A Cloudification Methodology for Numerical Simulations

Silvina Caíno-Lores, Alberto García,
Félix García-Carballeira, and Jesús Carretero

Universidad Carlos III de Madrid,
Department of Computer Science and Engineering,
Computer Architecture Group,
Leganés, Madrid, Spain
{scaino,agarcia,fgarcia,jcarrete}@arcos.inf.uc3m.es

Abstract. Many scientific areas make extensive use of computer simulations to study complex real-world processes. These computations are typically very resource-intensive and present scalability issues as experiments get larger, even in dedicated clusters since they are limited by their own hardware resources. Cloud computing raises as an option to move forward into the ideal unlimited scalability by providing virtually infinite resources, yet applications must be adapted to this new paradigm. We propose a generalist cloudification method based in the MapReduce paradigm to migrate numerical simulations into the cloud to provide greater scalability. We analysed its viability by applying it to a real-world simulation and running the resulting implementation on Hadoop YARN over Amazons EC2. Our tests show that the cloudified application is highly scalable and there is still a large margin to improve the theoretical model and its implementations, and also to extend it to a wider range of simulations.

1 Introduction

Scientific simulations constitute a major set of applications that attempt to reproduce real-world phenomena in a wide range of areas such as engineering, physics, mathematics and biology. Their complexity usually yields a significant resource usage regarding CPU, memory, I/O or a combination of them.

In order to properly scale the application it can be distributed to a cluster or grid. While these approaches have proved successful, they often rely on heavy hardware investment and they are tightly conditioned by its capabilities, which de facto limits actual scalability and the addressable simulation size. Since sharing resources across multiple clusters implies several limitations, cluster applications cannot be considered sustainable, because their scalability is strongly dependant on the cluster size.

Despite scientific simulations will likely benefit from the upcoming exascale infrastructures [1], the challenges that must be overcome –power consumption, processing speed and data locality, for instance [2]– will probably rise again in the future as applications become more complex; therefore, the ideal situation of unlimited scalability seems difficult to reach with this approach.

Moreover, recent advances in cloud interoperability and cloud federations can contribute to separate application scalability from datacenter size [7, 12]. From that point of view, applications would become more sustainable, i.e. they can be operated in a more flexible way through heterogeneous hardware, cross-domain interactions and shared infrastructures.

Another recent option is cloud computing, which has been increasingly studied as an alternative to traditional grid and high-performance distributed environments for resource-demanding and data-intensive scientific simulations [15]. Cloud computing emerged with the idea of virtual unlimited resources obtainable on-demand with minimal management effort [11]. It would enable the execution of large simulations with virtual hardware properly tailored to fit specific use cases like memory-bound simulations, CPU-dependant computations or data-intensive analysis. It holds further advantages, such as elasticity, automatic scalability and instance resource selectivity which, along with its so-called pay-as-you-go model, allow to adjust the required instances to the particular test case size while cutting-down the resulting costs.

There are several issues that can be tackled in order to develop a sustainable application, such as:

– Virtual unlimited scalability can be achieved by eliminating architectural bottlenecks such as network communications or master node dependences. This minimises the added overhead of working with more nodes, making a better use of the available resources.
– By making the application platform independent, we can aggregate computational resources possibly located in different places, hence local data center size would not be a limitation. Moreover, we can exploit cluster and cloud resources simultaneously following an hybrid scheme.
– A flexible application could scale up or down easily according to instantaneous user needs, thus adapting computing resources to specific simulation sizes and deadlines.
– If the application already exists and has to be adapted, it is desirable to minimize the impact on the original code, thus performing the minimal modifications needed to achieve the aforementioned objectives.

Given the former, we suggest a paradigm shift from multi-thread computations to a data-centric model that would distribute the simulation load across a set of virtual instances. This paper focuses on resource-intensive numerical simulations which hold potential scalability issues on large cases, since standalone and cluster hardware may not satisfy simulation requirements under such stress circumstances, and it proposes a generic methodology to transform numerical simulations into a cloud-suitable data-centric scheme via the MapReduce framework.

This process is illustrated by means of a real production application, a simulator which calculates power consumption on railway installations. This simulator, starting from the train movements (train position and consumption), calculates the instantaneous power demand (taking into account all railway elements such as tracks, overhead lines, and external consumers) indicating whether the power provisioned by power stations is enough or not. Simulator internals consist on composing the electric circuit on each instant, and solving that circuit using modified nodal analysis. The starting version of the simulator, based on multi-threading, is memory bounded, strongly limited by the number of instants to be simulated simultaneously (and therefore by the number of threads). The resulting performance is evaluated on Amazon Elastic Compute Cloud running Hadoop YARN MapReduce.

The rest of this paper is organized as follows: Section 2 discusses related works, Section 3 describes our proposed methodology, Section 4 illustrates the cloudification transformation method on a particular use case, Section 5 evaluates how the resulting design implementation on Hadoop MapReduce 1.1.2 (MRv1) and Hadoop YARN Mapreduce 2.2.0 (MRv2) behaves on both a cluster and Amazon Elastic Compute Cloud (EC2) and, finally, Section 6 provides key ideas as conclusions and some insight in future work.

2 Related Work

Scientific applications and their adaptability to new computing paradigms have been dragging increasing attention from the scientific community in the last few years. The applicability of the MapReduce scheme for scientific analysis has been notably studied, specially for data-intensive applications, resulting in an overall increased scalability for large data sets, even for tightly coupled applications [6].

The possibility to run such simulations in the cloud in terms of cost and performance was studied in [10], concluding that performance in the Abe HPC cluster and Amazon EC2 is similar –besides the virtualization overhead and high-speed connectivity loss in the cloud– and that clouds are a viable alternative for scientific applications. Hill [9] investigated the trade-off between the resulting performance and achieved scalability on the cloud versus commodity clusters; despite at the time of this work the cloud could not properly compete against HPC clusters, its low maintenance and cost made it a viable option for small scale clusters with a minimum performance loss.

The relationship between Apache Hadoop MapReduce and the cloud for scientific applications has also been tackled in [8], which establishes that performance and scalability tests results are similar between traditional clusters and virtualized infrastructures.

In this context, trends are naturally evolving to migrate applications to the cloud by means of several techniques, and this includes scientific simulations as well. D'Angelo [4] describes a Simulation-as-a-Service schema in which parallel and distributed simulations could be executed transparently, which requires dealing with model partitioning, data distribution and synchronization. He concludes that the potential challenges concerning hardware, performance, usability and cost that could arise could be overcome and optimized with the proper simulation model partitioning.

In [13], Srirama, Jakovits and Vainikko study how some scientific algorithms could be adapted to the cloud by means of the Hadoop MapReduce framework. They establish a classification of algorithms according to the structure of the MapReduce schema these would be transformed to and suggest that not all of them would be optimally adapted by their selected MapReduce implementation, yet they would suit other similar platforms such as Twister or Spark. They focus on the transformation of particular algorithms to MapReduce by redesigning the algorithms themselves, and not by wrapping them into a cloudification framework as we propose.

Finally, in [14] we find interesting efforts to move desktop simulation applications to the cloud via virtualized bundled images that run in a transparent multi-tenant fashion from the end user's point of view, while minimizing costs. As previously discussed, we believe the virtualization middleware might affect performance since it does not take into account any structural characteristics of the model, which could be exploited to minimize cloudification effects or drastically affect execution times or resource consumption.

Our work focuses in providing a general methodology to transform numerical simulations into a cloud-suitable execution framework with minimal impact to the original code, while exploiting simulation model features that inherently aid with partitioning and performance optimization. A related approach is the so-called *parameter sweep* [3], in which the same simulation kernel is executed multiple times with different input parameters, thus providing task independence. However, in our approach we transform a single simulation into several autonomous tasks through any independent variable that belongs to the simulation domain, not only input parameters. Domain decompositions and transformations can be used in applications where task independence is not so evident; therefore, task independence is a result of our methodology, not a means.

3 Methodology Description

The MapReduce paradigm consists of two user-defined operations: *map* and *reduce*. The former takes the input and produces a set of intermediate $(key, value)$ pairs that will be organized by key by the framework so that every reducer gets a set of values that correspond to a particular key [5].

As a data-centric paradigm, in which large amounts of information can be potentially processed, these operations run independently and only rely upon the input data they are fed with. Thus, several instances can run simultaneously with no further interdependence. Moreover, data can be spread across as many nodes as needed to deal with scalability issues.

Simulations, however, are usually resource-intensive in terms of CPU or memory usage, so their scalability is limited to hardware restrictions, even in large clusters. Our goal is to exploit the data-centric paradigm to achieve a virtually infinite scalability so that large numeric simulations can be executed independently of the underlying hardware resources, with minimal effects to the original simulation code. From this point of view, numeric simulations would become

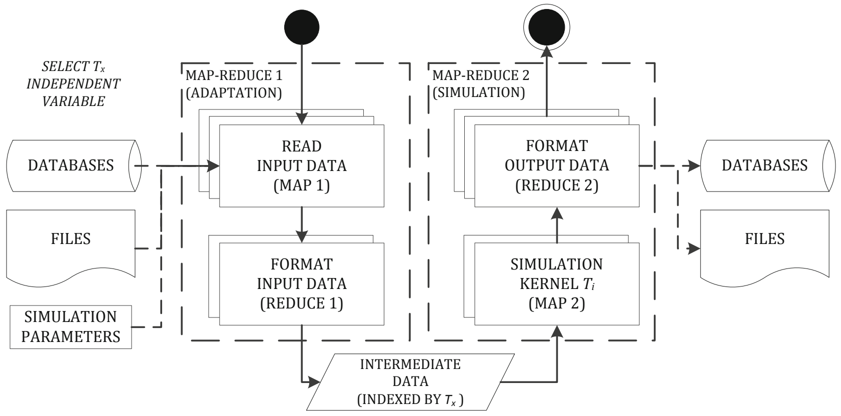
SELECT T_x INDEPENDENT VARIABLE
DATABASES
FILES
SIMULATION PARAMETERS
MAP-REDUCE 1 (ADAPTATION)
READ INPUT DATA (MAP 1)
FORMAT INPUT DATA (REDUCE 1)
INTERMEDIATE DATA (INDEXED BY T_x)
MAP-REDUCE 2 (SIMULATION)
SIMULATION KERNEL T_i (MAP 2)
FORMAT OUTPUT DATA (REDUCE 2)
DATABASES
FILES

Fig. 1. Methodology overview

more sustainable, allowing us to spread simulation scenarios of different sizes in a more flexible way, using heterogeneous hardware, and taking advantage of shared inter-domain infrastructures.

To achieve this, we will take advantage of MapReduce's lack of task interdependence and data-centric design, this will allow to disseminate the simulation's original input to distribute its load among the available nodes, which will yield the scalability we aim for. The steps involved in our proposed methodology are described in the following sections.

3.1 Application Analysis

Our purpose is to divide the application into smaller simulations that can run with the same simulation kernel but on a fragment of the full partitioned data set, so that we can parallelise the executions and lower the hardware requirements for each.

Hence, we must analyse the original simulation domain in order to find an independent variable –T_x in Fig. 1– that can act as index for the partitioned input data and the following procedures. This independent variable would be present either in the input data or the simulation parameters and it could represent, for example, independent time-domain steps, spatial divisions or a range of simulation parameters.

3.2 Cloudification Process Design

Once the application is shown suitable for the process, it can be transformed by matching the input data and independent variables with the elements in Fig. 1, thus resulting in the two MapReduce jobs described below:

– **Adaptation stage:** reads the input files in the *map* phase and indexes all the necessary parameters by T_x for every execution as intermediate output. The

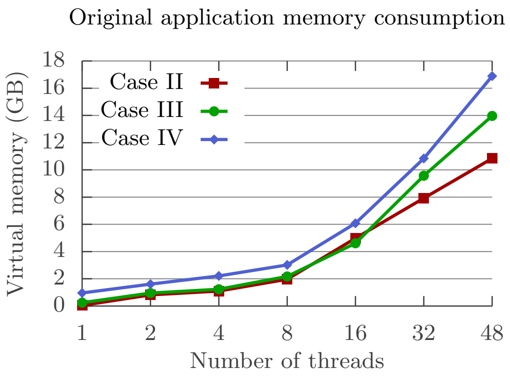

Fig. 2. Original simulation kernel memory consumption

Table 1. Test cases definition

Experiment	Simulated time (hours)	Input size (MB)
I	1	1.7
II	33	170
III	177	1228.8
IV	224	5324.8

Table 2. Execution environments

Configuration	Platform	Underlying infrastructure
1	Multi-thread	Cluster node
2	MRv1	Cluster node
3	MRv2	Cluster node
4	MRv2	EC2

original data must be partitioned so that subsequent simulations can run autonomously with all the necessary data centralized in a unique (T_x, $parameters$) entry.

– **Simulation stage:** runs the simulation kernel for each value of the independent variable along with the necessary data that was mapped to them in the previous stage, plus the required simulation parameters that are common for every partition. Since simulations might generate several output files, mappers would organize the output by means of file identifier numbers as keys, so as reducers could be able to gather all the output and provide final results as the original application.

4 Case Study

To illustrate how this methodology works on a real-world use case, we applied it to transform a memory-bound railway electric power consumption simulation.

Four test cases were considered with variations on the simulation's initial and final time and, consequently, input data volume and memory consumption. A description of these simulations is provided in Table 1. Cases I and II should

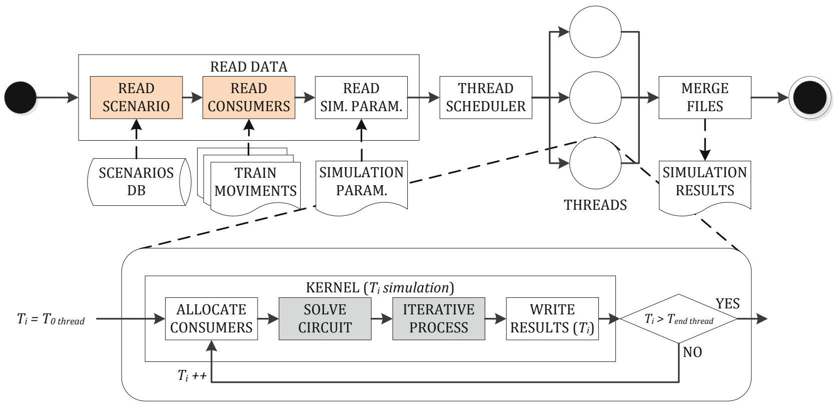

Fig. 3. Case study original simulation structure

not yield any significant load, yet simulation III is expected to show some differences, while the biggest experiment, case IV, should reveal the platforms' actual behaviour and limitations as simulations become larger, if any. These tests are meant to indicate the performance of the cloudified adaptation versus the original application under an increasing amount of input data and simulation time.

As seen in Fig. 2, this application does not scale well for large test cases in terms of memory usage in a standalone environment (Configuration 1, see Sec. 5.1 for further details). We believe we can achieve greater scalability by cloudifying the application, since we can distribute the simulation load across several nodes. It would also disperse memory usage so that we could always add a new node in case we need to tackle a larger case. To show its feasibility, next we will apply the method described in Section 3.

4.1 Analysis

The structure of the selected application is shown in Fig. 3. It consists of a preparation phase in which all the required input data is read and fragmented to be executed in a predefined number of threads. Each of the resulting threads then perform the actual simulation by means of an electric iterative algorithm, storing in shared memory the results that will be merged in the main thread to constitute the final output files.

This simulator, starting from the train movements –that describe train position and power consumption– and infrastructure design –tracks, power stations, among others– calculates the instantaneous power demand taking into account all railway elements such as tracks, overhead lines, and external consumers, indicating whether the power provisioned by power stations is sufficient or not. Simulator internals consist on composing the electric circuit on each instant, and solving that circuit using modified nodal analysis. The initial version of the simulator, based on multi-threading, is memory bounded, strongly limited by the number of instants to be simulated simultaneously, and therefore by the number of threads.

The key to adapt such algorithm to a cloud environment resides its input files, for they hold an indexed structure that stores in each line an $(instant, parameters)$ pair. Therefore, we can consider the temporal key as the independent variable required for the theoretical model.

4.2 Cloudification

Following the cloudification schema, the application was transformed into two independent MapReduce jobs executed sequentially.

In the first job, which matches the first MapReduce in Fig. 1, the movement input files, I_k, are divided into input splits by the framework according to its configuration. Each split is then assigned to a mapper, which reads each line and emits $(key, value)$ pairs where the key is the instant t_i and the value is the corresponding set of parameters for such instant; the intention is to provide reducers with a list of movement parameters per instant $I_n, \ldots, I_m$ –each element representing the movement of one of the trains involved in the overall system for a particular t_i– to concatenate and write to the output files, so that the simulation kernel can be executed once per instant with all the required data.

As described in Fig. 1, the output of the previous job is used as input to the mapper tasks by parsing each line in order to get the data corresponding to the instant being processed, which is passed to the electric algorithm itself along with the scenario information obtained from the infrastructure file that is also read by the mapper. The mappers' output is compounded by an output file identifier F_j as key and the actual content as value.

Reducers simply act as mergers gathering and concatenating mappers' output organized by file identifier and instant as a secondary key injected in the value content; this arranges the algorithm's output so that the full simulation results are shown as in the original application, in which each output file contains the results for the whole temporal interval of the simulation.

5 Evaluation

In order to asses the application's performance we compared its execution times on both a cluster and the cloud. The following sections describe the utilized resources and a discussion on the obtained outcome.

5.1 Execution Environments

Table 2 summarizes the infrastructures and software platforms on which the tests were conducted.

In a first place, we tested the original multi-thread application's memory consumption and performance on a cluster node consisting of a 24 Xeon E7 cores and 110GB of RAM (Configuration 1).

This node was also used to test the resulting cloudfied application to avoid variations that may arise from heterogeneous configuration, resource differences, or network latency in case of the MapReduce application [10]. This isolation favours the multi-thread application, which is especially designed to perform in standalone environments, yet it allows to focus on the actual limiting factors that may affect scalability in large test cases like I/O, memory consumption and CPU usage. Both Hadoop versions –MRv1 and MRv2– were installed and configured on the single-node cluster to benchmark their performance against the original application (Configurations 2 and 3, respectively).

MRv2 was chosen to be deployed on EC2 given its improved resource management options and better overall performance (Configuration 4). The cloud infrastructure consisted of a general purpose *m1.medium* node as dedicated master and several memory optimized *m2.xlarge* machines as slaves, with 2 CPUs and 17.1GB of RAM each. Tests on EC2 have been conducted using a variable number of slaves in order to check if scalability issues arise as the number of nodes increases.

5.2 Results Discussion

As we already discussed in Section 4, the original multi-thread application's memory usage suggests a lack of scalability in a cluster environment. We will now analyse whether the cloudified simulation behaves as expected in relation to performance and scalability by examining its execution times on several execution environments, which are shown in Fig. 4. This figure shows the time measurements obtained on the configurations in Tab. 2, in which the EC2 cluster is constituted by five slaves –graphs (a), (b) and (c)–. The EC2 values also served as baseline for the scalability study shown in (d).

(a) Cloudification phase
The data adaptation phase –graph (b)– is 65% slower on EC2 compared to the same MapReduce version in the local cluster, for the largest experiment. This is a result of the selected EC2 instances' characteristics, since memory optimised machines are meant to favour the memory-bound kernel execution phase. This stage would benefit from compute optimised instances, since a large number of cores would allow the execution of more mappers simultaneously.

(b) Kernel execution
The algorithm execution stage, (c), is the most determinant phase in the whole process, ranging from the 48% of the whole execution time, in case I on EC2, to an 89%, in case II in the same environment. The total resources held by the physical cluster in terms of memory make a substantial difference in this stage, resulting in simulation times 2.1 times lower than EC2, in average. Cloud's virtualization and communication overhead could also affect the simulation execution and the shuffle of the mapper's output, respectively, degrading performance against the single-node environment.

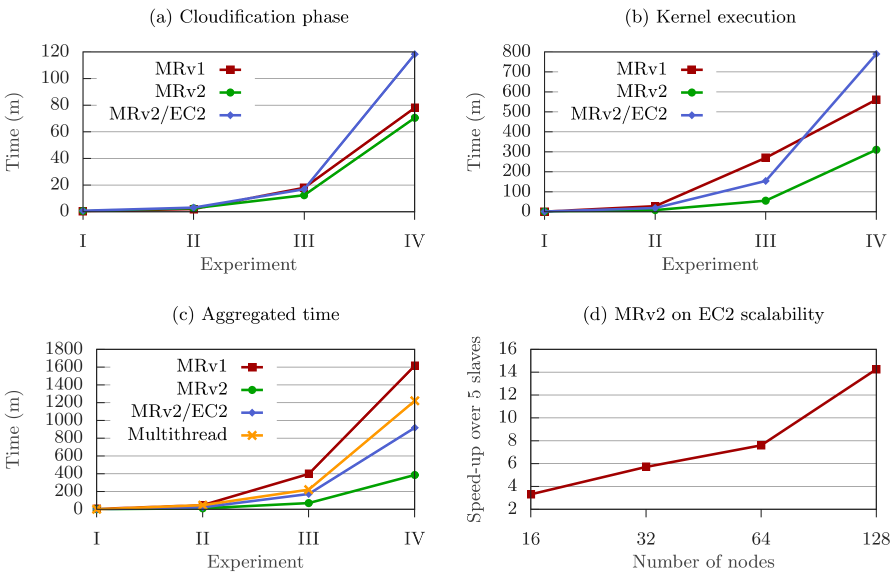

Fig. 4. Evaluation results

(c) Aggregated time
In (c) we observe the overall execution time for the application including both MapReduce jobs and input data upload, which must be considered given that replication and balance must be achieved by the platform to distribute load evenly. The graph indicates that the obtained performance with MapReduce on Yarn in both the single-node cluster and the elastic cloud is remarkably better than the original multi-thread application –68% and 25% less total simulation time for the largest experiment, respectively–. The shared memory simulator's results might be caused by the bottleneck constituted by the physical memory and the disk; the latter is particularly critical, as all threads write their results to disk while they perform their computations in the original simulator.
The smallest experiment is an interesting exception, with execution times ten times greater than the original application in all the platforms. This reflects how the MapReduce framework's overhead significantly affects the time taken to complete such a small simulation compared to the original application benchmark.

(d) Scalability study
Finally, in (d) we observe the speed-up obtained on EC2 running YARN when the number of slaves is increased. The speed-up shown in the figure is related to the execution times commented in the previous paragraphs, which were obtained in a five-slave cluster. As the figure indicates, increasing the number of slaves decreases the total simulation time. However, the

performance does not scale up linearly with the number of nodes: while with 16 nodes the speed-up is 3.3, with 64 nodes it is only 7.6. The reason behind this result is that the problem size becomes small for the cluster size as more nodes are added, hence less data is assigned to each slave and some resources become underutilised. Moreover, as we mentioned in the previous paragraph, in very small experiments the measured execution time is mostly spent in the platform's task preparation and scheduling, and not in the actual simulation, resulting in degraded performance due to platform overhead. Therefore, it is necessary to increase the problem size as well as the number of slave nodes in order to achieve linear scalability.

6 Conclusions

As the cloud is increasingly shown as a viable alternative to traditional computing paradigms for high-performance applications and resource-intensive simulations, we propose a general methodology to transform numeric simulations into a highly scalable MapReduce application that re-uses the same simulation kernel while distributing the simulation load across as many nodes are desired in a virtual cluster running on the cloud.

The procedure requires an application analysis phase in which at least one independent variable must be found, since this element will act as index for the cloudification phase. The cloud adaptation stage transforms the original input into a set of partitions indexed by the the previous variable by means of a MapReduce job; these partitions are fed to a second MapReduce job that executes the simulation kernel independently for each, merging the final results as well.

This methodology performs a paradigm shift from resource-bound applications to a data-centric model; such cloudification mechanism provides effective cloud migration of simulation kernels with minimal impact on the original code and achieves great scalability since limiting factors are scattered. Therefore, it provides a way to increase application's sustainability, breaking the dependence on local infrastructure, and allowing to spread simulation scenarios of different sizes in a more flexible way, using heterogeneous hardware, and taking advantage of shared inter-domain infrastructures.

Future works are strongly focused on extending the current methodology to a generalized framework which would allow to cloudify any scientific application. With this aim, several issues have to be solved:

– The behaviour of the methodology should be analysed when other different kinds of applications (CPU or network intensive) are cloudified. Currently we are cloudifying a classic MPI application such as the $n-$bodies problem, in order to assure performance even in cluster-oriented applications.
– Parameter extraction and application analysis is currently performed manually by the user, who is accountable for selecting an independent variable T_x. Current development is also oriented to ease this tasks through creating

data definitions which would allow the adaptation phase to select and split the input data automatically.

References

1. Ashby, S., Beckman, P., Chen, J., Colella, P., Collins, B., Crawford, D., Dongarra, J., Kothe, D., Lusk, R., Messina, P.: et al.: The opportunities and challenges of exascale computing. Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee (November 2010)
2. Bergman, K., Borkar, S., Campbell, D., Carlson, W., Dally, W., Denneau, M., Franzon, P., Harrod, W., Hill, K., Hiller, J., et al.: Exascale computing study: Technology challenges in achieving exascale systems. Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Tech. Rep 15 (2008)
3. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for scheduling parameter sweep applications in grid environments. In: Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000), pp. 349–363. IEEE (2000)
4. D'Angelo, G.: Parallel and distributed simulation from many cores to the public cloud. In: 2011 International Conference on High Performance Computing and Simulation (HPCS), pp. 14–23 (July 2011)
5. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM 51(1), 107–113 (2008)
6. Ekanayake, J., Pallickara, S., Fox, G.: Mapreduce for data intensive scientific analyses. In: IEEE Fourth International Conference on eScience 2008, pp. 277–284 (December 2008)
7. Grozev, N., Buyya, R.: Inter-cloud architectures and application brokering: taxonomy and survey. Software: Practice and Experience (2012)
8. Gunarathne, T., Wu, T.L., Qiu, J., Fox, G.: Mapreduce in the clouds for science. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), pp. 565–572 (November 2010)
9. Hill, Z., Humphrey, M.: A quantitative analysis of high performance computing with amazon's ec2 infrastructure: The death of the local cluster? In: 2009 10th IEEE/ACM International Conference on Grid Computing, pp. 26–33 (October 2009)
10. Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B., Maechling, P.: Scientific workflow applications on amazon ec2. In: 2009 5th IEEE International Conference on E-Science Workshops, pp. 59–66 (December 2009)
11. Mell, P., Grance, T.: The nist definition of cloud computing. National Institute of Standards and Technology 53(6), 50 (2009)
12. Petcu, D., Macariu, G., Panica, S., Crăciun, C.: Portable cloud applications—from theory to practice. Future Generation Computer Systems 29(6), 1417–1430 (2013)
13. Srirama, S.N., Jakovits, P., Vainikko, E.: Adapting scientific computing problems to clouds using mapreduce. Future Generation Computer Systems 28(1), 184–192 (2012)
14. Srirama, S.N., Ivanistsev, V., Jakovits, P., Willmore, C.: Direct migration of scientific computing experiments to the cloud. In: 2013 International Conference on High Performance Computing and Simulation (HPCS), pp. 27–34 (July 2013)
15. Yelick, K., Coghlan, S., Draney, B., Canon, R.S., et al.: The magellan report on cloud computing for science. US Department of Energy, Washington DC, USA, Tech. Rep. (2011)