tasks prevents Hadoop of heavily consuming network bandwidth with the transfer of large amount of data. When considering adapting task scheduling, we should also consider adapting data placement to the current context. One cannot fairly do one without the other.

In order to tackle all these issues, we need first to understand pervasive grid environments and its characteristics, as well as the notions of context and context-awareness when considered on these environments. Such understanding forms a common ground defining the requirements that should be observed in order to reach project and WP3 goals.

3 About context-awareness and pervasive grids

The term pervasive computing refers to a seamless integration of devices into the users' everyday life [7]. Ubiquitous or Pervasive Computing represents an emerging trend towards environments composed by numerous computing devices that are frequently mobile or embedded and that are connected to a network infrastructure composed of a wired core and wireless edges [22]. When considering pervasive environments, one should consider heterogeneous environments composed of fixed and mobile devices interconnected by a mix of standard infrastructures (fixed networks) and wireless networks [31].

Pervasive environments open new perspectives for distributed computing. Ausiello [6] illustrates this idea by suggesting future pervasive systems performing tasks in a distributed and autonomous way. The authors consider a scenario with hundreds of cooperating processing units (potentially tiny ones, with small local memory) that collect, fusion and analyze data coming from various sensors, in order to perform, in an autonomous and adaptive way, the planned tasks. *Pervasive grids* are about this, about using resources embedded in pervasive environments in order to perform computing tasks in a distributed way. Concretely, pervasive grids can be seen as computing grids formed in such dynamic environment, counting on heterogeneous resources that are potentially mobiles, coming in and out the grid dynamically.

According to Parshar and Pierson [24], 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. 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, 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 [12]. Devices from different natures, from desktop and laptop PCs until last generation tablets, should be integrated in seamlessly way. As a consequence, pervasive grids are inherently large, heterogeneous and dynamic, globally aggregating large number of independent computing and communication resources [24]. Such dynamic environment can then be characterized by:

(i) the volatility of its resources, whose participation on the grid is notably a matter of opportunity and availability;
(ii) the heterogeneity of these resources, whose capabilities may vary on different aspects (platform, OS, memory and storage capacity, network connection, etc.);
(iii) the variation on the internal status of these resources, which may also vary during the execution.

Indeed, pervasive grid effort focuses mainly on the closer integration of mobile devices and on the opportunistic participation of surrounding devices, as well as on context-awareness tackling mobility [24]. As a consequence, in pervasive grids, we cannot anticipate the apparition of new nodes, neither the evolution of nodes capabilities. For instance, during the execution of a job, a new mobile device

can be integrated to a pervasive grid. After integrating the grid, it may then change its network connection, passing from a wireless connection to a fixed one. Its available memory may vary too: after starting a job, device's owner may start new applications that modify device memory status, affecting tasks performance. Pervasive grids have then to adapt to this changing environment to take the maximum advantage of available resources in an opportunistic way.

Pervasive grids represent a challenging deployment environment because of its dynamic nature. They have to deal with additional constraints mainly related to the heterogeneity and volatility of the resources. They also have to deal with fault tolerance, since nodes may fail or just leave the grid environment during processing. All these factors may strongly affect the performance of tasks running on the top of pervasive grids.

Consequently, it becomes essential to adapt the applications to the grid variable behavior and to coordinate available resources (task scheduling, data placement, etc.). According to Coronato & De Pietro [12], pervasive grid environments should be able to self-adapt and self-configure in order to incoming mobile devices. Indeed, a pervasive system should be able to perform a desired function despite changes on the environmental conditions or the system state [30]. According to [17], the dynamic and ad-hoc nature of pervasive environments means that the environment has to adapt to changing operating conditions and to user changing preferences and behaviors in order to enable more efficient and effective operation while avoiding system failure.

This idea of adapting its behavior to a changing environment corresponds to the notion of context-awareness. *Context-awareness* can be defined as the ability of a software application to detect and respond to changes in the environment [21]. *Context-aware systems* can be seen as systems particularly designed to react to context changes and to adapt in consequence their behavior [23]. These systems are able to observe user and execution environment and to react to changes in such environment, adapting their behavior to these changes in order to better satisfy their goals. According to Najar et al. [23], they can be seem as systems supporting some variability, in which the selection of a given variant depends on the observed context. Multiple variation points are possible: a system can adapt content supplied to its users or the services it offers by selecting, discovering or composing available services or content; it may adapt its own composition, choosing and deploying components according available resources or opportunities offered by the environment, and so on. Indeed, several adaptation possibilities are referenced in the literature. Maamar *et al.* [21] mention, for instance, the opportunities for service composition according to execution context. Taylor et al. [34] also propose to personalize service discovery according to the user context. Preuveneers & Berbers [27] propose an opportunistic deployment of OSGi bundles according to available resources. Vanrompay et al. [35] suggests forming groups for information distribution according to nodes context. Whatever variability is considered, it reposes on "*a decision, an action that is taken under a given context*"[23]. Figure 1 illustrates this idea in a schematic way.

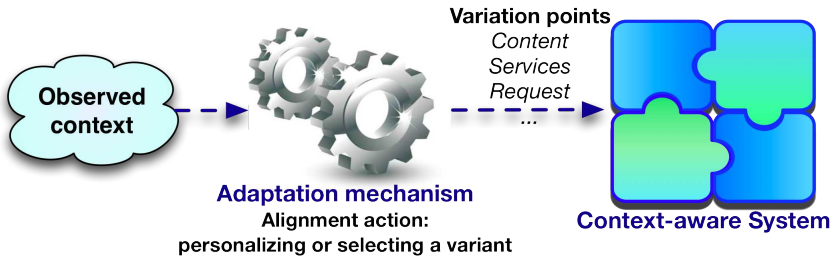

Figure 1. Schematic view of a context-aware system (based on [23]).

Thus, context information acts as a *parameter that leads to the selection of the most appropriate variant and to the adaptation process inherent to context-aware systems* [23]. Context can be seen as a set of parameters which are *extern to the application* and that influence the behavior of the application [11]. It can be defined as *any information that can be used to characterize the situation of an entity (a person, place, or object) considered as relevant to the interaction between a user and an application* [14]. It is worth noting that context corresponds to a large concept, encompassing almost any information that could be useful to identify the environment in which an application executes and that could be used for adaptation purposes.

We believe that pervasive grids should behave as context-aware systems. They must adapt its own behavior to dynamic changes on its environment in order to improve its overall performance by opportunistically using available resources. Context in such environment can be defined by the observation of several elements such as available nodes, their location and current status (available memory, processor type, storage capability, network connectivity, latency, etc.). Variation on context information (for instance, the arrival of a new node or the reduction of available memory on an existent one or a low level of battery) can launch adaptation process (for example, the redeployment of a task). In order to be successful, pervasive grids have to propose appropriate core services allowing such a context-aware adaptation of running applications. Such services have to cope with available resources in order to improve applications performance in a transparent way. Traditional core services offered by grid platforms, such as resource discovery, resource management and job management services [24] should then be improved with context-awareness capabilities.

The possibilities for context-aware adaptation on pervasive grids being numerous, it is necessary to establish clear requirements that should be observed by such adaptation mechanism. Nevertheless, before stating such requirements for our project, we must analyze MapReduce and Hadoop, since our goal (see section 2) is to improve Hadoop distribution and its MapReduce applications with a context-aware behavior.

4 About MapReduce and Hadoop

The main idea behind MapReduce is allowing the treatment of large amount of data in a distributed way. As an illustration, let's us consider a catalogue of postal address and phone number for a city like Paris or Montevideo. If such catalogue is not organized by neighborhood or geographic zone, how could we know the average of phone numbers by address? Or how many phone numbers at maximum we have in a building? In order to answer these questions, we must compute all data, but doing it sequentially will be very time consuming. A better solution, illustrated by Figure 2, will be to split data into smaller parts and spread out the analysis of such parts over several participants (nodes). Each participant may then analyze a smaller volume of data, and when all participants finish analyzing their own piece of data, we will be able to combine all intermediary results and to compute final result. This example quickly illustrates the working principle of MapReduce: an initial processing of data partitioned into smaller pieces of data (the map phase) followed by a final processing, combining first phase results (the reduce phase).

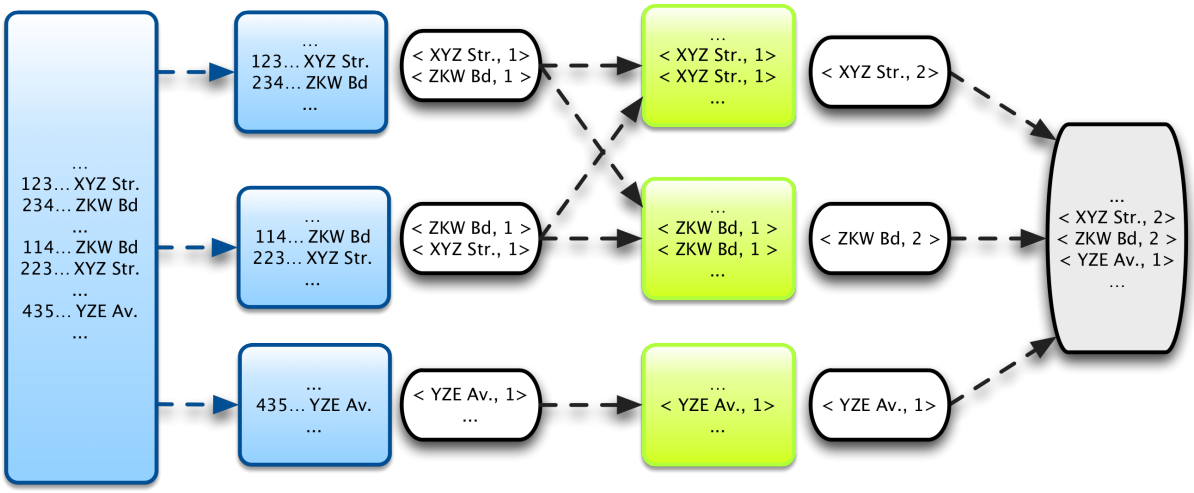

Figure 2. Schema illustrating MapReduce programming model.

MapReduce is then a programming model essentially for data processing. It works by breaking the processing into two phases, the *map phase* and the *reduce phase*, each one processing a set of key-value pairs as input and output [38]. MapReduce separates input data (for example, phone number and address in Figure 2) on a set of *input* $<key_{in}, value_{in}>$ pairs, which is divided and distributed among the participants for the *map phase*. During this phase, participants apply a *map function* for processing data, generating a set of intermediary $< key_{inter}, value_{inter} >$ pairs. The output of map phase is then sorted and transferred to the *reduce phase*, in which participants will integrate and combine these results, producing a set of $< key_{out}, value_{out} >$ as output (*e.g.*, an address and its occurrence number in Figure 2).

MapReduce model fits essentially data-oriented processes. Nevertheless, it is worth noting that, even if processing data is the center of MapReduce implementations, we may also consider using MapReduce for intensive computing (for instance, for computing the product of large matrices, or for dividing and exploring a large search space using a meta-heuristic).

Different implementations of MapReduce are available, Hadoop[1] being the most popular one. When submitting a MapReduce problem to Hadoop, a programmer submits a *job*. A Hadoop job is the unit of work a client wants to be performed, including tasks from two types: map tasks and reduce tasks [38]. Hadoop divides the job in these phases, organizes input data necessary for each phase and manages the distribution of both tasks and data. Programmer has only to program mapper and reducer functions. This behavior, like a "black box", is one of the stronger points of Hadoop: a programmer has no need to code or to manage data exchange or process communication among nodes. Hadoop is in charge of all these "management" aspects, and notably data distribution among tasks.

Data has an important influence on Hadoop behavior. For instance, during map phase, division and distribution of tasks seem to be guided by volume of input data. Such data is split into smaller blocs, which are distributed over declared nodes. According to White [38], Hadoop divides the input into fixed size pieces (about 64Mb), called *splits,* which are used to create map tasks: Hadoop creates one map task for each split, which runs the map function for each record on the split. Programmer may suggest split size and number of map tasks to Hadoop Job configuration, but he may not really impose such parameters to the framework. Finally, it is worth noting that data distribution is performed mainly through HDFS system, which is a distributed filesystem particularly designed for Hadoop. Indeed, tasks are planned in order to minimize data transfers, processing data that is co-localized in the same node. Hadoop tries to collocate data with the compute node, speeding up data access as it is locally stored [38]. The influence of data distribution, and consequently of HDFS, over Hadoop can also be observed during reduce phase. Hadoop tries to concentrate data related to a given key in a single node and to allocate the same node for reducing of such data. This way, reduce task distribution can be seen as a consequence of data distribution on Hadoop.

Data distribution can also influence the number of tasks that will be launched by Hadoop during a job. Ideally, the number of map tasks is determined by the total number of splits, *i.e.* by amount of available input data, while the number of reduce tasks is determined by the number of available keys (i.e. reduce input data). Nevertheless, in the practice, the number of tasks is often delimited by the number of available nodes.

It is important to observe that there can be many keys in each partition generated by map tasks [38], and that values related to a single key may be split over different nodes. Thus, data resulting from map tasks should be organized for the reduce tasks. This intermediary phase, between map and reduce phases, is called "shuffle and sort". During the shuffle, each reduce task can be fed by many map tasks [38], as illustrated by Figure 3. Also during the shuffle, data can be reorganized, being combined (if there is too many keys available) or partitioned (if there is to many data for each key). These are the roles of the "combiner" and "partitioner" functions. The combiner tries to combine different map results, aggregating results from a key, while the partitioner splits map results by assigning multiple intermediate keys. Programmers can propose such functions for tunneling his data, but the decision concerning the execution of such functions during a job is made by Hadoop, based on the results from the map phase.

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

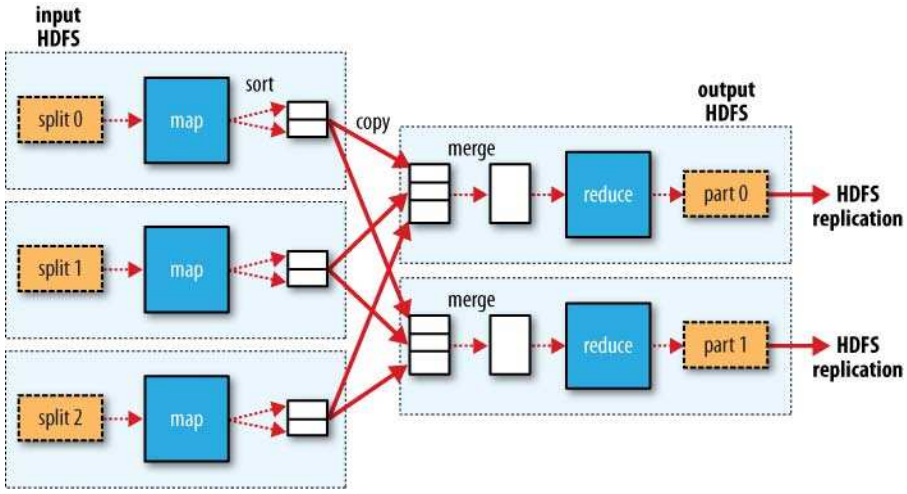

Figure 3. Data flow between map and reduce phases [38].

The Hadoop framework manages all job execution, launching map, reduce and shuffle phases when needed. It is also in charge of distributing data (mainly through HDFS) and tasks among declared nodes. Indeed, when executing on a cluster mode, Hadoop needs a preliminary configuration where participant nodes are declared. The inclusion of new nodes during execution is not allowed: the cluster must be stopped and configuration files updated in order to take into account new nodes.

Hadoop schedulers can use the information about the nodes composing the cluster in order to distribute tasks during execution. Indeed, Hadoop offers the possibility to configure different task schedulers. By default, Hadoop uses FIFO queue-based scheduler [38], but Fair Scheduler[2], Capacity Scheduler[3] and HOD Scheduler[4] are also available. New schedulers can also be added by extending *TaskScheduler* (Hadoop 1.x) class and configuring Hadoop to use a new class [33]. The same seems to be possible in Hadoop 2.x, in which a class named *YarnScheduler* is responsible for allocating resources for all the running applications [36].

From an architectural point of view, Hadoop adopts a master/slave approach, where some nodes (*JobTracker* and *NameNode* for Hadoop 1.x, *ResourceManager* and *NodeManager* for Hadoop 2.x) manage data and task distribution over the different slave nodes. *NameNode* considers data distribution (*i.e.* in which node data will be stored), while *JobTracker* considers the tasks distribution (what node will execute a task).

Thus, two types of nodes control job execution process [38]: the *JobTracker* and the *TaskTracker*. The first coordinates all jobs runs by scheduling tasks to run on the task trackers and by collecting overall progress information for each job. The former runs tasks and send progress reports to the *JobTracker* [38]. The *JobTracker* acts then as a master node, while *TaskTracker* represents the "slave workers" that actually execute the tasks. For data management, the structure is the same: we have a master, represented by the *NameNode*, and multiple "slave workers" represented by the *DataNodes*. According to White [38], a *NameNode* manages the filesystem namespace: it maintains filesystem information, notably metadata and the location of the data blocks over the *DataNodes*. These latters

[2] http://hadoop.apache.org/docs/stable/fair_scheduler.html
[3] http://hadoop.apache.org/docs/stable/capacity_scheduler.html
[4] http://hadoop.apache.org/docs/stable/hod_scheduler.html

keep the data blocks (no data block is stored on the *NameNode*) and are in charge of storing and retrieving blocks when needed, reporting periodically back to the *NameNode* with a list of locally stored blocks.

Both *NameNode* and *JobTracker* represent single points of failure in Haddop 1.x architecture. Failures on these nodes, and notably on *JobTracker*, are fatal to Hadoop processing (the execution of a Hadoop job cannot survive to failures on these nodes). This problem has been demonstrated by [15][16], in which authors used failure injection techniques in order to demonstrate Hadoop vulnerabilities concerning master nodes failures.

Concerning Hadoop 2.x, its architecture adopts the same master/slave, with *ResourceManager* and *NodeManager* replacing the *JobTracker* and *NameNode*. Nevertheless, internal architecture of Hadoop 2.x seems to be different from its predecessor. Indeed, Hadoop 2.x adopts a different architecture, based on YARN[5]. YARN is resource manager, independent from Hadoop MapReduce applications (running applications that are not necessarily MapReduce ones), that provides a set of daemons and API for scheduling resource requests and for supervising application execution [20]. YARN proposes to split up into separate daemons two major functionalities of the *JobTracker*, which are the resource management and the job scheduling/monitoring. The idea is to have a global *ResourceManager* (RM) and per-application *ApplicationMaster* (AM) [1]. By separating these elements, Hadoop 2.x allows running multiple applications against relevant data sets. Such multiple applications can then operate efficiently and in a predictable way within the same cluster [20].

All these changings severely impact Hadoop 2.x architecture, which becomes much more complex than its predecessor, with several dependencies (such as ProtocolBuffer dependency [10]). Besides, Hadoop 2.x is still evolving, being still in its alpha version at the moment this document was written. Analysis of Jira project site illustrates an active project [10], but the question such activity rises is the code stability: is this code (and subjacent architecture) sufficiently stable for allowing extensions such as those proposed by PER-MARE project? At this moment, we have no definitive answer for this important question, and both architectures (Hadoop 1.x and Hadoop 2.x) should be analyzed and considered for the moment.

[5] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

5 Requirements for context-aware MapReduce

Based on the analysis of pervasive grids (section 3) and Hadoop (section 4) characteristics, we could determine a set of important requirements that should be observed by PER-MARE project in order to supply a context-aware support for MapReduce and Hadoop applications. These requirements were organized into three main sections: general requirements (section 5.1), context modeling requirements (section 5.2) and distribution requirements (section 5.3).

5.1 General requirements

According to Coutaz et al. [13], the utility and the usability of a system are derived from the emergence of information and cooperation rather than the sophistication of its individual components. Pervasive grids share this point of view, offering to applications an environment that emerges from available resources instead of a single sophisticate cluster structure. Such emerging environment can be composed of several different kinds of resources, according opportunities and availability of these resources. Indeed, as pointed out in section 3, pervasive grids are characterized by the heterogeneity of the participating nodes. Such heterogeneity distinguishes pervasive grids from traditional cluster, which can be characterized by a general homogeneity of their resources.

The heterogeneity of pervasive grids imposes some challenges to MapReduce applications. First of all, applications must consider such heterogeneity when scheduling their tasks. This means to be able to collect information about nodes configuration and current status, and to able to reason about this information. In other words, context information about the nodes themselves should be observed, and this without a significant impact on the nodes performance [32]. This imposes us the following requirements for context-awareness support on PER-MARE:

- **[R1]** Applications and frameworks, such as Hadoop, running over pervasive grids should be able to observe context information of grid resources. *Context monitoring is then needed*.
- **[R2]** *Context monitoring components should be platform independent*. Considering the heterogeneity of pervasive grids, any mechanism for observing context on the participating nodes should be made available for a large variety of platforms, covering a large range of hardware and OS platforms. Platform-dependent solutions should then be avoided. Priority should be given to platform independent ones.
- **[R3]** *Context information about pervasive grid resources should be made available for scheduling needs*. Scheduling is an important service for grid platforms, being in charge of the task distribution according to available resources. These resources being heterogeneous and changing, it is important for a better scheduling to consider current context information when deciding task distributions.
- **[R4]** Applications and frameworks running over pervasive grid should be able to easily obtain context information from participating nodes when needed. *Context management is then a necessary service for pervasive grids*.

From an architectural point of view, this means that a new component, offering context management services, should be made available on MapReduce frameworks like Hadoop. However, context monitoring and management should not impact the overall performance of the running

applications, since final goal of using grid environments for MapReducing is precisely to improve execution performance. A new requirement can then be announced:

- **[R5]** *Context management on MapReduce frameworks should be lightweight* since those frameworks focus mainly on application performance. Performance of MapReduce applications should not be negatively impacted by context monitoring services.

Besides, pervasive grids are also characterized by their nodes volatility. Indeed, 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 [32]. From an architectural point of view, this means a flexible and fully distributed architecture in order to support nodes discover and nodes fail. Indeed, not only new nodes can join the grid, but also current nodes may leave the grid or fail. A more fault tolerant architecture is then needed. As discussed on section 4, such fault tolerant behavior is not necessarily present on Hadoop, traditionally based on a master/slave architecture, in which fails on master are difficult to overcome and in which new nodes cannot be added without stopping cluster execution. Indeed, current Hadoop implementation (as well as most Hadoop-related solutions [2][3][5]) is designed considering traditional clusters, in which available nodes are predefined during cluster definition. Moreover, as discussed in section 3, such dynamicity on pervasive environments is one of the reasons motivating a context-aware behavior on pervasive grids, and on PER-MARE project. Context-aware support on MapReduce should then be able to allow observing nodes disconnection and joining, which lead to the following requirements:

- **[R6]** New nodes can integrate the pervasive grid at any moment. *Context management service should allow this behavior and start monitoring new discovered nodes when available*.
- **[R7]** Inversely, *the fail or vanishing of one node must not lead stopping context monitoring and management services on the other nodes*. Consequently, a purely centralized architecture for context management should then be avoided.

Finally, it is worth noting that heterogeneity of pervasive grids makes harder to configure nodes for Hadoop execution. As stated before, Hadoop was designed for traditional clusters, which have a set of predefined homogeneous nodes. And even on such stable environment, Hadoop configuration can seen as a complex task, since an important set of files and properties must be defined according the nodes capabilities and their roles on the cluster (a configuration of a master node is roughly different from that of a slave node). Complexity of configuring and managing Hadoop cluster motivates the proposal of several tools for managing such clusters, such as Apache Zookeper [5], Ambari [2], Chukwa [3] or Mesos [4]. Each of these tools proposes solutions at different levels for managing a Hadoop cluster. However, most of them use platform-specific monitoring tools, needing managing different versions according available platforms, and they are often quite complex (as complex as configuring Hadoop). An illustration of this is Ambari, whose configuration and install are particularly awkward, with multiple dependencies, and whose compatibility is only assured for a small set of Linux based platforms. This kind of tool fits quite well traditional cluster environments, but it is difficult to be deployed on heterogeneous pervasive grids. Configuration of pervasive grids remains an open problem that can demotivate the adoption of these environments for MapReduce applications. From these observations, a new requirement can be dressed:

- **[R8]** *An automatic configuration of Hadoop nodes is necessary on pervasive grids*. Such automatic configuration can partially lie on acquired context information (for example, nodes available memory and storage capacity).

5.2 Context modeling requirements

In order to provide a context-aware behavior for MapReduce applications, different context information can be observed: node location, available memory, available storage, network latency, etc. [32] This information can be then used for adaptation purposes (node configuration or task scheduling, for instance). The way context information is used depends not only on what information is observed, but also on how it is represented: the adaptation capabilities of a context-aware applications depend on the context model used on it [23]. Representing context information in an appropriate model is then a need.

According to Bettini et al. [9], context information models have to deal with a large variety of context information sources that differ in their update rate and their semantic level. For these authors, adopting a good context information modeling formalism reduces the complexity of context-aware applications and improves their maintainability and evolvability. Indeed, formalizing context information on an appropriate model allows a clear separation between adaptation mechanisms and context acquisition technologies. Adaptation mechanisms may then take profit of context information represented through a given context model, without any knowledge on how such information is acquired or maintained. Thanks to an appropriate context modeling, it is also possible to evolve context acquisition methods without affecting adaptation mechanisms. Indeed, the former may evolve independently of the latter, thanks to the evolution on the available technologies or techniques.

In the last decade, several context models have been proposed in the literature, varying from simple key-values models until complex ontologies [9][23]. New hybrid models, combining different formalisms, have also appeared [9]. For instance, Reichle et al. [29] propose a multi-layered model, in which context information is represented using different formalisms at different levels (ontology for defining concepts, object-oriented for coding and XML for data transmission). The main goal of such hybrid models is to overcome performance limitations of ontologies, without ignoring reasoning advantages offered by those. Concerning context acquisition, several methods and tools are available [14][25]. Works such as [25] propose extensible mechanisms, based on pluggable components. For instance, Paspallis *et al.* [25] propose an extensible architecture in which context plugins can be dynamically loaded according to application needs.

Based on those observations, two new requirements can be pointed out:

- **[R9]** *Context information should be represented in an appropriate model, respecting requirement R5*.
- **[R10]** Context model should be extensible, allowing the introduction of new observed elements.

Different adaptation mechanisms can then be proposed based on a given context model. Such context model determines reasoning possibilities necessary for adaptation mechanisms. In the case

of MapReduce application, such adaptation mechanisms may consider the data distribution, based on schedulers that consider current context on available nodes for placing or moving a task from or to a node. Nevertheless, as indicated in section 4, task distribution is heavily influenced by data distribution, the respect of the data locality property being an important aspect for MapReduce applications. In other words, adapting task distribution means necessarily to adapt data allocation too. Both cannot be considered in a totally independent way.

- **[R11]** *Context model formalisms should allow the proposition of adaptation mechanisms for both task distribution (scheduler) and data placement*.

5.3 Distribution requirements

Context-aware applications collect and react to context information, exploiting it through predefined adaptation mechanisms. The success of such mechanisms depends on the availability of such information, which is disseminated over the network [18]. On pervasive grids, this means to distribute context information about the available nodes over the grid. In other words, 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 [32]. This gives us the following requirement:

- **[R12]** *Context information about available nodes should be distributed over the grid*, allowing adaptation mechanisms, and notably schedulers, to consider this information during their tasks.

A context distribution mechanism is then needed. Context distribution can be defined as the function in charge of distributing relevant context data to interested entities [8]. Different solutions have been proposed in the literature, using various architecture styles [8][18]. Centralized solutions, in which central server concentrates and manages context information collected from the environment, remain numerous [7]. These centralized approaches are prone to scalability and single point of failure problems. They are then incompatible with requirement R7, stated in section 5.1. On the opposite, distributed solutions, in which nodes share their context information with all other nodes on the network, also suffer from scalability problems, being exposed to a risk of message flooding. This is an important concern for pervasive grids and MapReduce applications. On the one hand, when considering pervasive grids, it is not possible to guarantee high quality links, as on traditional cluster. On the other hand, MapReduce paradigm is often used for data intensive applications. Processing large amount of data in a distributed way may easily lead to an important consumption of bandwidth. As a result, when considering both, MapReduce applications over pervasive grids, bandwidth becomes a major key resource that should be managed appropriately, since there is a significant risk of link saturation.

- **[R13]** As indicated in requirement R7, *distributed approaches should be preferred to totally centralized solutions for context distribution in order to better support nodes volatility and fault tolerance.*
- **[R14]** *Context distribution mechanisms for MapReduce applications should cope with R5 requirement (being lightweight), and notably avoiding communication flooding.* Network bandwidth is an important resource on pervasive grids. It should not be over-consumed by context distribution mechanisms.

Such requirements may appear antagonistic, since the easiest way to prevent excessive communication is to centralize all messages into a master node. However, such master node represents a single point of failure. It is then important to find out hybrid approaches, allowing us to minimize communication over the network without preventing nodes from receiving context information. In the literature, some hybrid approaches exists, such as [18]. In this work, several "super-nodes" concentrate context information from simple peer nodes (corresponding to nodes with less capacity) and form communities (groups) according their context information needs. Only authorized context information is exchanged among community nodes. Other possible hint could be to extend the heartbeat mechanism traditionally used by Hadoop to identify nodes failures for context distribution purposes [32]. In this case, context information would be "piggybacked" over exchanged messages, preventing bandwidth to be over-consumed with new context distribution messages.

6 Conclusions

In this document, we have analyzed definitions concerning context and context-awareness, as well as the characteristics of pervasive grids. We crossed this analysis with the study of Hadoop and MapReduce bases in order to extract a set of requirements that should be observed by PER-MARE project when considering context-awareness support.

Based on these requirements and on the multiples points highlighted by this study, we can now consider a set of alternatives and possible hints to be explored by PER-MARE partners. Among these, we can cite, for instance, the support for nodes volatility. It is now clear for us that a P2P-like behavior is needed in order to allow nodes to dynamically integrate the grid or to disconnect from it. It is also clear that handling such dynamicity will affect master nodes used for task and data distribution (JobTracker and DataNode on Hadoop 1.x, ResourceManager and NodeManager on Hadoop 2.x). A new context management service should be integrated to this platform, with a minimal impact on the overall performance. A new task scheduler can be proposed based on context information made available by such new service. Similar to [19], the idea is to adapt task scheduler to resource characteristics “like Computational or I/O strength” [19]. But contrarily to these authors, whose schedule is based on a simple classification of nodes (from “very good” to “very bad”), we intend to consider runtime information about available resources, handling not only nodes heterogeneity (such as [19]), but also nodes volatility.

Besides, it seems now clear that an extensible mechanism for context acquisition, such as [21], which allows to easily plugin new sensor capabilities to the platform, is needed. Context distribution too needs to be flexible, also adopting a P2P-like behavior for supporting nodes volatility. Different hints are now under study, such as “piggybacking” Hadoop heartbeat mechanism messages with context information about nodes. Automatic configuration of Hadoop nodes is also under study, since heterogeneity of such nodes makes complex the configuration of Hadoop platform. A specific document describing the project advances on this subject shall follow in the next months.

Finally, it is important to observe that not all requirements will be necessarily observed during PER-MARE project. Some requirements are contradictory and decisions should be made concerning their realization. The goal of this document is precisely to offer a base for discussion, rising hints and alternatives for further discussions and decisions.

7 References

1. Apache Software Foundation, "Apache Hadoop NextGen MapReduce (YARN)", http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html (last visit June 12th 2013)
2. Apache Software Foundation, "Apache Ambari", http://incubator.apache.org/ambari/ (last visit July 15th 2013)
3. Apache Software Foundation, "Release Notes - Hadoop Chukwa - Version 0.5", http://incubator.apache.org/chukwa/docs/r0.5.0/releasenotes.html (last visit July 15th 2013)
4. Apache Software Foundation, "Apache Mesos: Dynamic Resource Sharing for Clusters", http://mesos.apache.org/ (last visit July 15th 2013)
5. Apache Software Foundation, "Apache Zookeper", http://zookeeper.apache.org/ (last visit July 15th 2013)
6. Ausiello, G., "Cooperative data stream analysis and processing", In: A. Ferscha (Ed.), *Pervasive Adaptation: Next generation pervasive computing research agenda*, 2011, 15
7. Baldauf, M.; Dustdar, S. & Rosenberg, F., "A survey on context-aware systems", International Journal of Ad Hoc and Ubiquitous Computing, vol. 2, n° 4, 2007, 263-277
8. Bellavista, P.; Corradi, A.; Fanelli, M. & Foschini, L., "A survey of context data distribution for mobile ubiquitous systems", ACM Computing Survey, vol. 45, issue 1, March 2013, 1-49
9. Bettini, C.; Brdiczka, O.; Henricksen, K.; Indulska, J.; Nicklas, D.; Ranganathan, A. & Riboni, D., "A survey of context modelling and reasoning techniques", Pervasive and Mobile Computing, Elsevier Science Publishers, vol 6, issue 2, 2010, 161-180
10. Cassales, G.W. & Charão, A.S., "Hadoop, building YARN", *[PER-MARE internal document]* 2013
11. Chaari, T.; Laforest, F. & Celentano, A., "Adaptation in context-aware pervasive information systems: the SECAS project", *Journal of Pervasive Computing and Communications,* vol. 3, n° 4, 2007, 400-425
12. Coronato, A. & Pietro, G. D., "MiPeG: A middleware infrastructure for pervasive grids", Future Generation Computer Systems, vol. 24, n°1, 2008, 17 - 29
13. Coutaz, J.; Crowley, J.; Dobson, S. & Garlan, D., "Context is the key", Communication of the ACM, ACM Press, vol. 48, issue 3, 2005, 49-53
14. Dey, A., "Understanding and using context", Personal and Ubiquitous Computing, vol. 5, issue 1, 2001, 4-7
15. Gondim, E., Prates, B., Barcelos, P. P. & Charão, A., "Análise de alternativas para injeção de falhas no Apache Hadoop", Proceedings of the XII Simpósio em Sistemas Computacionais, Vitória, ES, Brazil, 2011
16. Gondim, E., Barcelos, P. P. & Charão, A.S., "Explorando o framework de injeção de falhas do Apache Hadoop", Proceedings of the XIII Simpósio em Sistemas Computacionais, Petrópolis, RJ, Brazil, 2012
17. Hagras, Hani. "Intelligent pervasive adaptation in shared spaces", In A. Ferscha (Ed.), *Pervasive Adaptation: Next generation pervasive computing research agenda*, 2011, 16-17
18. Kirsch-Pinheiro, M.; Vanrompay, Y.; Victor, K.; Berbers, Y.; Valla, M.; Frà, C.; Mamelli, A.; Barone, P.; Hu, X.; Devlic, A. & Panagiotou, G., "Context Grouping Mechanism for Context Distribution in Ubiquitous Environments", In: Meersman, R. & Tari, Z. (Eds.), On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International

Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Lecture Notes in Computer Science, vol. 5331, 2008, 571-588

19. Kumar, K. A.; Konishetty, V. K.; Voruganti, K. & Rao, G. V. P., "CASH: context aware scheduler for Hadoop", Proceedings of the International Conference on Advances in Computing, Communications and Informatics, ACM, 2012, 52-61
20. Lopez, Issac. "YARN to Spin Hadoop into Big Data Operating System", Datanami, May 28th 2013, http://www.datanami.com/datanami/2013-05-28/ yarn_to_spin_hadoop_into_a_big_ data_operating_system_.html (last visit: June 12th 2013)
21. Maamar, Z.; Benslimane, D. & Narendra, N. C., "What can context do for web services?", Communication of the ACM, vol. 49, issue 12, 2006, 98-103
22. Moran, T. & Dourish, P. "Introduction to this special issue on context-aware computing", Human-Computer Interaction, vol 16, 2-3, 2001, 87-95
23. Najar, S.; Saidani, O.; Kirsch-Pinheiro, M.; Souveyet, C. & Nurcan, S., "Semantic representation of context models: a framework for analyzing and understanding", Gomez-Perez, J. M.; Haase, P.; Tilly, M. & Warren, P. (Eds.), Proceedings of the 1st Workshop on Context, information and ontologies (CIAO 09), European Semantic Web Conference (ESWC'2009), ACM, 2009, 1-10
24. Parashar, M. & Pierson, J.-M., "Pervasive Grids: Challenges and Opportunities", In: Li, K.-C.; Hsu, C.-H.; Yang, L. T.; Dongarra, J. & Zima, H. (Eds.), Handbook of Research on Scalable Computing Technologies, IGI Global, 2010, 14-30
25. Paspallis, N., Rouvoy, R., Barone, P., Papadopoulos, G.A., Eliassen, F. & Mamelli, A., "A Pluggable and Reconfigurable Architecture for a Context-Aware Enabling Middleware System", In: Meersman, R. & Tari, Z. (Eds.), On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Lecture Notes in Computer Science, vol. 5331, 2008, 553-570
26. PER-MARE: Adaptive Deployment of MapReduce-based Applications over Pervasive and Desktop Grid Infrastructures, Project Proposal, Regional Program STIC-AmSud 2012 [*PER-MARE internal document*]
27. Preuveneers, D. & Berbers, Y., "Context-driven migration and diffusion of pervasive services on the OSGi framework", Int. J. Auton. Adapt. Commun. Syst., Inderscience Publishers, vol. 3, issue 1, 2010, 3-22
28. Preuveneers, D.; Victor, K.; Vanrompay, Y.; Rigole, P.; Kirsch-Pinheiro, M. & Berbers, Y., "Context-Aware Adaptation in an Ecology of Applications", In: Stojanovic, D. (Ed.) Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications, IGI Global, 2009, 1-25
29. Reichle, R.; Wagner, M.; Khan, M.; Geihs, K.; Lorenzo, J.; Valla, M.; Fra, C.; Paspallis, N. & Papadopoulos, G., "A Comprehensive Context Modeling Framework for Pervasive Computing Systems", In: Meier, R. & Terzis, S. (Eds.), Distributed Applications and Interoperable Systems, Lecture Notes in Computer Science, vol. 5053, Springer Berlin Heidelberg, 2008, 281-295
30. Römer, K. & Friedemann M., "Adaptation without antecipation", In A. Ferscha(Ed.), *Pervasive Adaptation: Next generation pervasive computing research agenda*, 2011, 28-29
31. Steffenel, L. A. & Kirsch-Pinheiro, M. "Strong Consistency for Shared Objects in Pervasive Grid", 5th IEEE International Conference on Wireless and Mobile Computing, Networking and Communication (WiMob'2009), Marrakesh, Morroco, 12-14 October 2009, 73-78

32. Steffenel, L. A., Flauzac, O., Charão, A.S., Pitthan Barcelos, P., Stein, B., Nesmachnow, S., Kirsch-Pinheiro, M. & Diaz, D., "PER-MARE: Adaptive Deployment of MapReduce over Pervasive Grids", to appear in 8th International Conference On P2p, Parallel, Grid, Cloud And Internet Computing.
33. StackOverflow.com, "How to Add a New Scheduler in Hadoop?", http://stackoverflow.com/questions/17125090/how-to-add-a-new-scheduler-in-hadoop (last visit: June 13th 2013)
34. Taylor, N.; Robertson, P.; Farshchian, B.; Doolin, K.; Roussaki, I.; Marshall, L.; Mullins, R.; Druesedow, S. & Dolinar, K., "Pervasive Computing in Daidalos", Pervasive Computing, IEEE, vol. 10, n° 1, 2011, 74 -81
35. Vanrompay, Y.; Kirsch Pinheiro, M.; Ben Mustapha, N. & Aufaure, M.-A., "Context-Based Grouping and Recommendation in MANETs", In: Kolomvatsos, K.; Anagnostopoulos, C. & Hadjiefthymiades, S. (Eds.), Intelligent Technologies and Techniques for Pervasive Computing, IGI Global, 2013, 157-178
36. Vavilapalli, V.K., "Apache Hadoop YARN – ResourceManager", Hortonworks, Aug. 31st 2012, http://hortonworks.com/blog/apache-hadoop-yarn-resourcemanager/
37. Venner, J. "Pro Hadoop", 1st edition, APress, ISBN 978-1-430-21942-2, pp. 440.
38. White, Tom, "Hadoop: The definitive guide", 2nd edition, O'Reilly, ISBN 978-1-449-38973-4, pp. 626