MapReduce in GPI-Space

Tiberiu Rotaru, Mirko Rahn, and Franz-Josef Pfreundt

Fraunhofer Institute for Industrial Mathematics,
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{tiberiu.rotaru,mirko.rahn,franz-josef.pfreundt}@itwm.fraunhofer.de

Abstract. The computing power of modern high performance systems cannot be fully exploited using traditional parallel programming models. On the other hand, the growing demand for processing big data volumes requires a better control of the workflows, an efficient storage management, as well as a fault-tolerant runtime system. Trying to offer our proper solution to these problems, we designed and developed GPI-Space, a complex but flexible software development and execution platform, in which the data coordination of an application is decoupled from the programming of the algorithms. This allows the domain user to focus on the implementation of its problem only, while the fault tolerant runtime framework automatically runs the application in parallel in complex environments. We discuss the advantages and the disadvantages of our approach by comparison with the most popular MapReduce implementation, Hadoop. The tests performed on a multicore cluster with the wordcount use case showed that GPI-Space is almost three times faster than Hadoop when strictly the execution times are considered, and more than six times faster when the data loading time is also considered.

1 Introduction

Nowadays, clusters with thousands of cores are present in many companies and institutions. At the same time, the price of the data storage has significantly decreased and the companies have now the possibility to stock or to archive big amounts of data related to their businesses, with affordable costs. In this way, they can occasionally or regularly process the backuped data and extract from it valuable information that may contribute to improve their services or to increase their profits. A fundamental question that arises in this context is how to efficiently exploit the full computing power of large multicore clusters when dealing with big amounts of data and which programming model to use for achieving this goal? The MapReduce model, although not perfect, is an answer to this question. Although the MapReduce paradigm existed for a long time in Computer Science, in one form or another, it was first evoked in the form known today by two researchers working at Google, in a paper published in 2004 [3]. Since then, a lot of research studies have been published and a number of different of MapReduce frameworks, written in different languages, have been developed [5].

D. an Mey et al. (Eds.): Euro-Par 2013 Workshops, LNCS 8374, pp. 43–52, 2014.
© Springer-Verlag Berlin Heidelberg 2014

In this paper we present GPI-Space, a complete software solution for dealing with data intensive computations on multicore clusters. On top of this infrastructure we developed a Petri net workflow for MapReduce computations, giving thus the users that are acquainted with this model the possibility to easily integrate their applications. GPI-Space offers superior performance compared to other tools by using the in-memory storage and sophisticated workflow parallelization and scheduling strategies. While most of the MapReduce implementations are targeted at commodity hardware, GPI-Space relies on the Global Address Space Programming Interface (GPI) middleware [1], which is able to take advantage of the modern network interconnects that enable the Remote Directory Memory Access (RDMA).

1.1 The MapReduce Model

MapReduce represents currently the most successful model that has been applied and has proved its efficiency at large scale [9]. This model tries to ease the task of building large scale data intensive applications, by hiding the details of parallelism, allowing the users to focus on the data processing strategies. The advantages and the limitations of the model are discussed in [7].

The MapReduce programming model is founded on concepts that are rather proper to the functional programming [6]. Generally, in order to build a MapReduce application, the user is required to provide only specific implementation for a reduced number of interface methods, with the signature known in advance. In the first place, the user is required to write the map and the reduce methods. Typically, map takes as input parameter a key-value pair and produces a list of intermediate pairs, while the reduce method takes as input parameters a key and a list of values associated with this key and returns a list of output values:

$$\text{map } (in\_key, in\_value) \longrightarrow \text{list } (out\_key, intermediate\_value)$$
$$\text{reduce } (out\_key, \text{list}(intermediate\_value)) \longrightarrow \text{list } (out\_value)$$

Additionally, the user may provide an implementation for a partitioning method, a compare method and a combine method [9].

Among all the MapReduce implementations, the most important are Google MapReduce [3] and Hadoop [14]. Hadoop is currently an important component of the infrastructure of many web companies, such as Yahoo [12], Facebook [2] and Twitter [8]. Other companies, like eBay, Amazon, IBM, Microsoft, Oracle use or offer services that are based on Hadoop.

2 Fraunhofer GPI-Space

The available computing power of distributed high performance systems, often with several thousands of cores, can hardly be mastered with the conventional programming models. On the other hand, the growing demand for processing huge amounts of data requires a better control of the workflows being executed,

an efficient storage management and a fault-tolerant execution of the workflows. Handling these aspects correctly may exceed the capabilities of a normal user, requiring expert knowledge. In order to properly handle this situation and to alleviate the domain specific programmer's mission, a natural idea would be to ensure a separation of the coordination level from the execution level of parallel applications. This requires a more sophisticated programming model and an execution framework that offers support for automatic parallelization and scheduling, a fault-tolerance mechanism and services for the job execution control, monitoring, memory management, etc. Trying to put this concept in practice, we designed and implemented a programming model and an execution framework that clearly realizes this separation. GPI-Space is a complex but flexible software platform, combining several important software components that can also be used either as standalone applications or in combination with other software tools or packages. Each of this components were designed and implemented taking into account the experience accumulated with other important high performance computing projects [11,10]. GPI-Space consists of: a distributed runtime system, a workflow engine and a storage layer.

2.1 The Distributed Run-Time System

The Distributed Run-Time System (DRTS) represents the execution layer of GPI-Space. It is dynamic, fault-tolerant and can dynamically build arbitrary topologies. It relies on a master-slave architecture, where an agent may have multiple masters, event subscribers and workers. On top of such an architecture stays the orchestrator, which is responsible with handling the user requests and scheduling them on the available agents. The agents may have access to the partitioned global address space and thus trigger large data transfers from or to the virtual memory. The agents may dispose of a workflow engine that interprets sub-workflows, creates intermediary tasks and assembles the results. A graphical representation of a DRTS, deployed as a tree topology, is as in the figure 1. In fact, the agents can form logical communication topologies with a much more complex graph structure, allowing thus a straightforward implementation of the parallel algorithms that assume a certain structure of the logical communication graph. The agents implement a Staged Event Driven Architecture [13] and are controlled by finite state machines. The stages are basically thread pools with a shared queue of events and they can asynchronously exchange messages, making thus the agent more flexible and more responsive. The whole distributed runtime system is fault-tolerant and an agent can join and leave the system at any time, without stopping the execution of the submitted workflow. The agent was designed with the bridge pattern in mind, trying to decouple as much as possible the abstract part from the implementation part [4]. An agent is composed of several software components like: a job manager, a scheduler, a worker manager and a backup service. The jobs may be submitted together with a list of requirements and the workers may have different capabilities. Each agent disposes of a scheduler that tries to fairly schedule the jobs on workers whose capabilities are best matching the requirements.

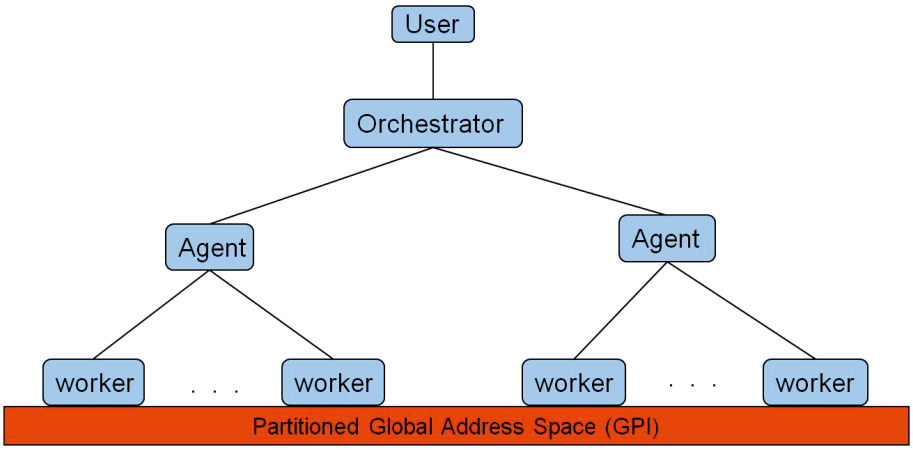

Fig. 1. Distributed run-time system deployed as a tree over a partitioned global address space

2.2 The Workflow Engine

The coordination level consists of a workflow engine, a workflow description language and a set of tools that are intended to help the user to build workflows. The workflows are basically high-level Petri nets, described in a proprietary XML-based language that we defined. We developed also a workflow engine that is capable to concurrently interpret and execute these workflows. The user may either write the workflow directly into the XML-based language or may use an editor (currently under development, but having basic functionality implemented). Apart of the workflow engine, a number of other tools were developed, with the goal to assist the user in the course of the development of a workflow:

- a Petri net compiler which translates the XML description of the workflow into some intermediary format that the workflow engine is capable to understand,
- a verification tool that is able to verify the basic properties of the net,
- a basic visualization tool, based on the Graphviz software package, and
- a graphical editor.

The compiler generates the internal representation of the runtime environment from the XML representation of the Petri net. It checks the semantic validity of the input net, being also able to check other properties like termination, absence of deadlocks, reachability, etc. and to eventually optimize the net. Typically, a user who wants to write an application for GPI-Space should focus on describing the workflow and on how to logically organize the storage layer. In the case of GPI-Space, the Petri nets have a double role: they are used not only for controlling the execution of the generated tasks, but also for controlling the access of

the tasks to the partitioned global address space, acting as a transactional mechanism and guaranteeing thus the data integrity. From a global point of view, the execution of a workflow in GPI-Space consists in the following steps:

- The orchestrator receives a job having attached the internal representation of a workflow and assigns it to one of the available agents.
- The agent hands over the attached description of a job received from a master to the workflow engine and this one extracts all the executable activities, some of them being executed locally and the others being sent to available workers.
- When the job completes, the assigned worker or agent sends the result to the corresponding master. This one hands the result over to its workflow engine, which inserts the corresponding output tokens and looks for new active transitions. If new activities were generated, they are sent to the available workers. The process continues until no transition can be fired anymore.
- When no other activities can be generated, the network is considered to be completely processed and the result of the job is sent back to the submitter.

2.3 The Storage Layer

The storage layer within GPI-Space is represented by GPI [1]. This is a middleware that allows for executing parallel applications complying with a Partitioned Global Address Space (PGAS) programming model, developed at our institute [10]. Typically, the memory of the individual nodes in a cluster is aggregated and seen as a single address space, where large objects, often exceeding the capacity of a single cluster node, can be stored. GPI is targeted at RDMA-enabled interconnects such as Infiniband or CRAY Gemini. The main idea here is that the full performance of the RDMA-enabled networks can be delivered to the application directly, without interrupting the CPUs. GPI constitutes an alternative to the traditional message-passing model for the development of parallel applications that are intended to run on modern multicore systems. GPI focuses on one-sided communication and the development of asynchronous algorithms, leveraging the capabilities of modern interconnects to overlap communication with computation.

At the node level, the Manycore Threading Package (MCTP), developed in our department with the goal to better take advantage of new multicore architectures, is used. Within GPI-Space, different applications can be combined into a workflow and executed using the virtual memory as a persistent layer that facilitates the interaction with each-other.

3 MapReduce in GPI-Space

As seen above, GPI-Space is a complex software development and execution platform that allows for writing and executing workflows that rely upon a Petri net semantics. However, writing such relatively complex workflows might be for

many users beyond the scope of domain. Therefore, we went one step forward in this direction and we developed a workflow for MapReduce computations in GPI-Space. In this way, the mission of the users that are acquainted to this model is greatly simplified and consists only in overloading a couple of methods.

There are certain similarities between Hadoop and GPI-Space:

- both are focusing on data intensive applications, being able to process large amounts of data,
- both are essentially using a master/slave architecture,
- both of them have their own execution framework,
- both of them use a proprietary storage layer,
- both are addressing important aspects like fault-tolerance, load balancing and data integrity.

However, there are also notable differences like:

- GPI-Space stores the intermediary results into the global space resulted from the aggregation of RAM parts of the cluster nodes and eventually on the local disks, while Hadoop uses for this purpose its own distributed file system.
- In GPI-Space one can write much more complex workflows and the user has the freedom to chose, define and combine his own methods, instead of being restricted to a reduced set of primitives that are to be used for modeling the problem.
- In GPI-Space, the agents can form logical communication topologies with a structure of type graph, allowing thus a straightforward implementation of the algorithms that assume a certain structure of the logical communication graph.

Two important aspects must be taken into consideration when implementing the MapReduce pattern on top of GPI-Space: the organization of the virtual memory and the Petri net scheme that controls the generation and the execution of the tasks and their accesses to the virtual memory. As a graphical or XML description of the workflow is not possible here due to space limitations, we try to sketch out hereafter the main idea behind it. Initially, a configurable number of memory slots is allocated for each operation type (read, map, partition, reduce). The input data is split into chunks of equal sizes, which are then read in binary format, in parallel, and stored into the available read slots. As soon as a map slot becomes free, a nonempty read slot is chosen, its content is parsed and the map function is applied to the found items. The output pairs with the keys in the same partition are grouped together and all the groups are then stored into a map slot. Afterwards, the read slot is freed and can be reused for loading a new data chunk. As soon as a partition slot becomes free, a map slot that contains a contribution to that partition is chosen and the corresponding group of pairs is first sorted and then merged and reduced in one pass with the content of the partition slot. When a partition slot becomes full, its content is merged and reduced in one pass with the content of a free reduce slot, before allowing any other merge and reduce with the corresponding slice of a map slot. If a reduce

slot becomes full, its content is reduced on the disk with the corresponding file (typically, one per partition). An invariant of the application is that the partition slots and the reduce slots always contain pairs whose keys are sorted according to a comparison criterion provided by the user. The access to the virtual memory is controlled by the Petri net, which guarantees the mutual exclusion and the data integrity. Typically, to each slot it corresponds a token that may be consumed or released by the transitions corresponding to the MapReduce operations.

One problem that may arise here due to the fact that the data is read in binary format is that some words may be split into pieces that are stored into different data chunks. The Petri net scheme takes care of this aspect, too, and recovers the original words.

We considered as a use case the wordcount example. This is a typical example available with many MapReduce implementations, consisting in determining for each word appearing into a file or a collection of files the number of its occurrences. We performed tests with this use case on both GPI-Space and Hadoop frameworks. For GPI-Space we developed our own workflow, while with Hadoop we used the wordcount example provided in Hadoop-1.1.2, from the Apache incubator. As a hardware setup, we used a set of 12 cluster nodes, each node having 12 Intel Xeon X5660 cores (2.8 Ghz) and disposing of 48 GB RAM. Hadoop was configured to use one DataNode per node, disposing of 800 gigabytes local disk space on each node and using a replication factor of 3. GPI-Space was configured to use a half of the total aggregated RAM for building the partitioned global address space, each worker having reserved one gigabyte shared memory. With Hadoop, we used a block size of 64 megabytes. The figure 2 is a snapshot of the execution monitor of GPI-Space where a MapReduce workflow with the input data size of one terabyte is executed. In the left hand side, the cores and their capabilities are shown. The Gantt diagram illustrates the tasks that are executed and their status. The grey color is used for the created tasks, yellow for the started tasks, green for the finished tasks, red for the failed tasks and magenta for the canceled tasks. The figure shows a good CPU utilization and a fair distribution of the tasks over the cores. The figure 3 illustrates the execution time of the wordcount example with GPI-Space compared to the execution time with Hadoop. Wordcount performs better on GPI-Space than on Hadoop, in terms of execution times. The gap between the two execution times increases with the size of the input data. For 1031 gigabytes (which is approximately one terabyte) of data, the execution time on Hadoop is 9097 seconds while on GPI-Space is 3263 seconds. However, we should note that the data loading into GPI-Space is done at runtime, in parallel with the other MapReduce operations. Moreover, GPI-Space uses RDMA transfers and thus most of the communication overlaps the computation. GPI-Space does not require that all the input data be loaded into the virtual memory before starting the workflow. In the case of Hadoop, data loading and computing are two distinct phases: first the data is loaded into the Hadoop distributed file system (HDFS) and only afterwards the computations are started. The figure 4 shows in comparison the total time for processing input data files of different sizes, including the data loading, on both execution

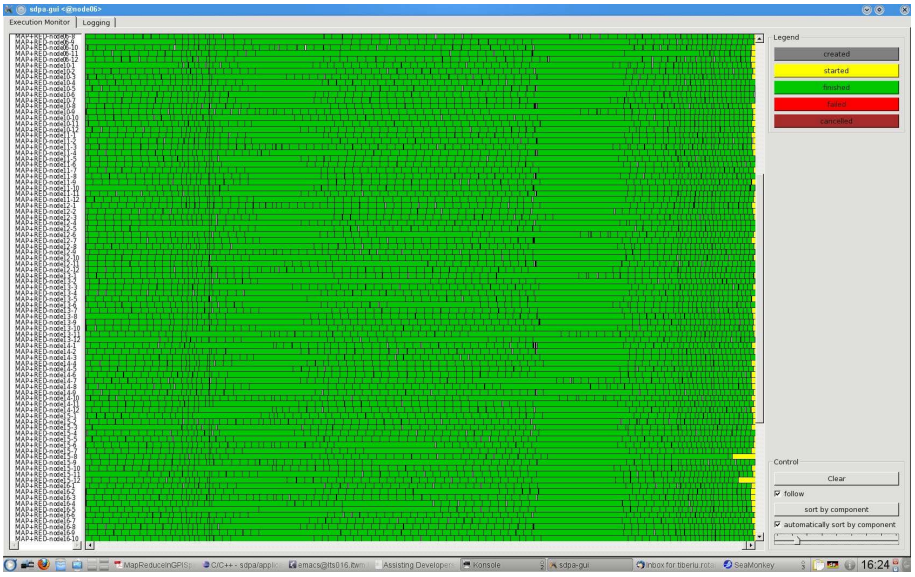

Fig. 2. The task execution monitor of GPI-Space showing the execution of a wordcount job with 1 TB input data

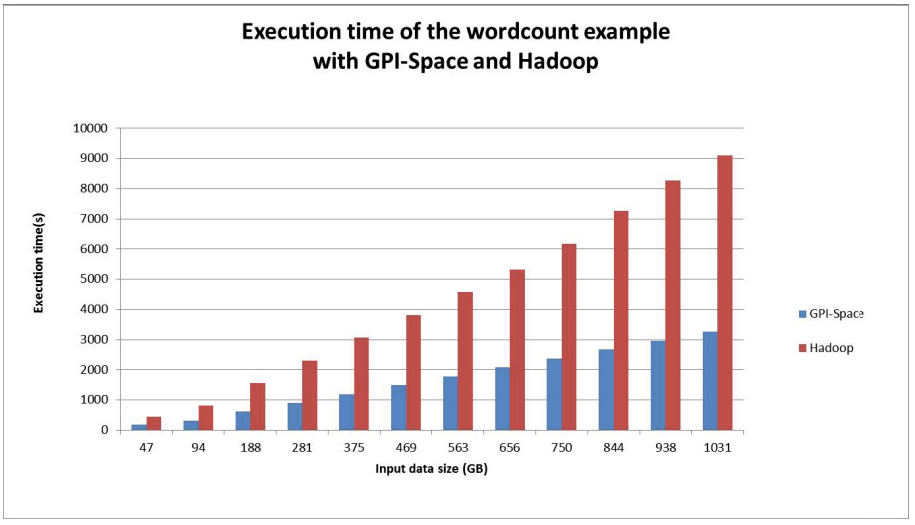

Fig. 3. Execution time of wordcount on GPI-Space and Hadoop

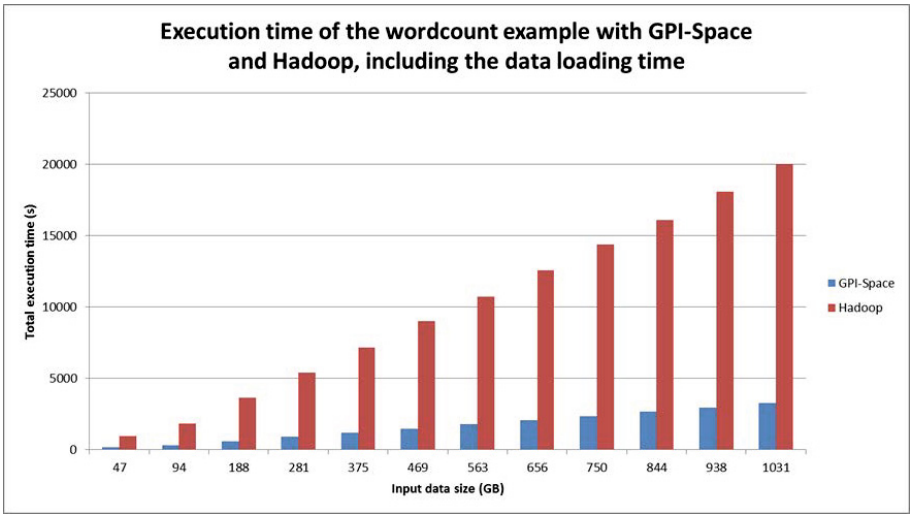

Fig. 4. Total processing time, including data loading, on GPI-Space and Hadoop

frameworks. In this case, processing one terabyte of data with GPI-Space takes 3263 seconds, while with Hadoop, if we count also the data loading time, it takes 20011 seconds. For the users who just want to occasionally use a MapReduce implementation for performing fast analytic operations and not willing to keep or migrate their data on HDFS for different reasons, like security for example, GPI-Space may represent a better option.

4 Conclusions and Future Work

In this paper we presented GPI-Space, a software development and execution platform for multicore clusters and we reported experiments with the construction of a MapReduce workflow on top of this architecture. GPI-Space is fault-tolerant and does automatic parallelization and scheduling, allowing the users to focus on the description of the workflow, primarily. Given the popularity of Hadoop, we considered appropriate to relate to this tool first. Compared to this, GPI-Space is more flexible, offering tools that facilitate the development of more complex workflows than MapReduce. In contrast to Hadoop, which stores the intermediary results on a distributed file system, GPI-Space stores most of the intermediary results into the partitioned global address space resulted from the aggregation of RAM slices of cluster nodes and is able to take advantage of fast RDMA data transfers. The example that we systematically tested, word-count, showed that GPI-Space is faster than Hadoop, when run over the same collection of cluster multicore nodes and using the same sample input data. Our approach has also the advantage that data loading into the virtual memory is carried out in parallel with the computations. Hadoop requires the data to be already on its distributed file system before the computations are started.

GPI-Space may represent a better alternative for the users who want to occasionally run fast MapReduce computations and not having the input data stored on HDFS. Our next goal is to implement more use cases for MapReduce in GPI-Space and to deploy and test the framework at larger scales.

References

1. http://www.gpi-site.com/gpi2/ (accessed October 2013)
2. Borthakur, D., Gray, J., Sarma, J.S., Muthukkaruppan, K., Spiegelberg, N., Kuang, H., Ranganathan, K., Molkov, D., Menon, A., Rash, S., Schmidt, R., Aiyer, A.: Apache hadoop goes realtime at Facebook. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, pp. 1071–1080. ACM, New York (2011)
3. Dean, J., Ghemawat, S.: Mapreduce: Simplified Data Processing on Large Clusters. In: Proceedings of the 6th Symposium on Opearting Systems Design and Implementation (OSDI 2004), San Francisco, CA, USA, pp. 137–150 (2004)
4. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software, 1st edn. Addison-Wesley Longman, Amsterdam (1995), 37. Reprint (2009)
5. Jin, H., Ibrahim, S., Qi, L., Cao, H., Wu, S., Shi, X.: The MapReduce Programming Model and Implementations. In: Buyya, R., Broberg, J., Goscinski, A. (eds.) Cloud Computing: Principles and Paradigms, pp. 373–390. John Wiley & Sons, Inc. (2011)
6. Lämmel, R.: Google's MapReduce programming model - Revisited. Sci. Comput. Program. 70(1), 1–30 (2008)
7. Lee, K.H., Lee, Y.J., Choi, H., Chung, Y.D., Moon, B.: Parallel data processing with MapReduce: a survey. SIGMOD Rec. 40(4), 11–20 (2012)
8. Lin, J., Ryaboy, D.: Scaling big data mining infrastructure: the twitter experience. SIGKDD Explor. Newsl. 14(2), 6–19 (2012)
9. Linn, J., Dyer, C.: Data-Intensive Text Processing With MapReduce. Synthesis Lectures on Human Language Technologies Series. Morgan & Claypool Publishers (2010)
10. Machado, R., Lojewski, C., Abreu, S., Pfreundt, F.J.: Unbalanced tree search on a manycore system using the GPI programming model. Computer Science - Research and Development 26(3-4), 229–236 (2011)
11. Rotaru, T., Dalheimer, M., Pfreundt, F.J.: Service-oriented middleware for financial monte carlo simulations on the cell broadband engine. Concurrency and Computation: Practice and Experience 22(5), 643–657 (2010)
12. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The Hadoop Distributed File System. In: Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2010, pp. 1–10. IEEE Computer Society, Washington, DC (2010)
13. Welsh, M., Culler, D., Brewer, E.: SEDA: an architecture for well-conditioned, scalable internet services. SIGOPS Oper. Syst. Rev. 35(5), 230–243 (2001)
14. White, T.: Hadoop - The Definitive Guide: Storage and Analysis at Internet Scale, 3rd edn. O'Reilly Media (2012)