D-Mason on the Cloud: An Experience with Amazon Web Services

Michele Carillo[1], Gennaro Cordasco[2], Flavio Serrapica[1], Carmine Spagnuolo[1](✉), Przemysaw Szufel[3], and Luca Vicidomini[1]

[1] ISISLab–Dipartimento di Informatica, Università degli Studi di Salerno, Fisciano, Italy
michele.carillo@gmail.com, flavio.serrapica@gmail.com
{cspagnuolo,lvicidomini}@unisa.it
[2] Dipartimento di Psicologia, Seconda Università degli Studi di Napoli, Caserta, Italy
gennaro.cordasco@unina2.it
[3] Warsaw School of Economics (WSE - SGH), Warsaw, Poland
pszufe@gmail.com

Abstract. D-Mason framework is a parallel version of the Mason library for writing and running Agent-based simulations – a class of models that, by simulating the behavior of multiple agents, aims to emulate and/or predict complex phenomena. D-Mason has been conceived to harness the amount of unused computing power available in common installations like educational laboratory. Then the focus moved to dedicated installation, such as massively parallel machines or supercomputing centers. In this paper, D-Mason takes another step forward and now it can be used on a cloud environment.

The goal of the paper is twofold. Firstly, we are going to present D-Mason on the cloud – a D-Mason extension that, starting from an IaaS (Infrastructure as a Service) abstraction, and exploiting Amazon Web Services and StarCluster, provides a SIMulation-as-a-Service (SIMaaS) abstraction that simplifies the process of setting up and running distributed simulations in the cloud. Secondly, an additional goal of the paper is to assess computational and economic efficiency of running distributed multi-agent simulations on the Amazon Web Services EC2 instances. The computational speed and costs of an EC2 cluster will be compared against an on-site HPC cluster.

Keywords: Agent-Based simulation Models · Cloud computing · D-Mason · Parallel computing · Distributed systems · High performance computing

1 Introduction

Computational science is a rapidly growing novel field that uses advanced computing in order to solve complex problems. This new discipline combines new technologies, modern computational methods and simulations to address problems too complex to be reliably predicted by theory and too dangerous or expensive to be reproduced in the laboratory.

Many simulation paradigms have been proposed. Among them, Agent-Based simulation Models (ABMs) are an increasingly popular tool in terms of expressiveness and easy to understand for the developer of simulation models [9].

Successes in computational sciences over the past ten years have caused demand for supercomputing resources, to improve the performance of the system and to allow the growth of the models, in terms of sizes and quality. From a computer scientist's perspective, it is natural to think to distribute the execution of the simulations among multiple machines: it is well known that the speed of single-processor computers is reaching some physical limits. For these reasons, parallel computing has become the dominant paradigm for computational scientists who need the latest development on computing resources in order to solve their problems.

The cloud computing paradigm [1] is becoming very popular these days. With cloud computing, the cloud vendors provide IT resources to users as a utility like the electricity; the user accesses the IT resources easily and pays only for the ones they consumed. Cloud providers offer managed infrastructures (IaaS - Infrastructure as a Service) as well as managed platforms such as a key-value storage or a relational database (PaaS - Platform as a Service). The offered IaaS and PaaS services enable Cloud computing services that range from simple data backup to the possibility of deploying entire computing clusters or data centers in a remote environment.

The goal of cloud computing is to allow users to take benefit from IT resources, without the need for deep knowledge about or expertise with each one of them. Moreover, individual user or small groups do not need to provide and maintain IT-infrastructure on their own, but instead rely on cloud services to satisfy their needs. The flexibility, cost-efficiency, scalability, accessibility as well as user-friendliness of cloud services make it also an attractive model to address computational challenges in the scientific community. The design of ABMs is usually done by domain experts who seldom are computer scientists and have limited knowledge of managing a modern parallel infrastructure. In this context, there is the need of tools that allow, in a transparent way, the use of cloud resources to non-expert users [7,8].

In this paper, we introduce D-Mason on the cloud – a D-Mason extension that provides a SIMulation-as-a-Service (SIMaaS) infrastructure that simplifies the process of setting up and running distributed simulations in the cloud. We will present a preliminary evaluation of the novel infrastructure in order to assess computational and economic efficiency of running distributed multi-agent simulations on the Amazon Web Services EC2 instances. The computational speed and costs of an EC2 cluster will be compared against an on-site HPC cluster.

2 Background

2.1 D-Mason

D-Mason [3,4] is the distributed version of Mason [10,11], a discrete-event simulation core and visualization library written in Java, designed to be used for a wide range of ABMs. Mason is composed of two independent layers: the *simulation* layer and the *visualization* layer. D-Mason adds a new layer named *D-simulation*, which extends the Mason simulation layer. The new layer adds some features that allow the simulation work distribution on multiple, even heterogeneous, Logical Processors (LPs). D-Mason has been designed to enable the porting of existing applications to a distributed platform in a transparent and easy way.

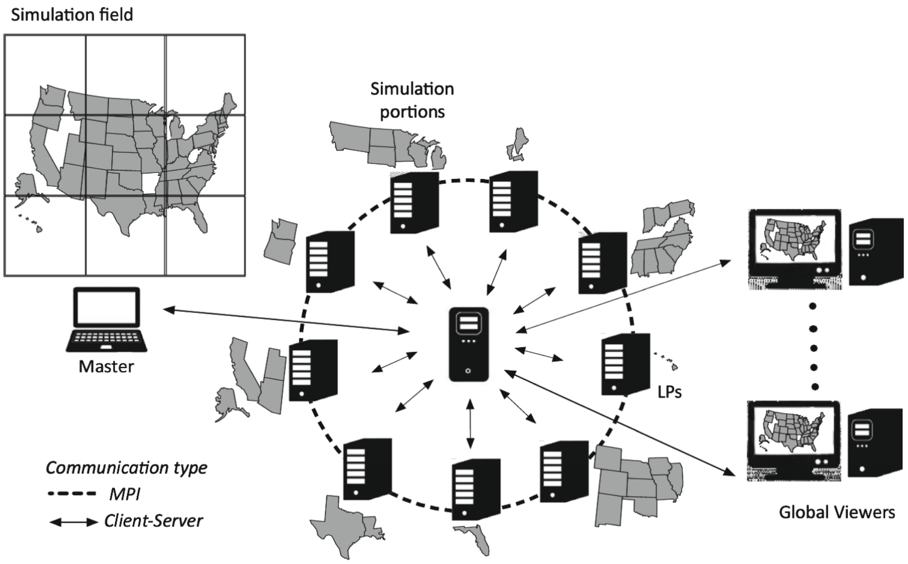


Fig. 1. D-Mason scheme.

D-Mason is based on a Master/Workers paradigm that exploits a space partitioning approach: the master partitions the space to be simulated (i.e., the field) into cells (see Fig. 1). Each cell, together with the agents it contains, is assigned to an LP; then each LP is in charge of:

– simulating the agents that belong to the assigned cell;
– handling the migration of agents;
– managing the synchronization between neighboring cells (this information exchange is required in order to let the simulation run consistently).

D-Mason LPs communicate via a well-known mechanism, based on the Publish/Subscribe paradigm: a multicast channel is bound to each cell; LPs then simply subscribe to the topics associated with the cells, which overlap with their Area of Interest (AOI) in order to receive relevant message updates. Other topics are also used for system management and visualization.

D-Mason has been conceived to harness the amount of unused computing power available in common installations like educational labs, that is, a loosely coupled environment with heterogeneous machines. To take advantage of this environment, the former version of D-Mason used a centralized communication mechanism (JMS), while the main emphasis was represented by load balancing [6]. Indeed, when the number of LPs available is limited, a centralized communication is, at the same time, easy to develop/manage and efficient. Then the initial design idea has spanned beyond this. The focus moved to dedicated installation, such as massively parallel machines or supercomputing centers. These platforms usually offer a large number of homogeneous machines that, on one hand, simplify the issue of balancing the load among LPs, but, on the other hand, the considerable computational power provided by the system weakens the efficiency of the centralized communication server. For this reasons, a novel decentralized communication mechanism, which realizes a Publish/Subscribe paradigm through a layer based on the MPI standard, was implemented in D-MASON [5].

2.2 Amazon Web Services

Amazon Web Services (AWS) is a scalable and highly reliable cloud infrastructure for deploying applications on demand. The main idea is to let the user building its services with minimal support and administration costs. AWS provides different services on the cloud. In this work, we are interested to the web services that enable either the modeler or the developer to run their simulation on the cloud. Amazon Elastic Compute Cloud (Amazon EC2) provides resizable computing capacity in the cloud. In terms of abstraction layers, the Amazon EC2 is an instance of the Infrastructure as a Service (IaaS) model, where the Amazon infrastructure is seen as a complete virtual environment which allows to execute different instances of virtual machines. Specifically, Amazon allows bundling operating system, application software and configuration settings into an Amazon Machine Image (AMI). Then each user can configure and deploy a cluster of machines using a specific AMI instance to run distributed simulations. Advanced users may also create their own AMIs and publish them on the Amazon Marketplace Web Service (Amazon MWS).

In terms of business model, Amazon offers three different purchasing mechanisms: On-Demand Instances, Reserved Instances and Spot Instances. On-Demand Instances have fixed price (per hour) and allow using the resources immediately. With Reserved Instances, it is possible to reserve the utilization of some instances for a predefined period (from 1 to 3 years) with lower payment. Finally, when the timing is not crucial, with Spot Instances, it is also possible to bid for unused resources in order to reduce drastically the costs.

2.3 StarCluster

The main issue a user needs to solve in order to use an IaaS service, to run a distributed application, is the configuration and management of each machine. Even using a dedicated AMI, which bundle the basic software components, there are still several parameters that have to be configured separately on each machine. Moreover, the management of the machines is usually time-consuming and requires repetitive tasks that need to be executed for each instance and therefore should be automate to avoid human mistake. To face this issue, a cluster-computing toolkit, StarCluster [15], released under the LGPL license, has been deployed to configure and manage Amazon EC2 instances. StarCluster enables users to easily setup a cluster computing environment in the cloud, suited for distributed and parallel computing applications and systems.

StarCluster is useful to configure the network of the cluster, create user accounts, enable password-less connections sharing the SSH password between the cluster's nodes, setup NFS shares and the queuing system for the jobs. StarCluster is also customizable via plug-ins, which allow users to configure further the cluster with their specific configuration. Plugins are written in Python exploiting StarCluster API to interact with the nodes. The API supports executing commands, copying files, and other OS-level operations on the nodes. StarCluster supports also the use of Spot instances allowing the user to run on-demand experiments in easy way and at affordable prices.

3 D-Mason on the Cloud

D-Mason on the cloud has been realized with the purpose to provide a SIMulation-as-a-Service (SIMaaS) environment. The architecture of the system is depicted in Fig. 2. D-Mason on the cloud is based on a modular approach, which comprises three levels: The Infrastructure is given by Amazon EC2 which provides a wide portfolio of instance types [13] designed to be adopted for different use cases. Instance types vary by CPU performances, memory, storage (size and performance), and networking capacity. The user is free to select an AWS cell according to prices and availability or resources. Starting with a free available Amazon AMI (ami-52a0c53b) that includes a minimal software stack for distributed and parallel computing [15], we realized an AMI specifically configured for executing D-Mason on the cloud. The D-Mason AMI, public available on Amazon Infrastructure, provides also Java 8, Maven, D-Mason 3.1. On top of that, we developed a StarCluster plugin, which exploits all the functionality provided by StarCluster in order to create automatically a runnable D-Mason environment based on the D-Mason AMI. With more details, the StarCluster plugin:

- configure the cluster network environment (users account, hostnames setting, SSH key share, NFS setup);
- appoint one of the machines as a Master node;
- install and configure the D-Mason environment.

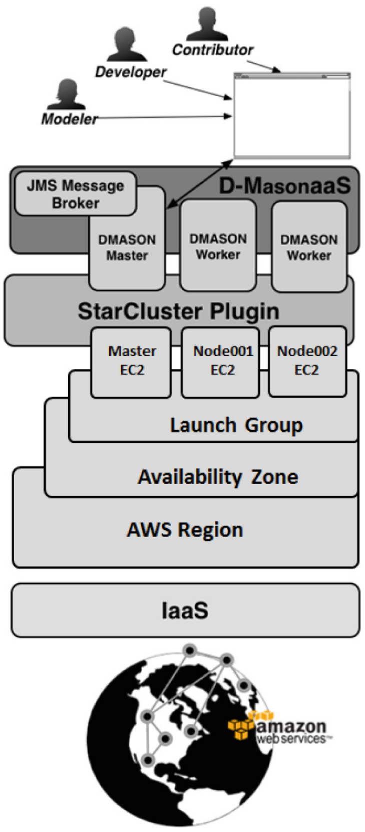


Fig. 2. D-Mason on the cloud: architecture.

The master node runs the D-Mason Master application, the JMS message broker (ActiveMQ) and the web system management server (see Sect. 3.1). The other machines run the D-Mason Worker applications, which communicate using the JMS message broker running on the Master node. Each D-Mason Worker application provides a simulation slot for each core available on the machine. The StarCluster D-Mason Plugin is freely available on GitHub D-Mason source code repository [14].

The D-Mason tier did not require any particular change but, since now the system will be executed on a cloud environment, a novel Web system management interface has been developed in order to manage the system.

3.1 D-Mason Web System Management

The former version of D-Mason system management was introduced in [2]. Briefly, it is a console written in Java, using *Java swing* framework, for managing and monitoring D-Mason simulations. In details, D-Mason system management enables to:

- configure a simulation (choose the simulation and its parameters, define the partitioning and the communication strategies);
- select a set of workers to be used as LPs;
- manage the simulation execution (play/pause/stop);
- collect the simulations' logs and the outputs.

The former version of D-Mason system management had two disadvantages: First, it was not fully decoupled from the simulation part. Hence, adding new features often requires complex interventions with a considerable waste of time. Moreover, the system was designed for local interactions (that is assuming that both the simulation and the management applications are reachable on several IP ports). Unfortunately, this is not always the case, both NAT and firewall services may result in unreachable ports. For the reasoning above, we decided to develop a fully decoupled system management services easily available via web services.

Design. We decided to embed a portable web server into our architecture. After a deep analysis of the open web servers available for Java, we decided to opt for Jetty [18]. In order to develop an efficient and pleasant interface, we were inspired by Google material design [16], the guidelines provided by Google for the development of good design interfaces. Our interface is based on a useful library named Polymer [17], which has been designed to create components for the modern web, following the material design guideline.

Architecture. The novel web server components has been encapsulated into the D-Mason Master application, which now comprises two communication components:

- ActiveMQ, for communication between D-Mason applications (either master-worker or worker-worker)
- Jetty, for communication between the user and the master application (via web interface)

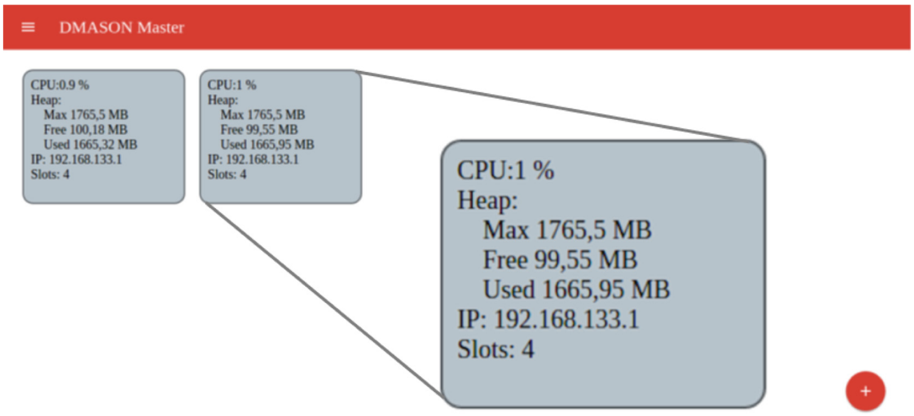
When the user starts the Master application, both the ActiveMQ and the Jetty server will run on the host. In particular the Jetty server is reachable on a TCP port (default is 8080) and the user can access the management console via browser. Using this approach the user can manage and monitor its simulation, provided that the port 8080 of the Master node is reachable on the Internet.

We posit that the load of the Jetty Server will have no impact on the overall performances of the system. This is true especially when the number of users is small and the user interaction is limited. Indeed, the load of the Jetty server is only due to the activity of discovering and monitoring of LPs. In any case, when this load increases (i.e., a huge number of users continuously interacting with the master and/or the number of LPs to be monitored is large) the master node can be configured to use an external ActiveMQ communication server.

A dedicated hand-shaking mechanism allows bonding the Master application with the workers available. When a worker joins the system, it communicates how many slots (LPs) it can afford. As soon as the master realizes that he has enough LPs to start the simulation, the system enables the user to interact with the simulation.

The web system management enables also the user to monitor the resources available on all connected workers (see Fig. 3). Using such information the user is able to choose appropriately the workers to be engaged for future simulations.

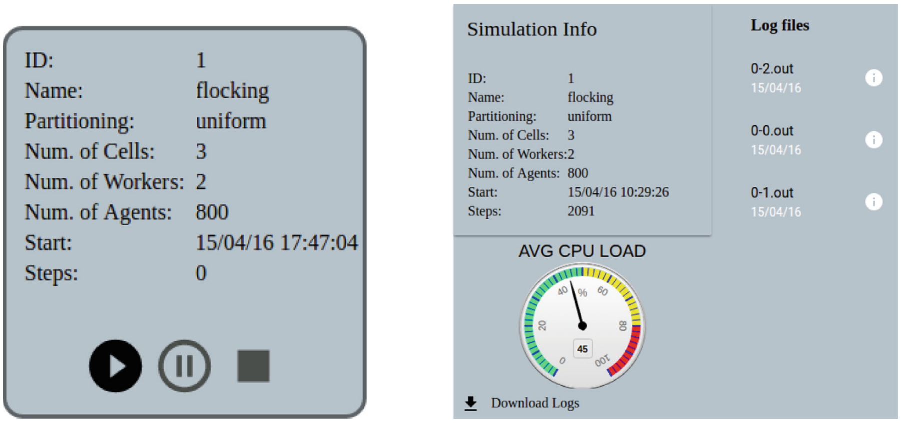
The system management provides a library of preloaded simulation but at the same time, it is possible to upload a novel simulation as a jar file. Once a simulation has been chosen, the user has to select the simulation's parameters and submit it to the selected workers. The simulations page shows the list of all the simulations running on the system; for each simulation, using the *Simulation Controller* (see Fig. 4 (left)), the user can start, pause or stop the execution until the end of the simulation. In order to monitor the evolution of a simulation, a logging mechanism has been implemented. All the log files are available at run-time on the Simulation Info panel (see Fig. 4 (right)). Moreover, a history page is available in order to get all the information available about executed simulations. The history page allows also downloading log files.



CPU:0.9 %
Heap:
Max 1765,5 MB
Free 100,18 MB
Used 1665,32 MB
IP: 192.168.133.1
Slots: 4

CPU:1 %
Heap:
Max 1765,5 MB
Free 99,55 MB
Used 1665,95 MB
IP: 192.168.133.1
Slots: 4

Fig. 3. Workers seen from master


ID:	1
Name:	flocking
Partitioning:	uniform
Num. of Cells:	3
Num. of Workers:	2
Num. of Agents:	800
Start:	15/04/16 17:47:04
Steps:	0

Simulation Info

ID:	1
Name:	flocking
Partitioning:	uniform
Num. of Cells:	3
Num. of Workers:	2
Num. of Agents:	800
Start:	15/04/16 10:29:26
Steps:	2091

Log files

0-2.out 15/04/16
0-0.out 15/04/16
0-1.out 15/04/16

Fig. 4. Simulation Controller (left) and Simulation Info (right)

4 Performance and Cost Evaluation

We performed several benchmarks in order to evaluate the performance of D-Mason on the cloud. All the tests have been performed on *D-Flockers*, which is the distributed version of *Flockers* Mason model and represents an implementation of the Boids model [12]. Boids/Agents have been simulated on a 2D geometric field having size 6400×6400. For each test we executed a reproducible simulation with 1 million agents for 15 min. At the end of the simulation, we computed the number of simulation steps performed. We used the novel system management, to start and stop the simulation and to collect the log files.

Five 2D space partitioning strategies (2×2, 2×4, 3×4, 4×4, 4×5) which generate respectively 4, 8, 12, 16 and 20 cells have been considered. All the simulations have been performed with a number of LPs (cores) equal to

Table 1. Cost calculation for in-house hosting of a single server with 8 Xeon 2-cores processors.

Cost factor	Value	Calculated cost
Hardware purchase	\$6500	
Amortization - number of months	36	
Monthly server hardware cost	\$200	
Average number of hours in month	730	
Server usage %	50%	
Average number of effective hours in month	365 h	
Hardware cost for effective hour		\$0.49\$
Power consumption full load	500 W	
Power consumption stand by	200 W	
Power management unit (PMU)	2.5	
Server usage %	50%	
Average hourly consumption	$350 \times 2.5 = 875$ W	
Electricity price per KWh	\$0.13	
Electricity cost for effective hour		\$0.11
Rack space	\$30/month	
UPS	\$20/month	
Internet connection	\$20/month	
Collocation effective hour		\$0.19
Human hardware maintenance	\$200/server × month	
Managing per effective hour		\$0.55
Total effective costs per server hour		\$1.34
Number of CPUs	16	
Total effective costs per CPU		\$0.08

the number of cells described by the partitioning strategy on four instance type (either cloud or HPC). Specifically we tested two cloud instances available on Amazon EC2:

c3.large, processor Intel Xeon E5-2680 v2 (Ivy Bridge) with 2 vCPU, 3.75 GB of memory and 2×16 GB SSD storage (cost $0.105/h — or 0.019/h for spot at the low price range);

c3.xlarge, processor Intel Xeon E5-2680 v2 (Ivy Bridge) with 4 vCPU, 7.5 GB of memory and 2×40 GB SSD storage. (cost $0.210/h — or 0.039/h for spot at the low price range).

In order to compare the results against a dedicated on–site environment, we performed the same tests on an HPC cluster. The HPC cluster consists of 16 nodes – each one equipped with 2 × Intel(R) Xeon(R) CPU E5-2430 with 12 vCPU, 16 GB of memory and 1 TB HDD storage – interconnected through a Gigabit Ethernet. Each node is running Ubuntu 14.04 operating system with latest updates. The (per node) cost of the considered HPC environment is reported in Table 1.

We considered two different configurations. In the former one, named **HPC1**, all the LPs are executed using a single node, while in the latter, named **HPC***,

Table 2. Performance and costs comparison.

Instance type	# of instances	# of LPs	Partitioning	Performed steps in 15 min (Avg)	Overall cost cost	Overall EC2 spot	Cost (x step) $/1000
c3.large	2	4	2 × 2	110	$0.210/h	$0.038/h	0.48
c3.large	4	8	2 × 4	271	$0.420/h	$0.076/h	0.39
c3.large	6	12	3 × 4	408	$0.630/h	$0.114/h	0.39
c3.large	8	16	4 × 4	601	$0.840/h	$0.152/h	0.35
c3.large	10	20	4 × 5	846	$1.05/h	$0.19/h	0.31
c3.xlarge	1	4	2 × 2	139	$0.210/h	$0.038/h	0.38
c3.xlarge	2	8	2 × 4	325	$0.420/h	$0.076/h	0.32
c3.xlarge	3	12	3 × 4	555	$0.630/h	$0.114/h	0.28
c3.xlarge	4	16	4 × 4	598	$0.840/h	$0.152/h	0.35
c3.xlarge	5	20	4 × 5	955	$1.05/h	$0.19/h	0.27
HPC1	1	4	2 × 2	245	$1.34/h	N/A	1.37
HPC1	1	8	2 × 4	336	$1.34/h	N/A	1
HPC1	1	12	3 × 4	375	$1.34/h	N/A	0.89
HPC1	1	16	4 × 4	387	$1.34/h	N/A	0.87
HPC1	1	20	4 × 5	389	$1.34/h	N/A	0.86
HPC*	2	4	2 × 2	326	$2.68/h	N/A	2.05
HPC*	4	8	2 × 4	651	$5.36/h	N/A	2.06
HPC*	6	12	3 × 4	966	$8.04/h	N/A	2.08
HPC*	8	16	4 × 4	1293	$10.72/h	N/A	2.07
HPC*	10	20	4 × 5	1591	$13.4/h	N/A	2.11

we executed exactly 2 LPs for each machine. Hence in this last configuration the system uses up to 10 nodes.

We tested the four instances (**c3.large, c3.xlarge, HPC1, HPC***) with 5 partitioning configuration (20 tests overall). We notice that all the tests have been executed on a reproducible deterministic simulation using the same JVM (version 1.8.0_72). We executed each test 10 times. The results are compared using means of simulation steps performed (we observed a minimum variance in the cloud instance results, while on the HPC instances the variance was negligible). Results about performance and costs are reported in Table 2.

Analyzing the results from Table 2, we notice that D-Mason on the cloud scales pretty well. In general, we provide the following observations. The **HPC*** instance provides the best performance. This result was expected and we believe that it is mainly due to the quality of the dedicated interconnection network. It should be highlighted, however, that the **HPC*** configuration is considerably more expensive. On the other hand the cloud instances are much cheaper than the **HPC** ones. Moreover, both the cloud instances scale better than the **HPC1**, which have comparable costs. Finally, in order to measure the trade-off between performances and cost, we computed the cost (per step) of each test setting (see last column of Table 2). The results show that the cloud instances are much cheaper than dedicated instances.

5 Conclusion

The performance results described in Sect. 4 show that the proposed SIMulation-as-a-Service (SIMaaS) infrastructure provides a very attractive price-performance ratio. As a future work, it would be interesting to analyze the performance of other cloud instances also on much more demanding simulations (both in terms of computation and communication requirements).

References

1. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gen. Comput. Syst. **25**(6), 599–616 (2009). http://www.sciencedirect.com/science/article/pii/S0167739X08001957
2. Cordasco, G., Chiara, R., Fulgido, F., Vitale, M.F.: Supporting the exploratory nature of simulations in D-Mason. In: Mey, D., et al. (eds.) Euro-Par 2013. LNCS, vol. 8374, pp. 555–564. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54420-0_54
3. Cordasco, G., Chiara, R., Mancuso, A., Mazzeo, D., Scarano, V., Spagnuolo, C.: A framework for distributing agent-based simulations. In: Alexander, M., et al. (eds.) Euro-Par 2011. LNCS, vol. 7155, pp. 460–470. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29737-3_51
4. Cordasco, G., De Chiara, R., Mancuso, A., Mazzeo, D., Scarano, V., Spagnuolo, C.: Bringing together efficiency and effectiveness in distributed simulations: The experience with D-MASON. SIMULATION: Trans. Soc. Model. Simul. Int. **89**(10), 1236–1253 (2013)

5. Cordasco, G., Milone, F., Spagnuolo, C., Vicidomini, L.: Exploiting D-Mason on parallel platforms: a novel communication strategy. In: Lopes, L., et al. (eds.) Euro-Par 2014. LNCS, vol. 8805, pp. 407–417. Springer, Cham (2014). doi:10.1007/978-3-319-14325-5_35
6. Cosenza, B., Cordasco, G., De Chiara, R., Scarano, V.: Distributed load balancing for parallel agent-based simulations. In: Proceedings of the 19th International Euromicro Conference on Parallel, Distributed, and Network-Based Processing, (PDP 2011), pp. 62–69 (2011)
7. D'Angelo, G., Marzolla, M.: New trends in parallel and distributed simulation: from many-cores to cloud computing. Simul. Model. Pract. Theory **49**, 320–335 (2014). http://www.sciencedirect.com/science/article/pii/S1569190X14001014
8. Fujimoto, R., Malik, A., Park, A.: Parallel and distributed simulation in the cloud. Int. Simul. Mag. Soc. Model. Simul. **3**(1) (2010)
9. López-Paredes, A., Edmonds, B., Klugl, F.: Editorial of the special issue: agent based simulation of complex social systems. SIMULATION: Trans. Soc. Model. Simul. Int. **88**(1), 4–6 (2012)
10. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K.: MASON: a new multi-agent simulation toolkit. In: Proceedings of the 2004 SwarmFest Workshop (2004)
11. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: MASON: a multiagent simulation environment. Simulation **81**(7), 517–527 (2005). http://dx.doi.org/10.1177/0037549705058073
12. Reynolds, C.W.: Flocks, herds and schools: a distributed behavioral model. SIGGRAPH Comput. Graph. **21**(4), 25–34 (1987). http://doi.acm.org/10.1145/37402.37406
13. Amazon EC2. https://aws.amazon.com/ec2
14. D-MASON Official GitHub Repository. https://github.com/isislab-unisa/dmason. Accessed May 2016
15. StarCluster. http://star.mit.edu/cluster/index.html
16. Google Material Design. https://www.google.com/design/spec/material-design
17. Polymer. https://www.polymer-project.org/1.0/
18. Jetty. http://www.eclipse.org/jetty/