Leveraging EGI Federated Cloud Infrastructure for Hadoop analytics

J. López Cacheiro[1], A. Simón[1], J. Villasuso[1], E. Freire[1], I. Díaz[1], C. Fernández[1] and B. Parak[2]

[1] Fundación Centro de Supercomputación de Galicia, Spain
grid-admin@cesga.es
[2] CESNET, Czech Republic
boris.parak@cesnet.cz

Abstract. FedCloud, the federated cloud infrastructure developed inside EGI, is in production since mid May 2014. In this paper we evaluate its suitability to perform Big Data analytics using Hadoop. Due to the size of the benchmarks performed, they represent also a real benchmark of the performance of FedCloud under heavy usage conditions, providing the first results about the scalability of the system.

1 Introduction

Nowadays, the possibility to run Big Data calculations over federated clouds is attracting the interest of the research community. Federation of resources poses serious challenges both from the cloud and Big Data perspectives. The aim of this paper is to evaluate the suitability of the EGI Federated Cloud infrastructure (FedCloud) to run Big Data analytics using Hadoop.

The EGI Federated Cloud Task Force [1] started its activity in September 2011 with the aim of creating a federated cloud testbed (FedCloud) to evaluate the utilization of virtualized resources inside the EGI production infrastructure. FedCloud is in production since mid May 2014 as presented at the EGI Community Forum 2014 [2], and during the last year it has been already available for experimentation by the different user communities. A more detailed description of the EGI FedCloud infrastructure is given in [3].

Big Data is an emerging field that can benefit from existing developments in cloud technologies. One of the most well-know solutions in this arena is Apache Hadoop [4], an open-source framework that implements the MapReduce programming model [5]. It provides both a distributed filesystem (HDFS), a framework for job scheduling, and a MapReduce implementation for parallel processing of large data sets.

In the MapReduce programming model users simply specify two functions: a *map* function and a *reduce* function. The *map* function processes a set of key/value pairs–usually the lines of a text or sequence file–and generates a new set of intermediate key/value pairs which are later on passed–partitioned and sorted–to the *reduce* function provided by the user merging all intermediate values associated with the same key.

The aim of our work is to do an initial assessment of the suitability of FedCloud to run Hadoop jobs through a series of real-world benchmarks. The paper is organized as follows: in Section 2 we describe the methodology used to deploy Hadoop inside the FedCloud federated cloud infrastructure; in Section 3 we include a comparison of startup times with Amazon EC2 and we present the results of the TeraGen and TeraSort benchmarks as well as the two selected use cases; finally in Section 4 we present the main conclusions of this work, summarizing the main problems encountered as well as outlining the future steps that could be taken to provide a Hadoop on-demand service in FedCloud.

2 Methodology

In this Section we describe the methodology that we have followed to deploy a Hadoop cluster instance inside the FedCloud federated cloud infrastructure.

2.1 Cluster startup

The Hadoop clusters consists of one *master* node and a variable number of *slave* nodes depending on the size of the cluster. For example in the case of a 101 node cluster, one node will be configured as master and the remaining 100 as slaves. The master node will run the *namenode*, *secondarynamenode* and *jobtracker* services and each of the slaves will run just the *tasktracker* and *datanode* services.

A customized virtual machine (VM) image was created including different versions of Hadoop (up to 1.2.1) and Oracle Java JDK (both 1.6 and 1.7). This customized image was registered in the EGI Marketplace [6] which provides a common place where where we can can store images. The Marketplace does not store the VM images into a common repository, it just shows which resource providers (RPs) have the VM image available at their endpoint. This means that each RP has to manually download the image to his local site in order to make it available at his endpoint.

To instantiate the virtual machines that will form the Hadoop cluster we used rOCCI [7], the implementation of the OCCI 1.1 specification developed by one of the authors, using VOMS proxy credentials. The new rOCCI implementation provides several advantages, like the fact that we are able to avoid the previous issue that all user certificates request are mapped to a single ONE_USER account. Additionally, the usage of VOMS over plain X509 auth greatly simplifies the access to the different sites that form FedCloud without the need to request each site to give access to our X509 certificate's DN.

FedCloud does not count with a workload management system, so each VM creation request must be sent to the appropriate endpoint, and we should take care of partitioning the cluster between the different FedCloud resource providers.

In order to increase the concurrent number of VMs, we used small size VM instances with 1GB of RAM and 1 CPU for the small and medium-size use cases and 2GB of RAM and 1 core for the TeraGen and TeraSort bechmarks.

Table 1. Tuned Hadoop configuration for small VM instances. It includes both the HDFS and MapReduce parameters used.

Parameter	Type	Value
$fs.inmemory.size.mb$	core-site	200MB
$io.file.buffer.size$	core-site	128KB
$mapreduce.task.io.sort.factor$	core-site	100
$mapreduce.task.io.sort.mb$	core-site	100
$mapred.tasktracker.map.tasks.maximum$	mapred-site	1
$mapred.tasktracker.reduce.tasks.maximum$	mapred-site	1
$mapred.reduce.tasks$	mapred-site	$0.95 \times num.\ slaves$
$dfs.datanode.du.reserved$	hdfs-site	1GB
$dfs.block.size$	hdfs-site	1MB / 64MB
$dfs.replication$	hdfs-site	3
HADOOP_HEAPSIZE	hadoop-env	512MB

The tuned configuration parameters for running the selected use cases are given in Table 1. The column type identifies the configuration file where the parameter is set, the complete configuration files can be found at [8].

These parameters have been selected to tune the Hadoop cluster to the resources available (1GB of RAM and 1 CPU per node). The *dfs.replication* parameter indicates how many copies we want to store of each block, in this case we use three replicas that will allow different nodes to execute the same MapReduce tasks–speculative execution–mitigating the performance degradation that could be experienced in a heterogeneous cluster with VMs running under hypervisors with different load conditions. The *mapred.tasktracker.map.tasks.maximum* and *reduce.tasks.maximum* are set to 1 because we only have 1 CPU available to run both services and the HADOOP_HEAPSIZE is set to 512MB to allow both running at the same time without exhausting the node's memory. The number of reduce tasks, *mapred.reduce.tasks*, is set to 95% of the number of slaves in the Hadoop cluster.

Our Hadoop installation will cope with two very different use cases that influence our decission about the *dfs.block.size* values to use. In the case of the Encyclopædia Britannica's use case the block size is set to 1MB and in the Wikipedia's use case, that uses a much larger data set, to 64MB. Under these conditions both use cases generate less than 700 total MapReduce tasks which can be handled well by the Hadoop master node which has only 1GB of RAM and 1 CPU. For the TeraGen and TeraSort becnhmarks we use an even larger block size of 128MB.

2.2 Benchmarks

In order to evaluate the suitability of FedCloud to perform Big Data analytics using Hadoop we selected the TeraGen and TeraSort benchmarks as well as two use cases representing usual small and medium-size jobs. These use cases use real data sets and a fairly common MapReduce operation.

In the Big Data arena the TeraGen and TeraSort benchmarks could be considered the equivalent of Linpack in HPC. There is even a list equivalent to the well-know Top500 list of the most powerful supercomputers in the world. This list was created by Jim Gray and it is known as the Sort Benchmarks list [9]. This list is not restricted to Hadoop clusters, and any Big Data system that is able to sort files can compete to appear there. Currently the largest sort rate is achieved by a Hadoop cluster of 2100 nodes that reaches a sort rate of 1.42TB/min [10].

TeraGen is the part of the bechmark that generates the input data set that will be used by TeraSort. It can be configured to run in parallel using the whole cluster so the throughtput it obtains is much higher than a standard HDFS put operation.

TeraSort is a standard MapReduce sort job that uses a custom partitioner with a sorted list of $N-1$ sampled keys to guarantee that the output keys of reduce n are lower than the output keys of reduce $n+1$. This allows the concatenation of all the output files generated by each Reducer to produce one ordered output file.

The selected use cases using two well-known data sets which are quite representative of a broad range of jobs that could be performed in a Hadoop cluster–most MapReduce jobs are based on the same type of operations–. The use cases were run both in a federated cluster deployment and in a one-site-only cluster deployment, allowing us to compare the results and analyze the degradation when using remotely distributed resources.

The first use case, representing small-size Hadoop jobs, is based in the Encyclopædia Britannica[11] data set–the 1911 version that is already in the public domain–. This represents a rather small data set of just 176MB, so that this use case can be considered as a small scale benchmark that will serve to evaluate the expected degradation in performance due to fact that we will be using federated resources located at different sites. The data set is downloaded directly from the master node and later on it is distributed from there to the slaves, what is called a *put* operation in Hadoop argon. This use case has a great overload due to the creation of rather small MapReduce tasks that will stress the communication system because it uses a rather small block size of just 1MB, so each map operation has little to process.

The second use case, representing medium-size Hadoop jobs, is based in the Wikipedia[12] data set. This represents a much larger data set comprising 41GB, so it can be considered a more realistic benchmark of a medium-size Hadoop job. Due to the size of the data set the files could not be downloaded directly in the master node so the *put* operation is done directly from our local Hadoop client. This use case will incur in a lower task creation penalization because it uses a larger block size of 64MB, so each map task will take much longer than the time needed for its creation.

Both use cases were run for different cluster sizes ranging from 10 to 101. In all the executions we measured two parameters:

1. *The put time:* the time to load the files into the Hadoop HDFS filesystem
2. *The wordcount time:* the time to perform a simple wordcount MapReduce job that computes the number of occurrences of each word over the complete data set.

All the measurements where repeated at least 5 times in order to evaluate the variability of the results, the only exception were the *put* times for the Wikipedia that due to its duration could only be run once.

3 Results

Following the methodology described in previous Section we deployed Hadoop in FedCloud and executed the benchmarks and use cases mentioned in Subsection 2.2.

In Subsection 3.1 we present the results related to the cluster startup process. In this case we were dealing with the instantiation of a number of VMs ranging from 10 to 101, so it can serve also as a stress test of both rOCCI and the underlying cloud management backends.

In Section 3.2 we present the benchmark results both for the two selected use cases–Encyclopædia Britannica and Wikipedia–and for the TeraGen and TeraSort benchmarks.

3.1 Cluster Startup: Comparison with Amazon EC2

The startup times for each cluster deployment varied, being the most representative ones those obtained for the larger deployment: the 101-node cluster. In this case, the time needed by the rOCCI client to return all the resource endpoints corresponding to each VM ranged from 71 to 86 minutes. This is just the time to get the identifier corresponding to each VM instance, actually to have all the VMs running took longer: around 80 minutes more. So the total cluster startup time ranged from 2.5 to 3 hours.

Some of the rOCCI requests failed with an *execution expired* error and some of the VMs did not start. In the worst case 21 out of 101 failed. This type of errors where observed when deploying more than 20 VMs at CESGA. This analysis could not be done in CESNET because there were only 10 VM slots available at the time of the benchmarks, due to a restriction in the amount of public IPs that they had available.

Trying to understand the source of this issue the performance of the OpenNebula frontend was tracked during the cluster startup. As it can be seen in Figure 1 the frontend experiences a high load several minutes after the start of the cluster deployment, this causes it to respond slower, producing the rOCCI execution expiration errors mentioned above. Looking at the processes in the OpenNebula frontend, we could see that this increase in the load is mainly due to the *scp* processes launched to copy the image template to the OpenNebula nodes, more than 20 simultaneous scp processes seem to affect considerably the system performance, reducing its response times considerably. The copy process is also limited by the fact that the frontend has only 1 Gbps connectivity, acting as a bottleneck for the image delivery process.

In order to optimize the startup times we reduced the size of the image to 4GB. But this configuration would not leave enough space for applications, to solve

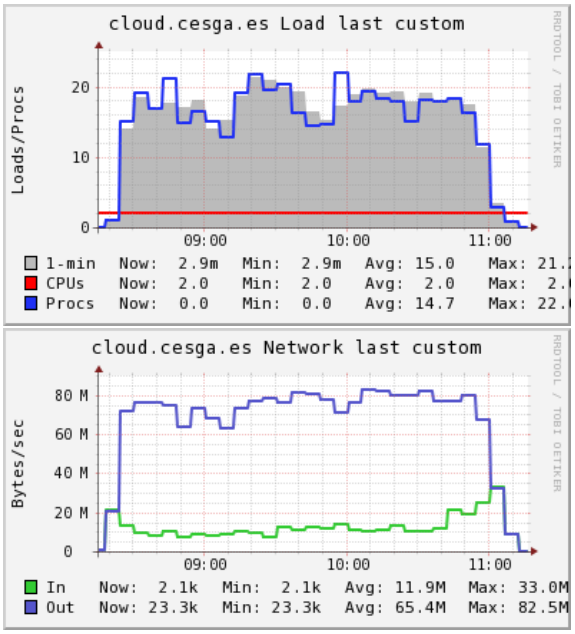

Fig. 1. OpenNebula frontend load and network usage graphs taken from Ganglia during the deployment of the 101 nodes cluster.

this issue an additional disk of 70GB was created on-the-fly using a *fs* disk type in OpenNebula and an *ephemeral* disk in OpenStack. Furthermore, we modified the *datastore* used in OpenNebula, so that the transfer manager (TM) used is *shared* instead of *ssh*. Since the instances are declared as non-persistent, this allows to perform a powerful optimization in the instance deployment process done by OpenNebula because the image copy process will be no longer done in a centralized way from the front-end node but in parallel from each node using a simple copy operation from the shared storage. The optimized startup times can be found in Table 2.

Table 2. Optimized startup times at CESGA after tuning OpenNebula configuration.

Cluster size	Startup time (s)
10	249
21	269
51	822
101	1096

With the aim of comparing the startup times obtained with those of other commercial clouds we performed performed similar measurements of cluster startup times in Amazon EC2, this way we we can have a better understanding of how FedCloud performs compared to commercial clouds.

Amazon Elastic Compute Cloud (Amazon EC2) is a very popular commercial cloud platform that provides resizable compute capacity in a pay-per-use model. Even if the infrastructure behind Amazon EC2 is much larger than the one behind FedCloud, notice should be given to the fact that to launch more than 20 instances in Amazon EC2 you have to fill a form indicating the number of instances you need and their expected usage. For some users, especially commercial users, this requirement may be unnacceptable. This request is not validated automatically so you have to wait several hours until it is validated.

There are different tools available to create clusters in Amazon EC2, being Apache Whirr [13] one of the most popular. The Apache Whirr project started as a set of bash scripts distributed with Apache Hadoop for running Hadoop clusters on Amazon EC2, but it evolved to a Java version based on Apache jclouds [14], an open source multi-cloud toolkit that allows Whirr to support many different cloud providers. Unfortunately Whirr does not scale well due to the large number of REST requests it sends to the Amazon API that causes Amazon to temporarily block the client with a *Request limit exceeded* error. The largest cluster we were able to launch in the eu-west-1 region was 21 nodes (see Table 3).

To avoid this issue a custom tool [15] to create and configure the Hadoop cluster was developed. This tool used Amazon EC2 *instance-count* functionality to request all the instances in the same REST request avoiding the *Request limit exceeded* problem. The results are shown in Table 4. It can be seen that all the startup times are below 5 minutes. It is especially relevant the fact that both in the 51 and 101 clusters there where nodes that could not be reached for different reasons. In some cases the instance launch directly failed and the error was reported by Amazon. However, in most cases the failing instances were reported as ready, but they were not reachable from the other nodes in the cluster. The reason seems to be a security group internal network issue at Amazon.

Table 3. Amazon EC2 startup times using Whirr: the AMI and instance type used are also specified.

Cluster size	AMI	Instance type	Startup time (s)
10	eu-west-1/ami-c37474b7 EBS	t1.micro	555
10	eu-west-1/ami-c37474b7 EBS	m1.small	891
21	eu-west-1/ami-c37474b7 EBS	m1.small	1670
31	eu-west-1/ami-c37474b7 EBS	m1.small	Fails

Table 4. Amazon EC2 startup times using our custom tool: the AMI used was *eu-west-1/ami-54d43023* EBS plus instance store. The instance type used was *m1.small* in all cases.

Cluster size	Startup time (s)	Comments
10	191	
21	178	
51	583*	2 nodes not working
101	276*	16 nodes not working
101	297*	1 node not working

3.2 Benchmarks

In Table 5 we show the results obtained for the TeraGen and TeraSort benchmarks. In this case we can generate much larger data sets than the Encyclopædia Britannica and Wikipedia data sets, so we start at 50GB and go up to 2TB. Due to the larger size of the data sets the $dfs.blocksize$ was increased to $128MB$. We can see the scaling is good but after reaching 600GB the TeraSort benchmark fails because there is no enough space–total storage available is 2.65GB–to store the intermediate data. In order to evaluate larger data sets we reduced the number of HDFS replicas to one–reducing them to two would not provide much additonal margin–and rerun the whole set of benchmarks. In this way, we could reach 2TB in the Teragen benchmark. Unfortunately having just one replica eliminates the fault tolerance of the system, making the system much less reliable, and causing the larger TeraSort benchmarks to consistently fail.

The best results were obtained using 100 mappers in TeraGen and 100 reducers in TeraSort (the number of mappers is determined by the number of blocks).

We saw the importance of using a number of parallel tasks higher than the number of nodes so we can take advantadge of Hadoop's speculative execution. This is something critical in a cloud environment where the performance of the instances is very heterogeneous, depending greatly on the site where it runs and the activity of the other instances that share the same host.

Rack awareness was used take into account the node's physical location. Each instance's network gateway is used to define the datacenter part of the *datanode rack id*: */¡GW¿/default-rack*. This allows to define the topology automatically in a federated environment–taking into account local rack location is not feasible in this environment.

The MapOutput in-memory buffer was also especifically tuned for TeraSort using $io.sort.mb = 150$, $io.sort.record.percent = 0.138$ and $io.sort.spill.percent = 1.0$. This configuration avoids the expensive cost of *spills*–spilling to disk more than once would triple the disk I/O required.

Additionally to the previous large-scale benchmarks we also evaluated some real-world examples of medium and small Hadoop jobs. These use cases are, from

Table 5. Results obtained for the TeraGen and TeraSort benchmarks: times where measured in a federated cluster that included 20 slave nodes running at CESGA and 20 at CESNET with the master node at CESGA.

Dataset size	replicas	teragen (s)	terasort (s)
50GB	3	1342	750
100GB	3	2510	1504
200GB	3	4575	3278
300GB	3	5934	5766
400GB	3	7771	7244
600GB	3	11595	Failed
800GB	3	15351	Failed
50GB	1	86	820
100GB	1	140	1506
200GB	1	272	3472
300GB	1	379	5529
400GB	1	538	7679
500GB	1	624	10094
600GB	1	715	Failed
800GB	1	1014	Failed

our experience, more representative of the typical needs of most users, so they will help to evaluate the suitability of FedCloud for these tasks.

In Table 6 we show the results obtained for the Encyclopædia Britannica use case. All measurements were repeated 5 times and the average and standard deviation are displayed, allowing to have an estimation of the variability involved in the measurement. As it can be seen even if the block size used in this case is very small (1MB), the wordcount map reduce job scales really well and the overhead introduced by the federated cluster is small, being in general in order of 10-15%, and 28% in the worst case that corresponds to the 101 cluster. This is mainly due to the large overhead that MapReduce task creation and distribution imposes in this use case, especially in the larger cluster sizes that involve more reduce tasks (we are using a number of reduce tasks equal to the 95% of the total number of slaves).

As expected the *put* times are much lower in one-site-only scenario because in this case the connection between all the VMs is much faster because all of them are generally located in the same data centre.

In Table 7 we show the results obtained for the Wikipedia use case. It should be noted that due to the high duration of the transfers, we did not repeat the *put* measurements 5 times, and only the wordcount measurements were repeated. Also due to the longer duration of the benchmarks we only used the two larger cluster sizes: 51 and 101.

In this case the wordcount mapreduce results are quite surprising because the federated cluster instance is able to outperform the one-site-only instance in both

Table 6. Benchmark times obtained for the Encyclopædia Britannica use case. Both the average time and standard deviation are reported. Federated times where measured in a federated cluster that included resources both from CESGA and CESNET; all one-site-only times where obtained in a one site deployment at CESGA except the 10 node cluster indicated in the second row that run at CESNET.

	Federated		One-site-only	
Cluster size	put (s)	wordcount (s)	put (s)	wordcount (s)
10 (CESGA)			47 ± 2	169 ± 1
10 (CESNET)			9.5 ± 0.2	160 ± 1
21	235 ± 12	108 ± 2	37 ± 1	97 ± 1
31	189 ± 4	90 ± 2	36 ± 2	78 ± 2
41	190 ± 11	81 ± 4	33 ± 2	71 ± 3
51	133 ± 7	73 ± 3	33 ± 1	66 ± 2
101	94 ± 7	67 ± 22	33 ± 4	52 ± 5

scenarios. This is attributed mainly to the faster execution of the benchmark in CESNET nodes compared to CESGA nodes (CESNET VMs use Xen paravirtualization and a faster CPU whereas CESGA VMs use KVM and QEMU). In this case the relative overhead due to MapReduce task creation is much lower than in the previous use case because we are using a much larger block size of 64MB.

The *put* times are much larger in the federated cluster due to the fact that the amount of data to transfer is much larger: HDFS stores three copies of the Wikipedia, 42GB each, so that the available bandwidth between the VMs and the UI located at CESGA plays a key role, being much lower in the case of CESNET VMs.

Table 7. Benchmark times obtained for the Wikipedia use case. Both the average time and standard deviation are reported except for the put times that where only measured once for each deployment. Federated times where measured in a federated cluster that included resources both from CESGA and CESNET; all one-site-only times where obtained in a one site deployment at CESGA.

	Federated		One-site-only	
Cluster size	put (s)	wordcount (s)	put (s)	wordcount (s)
51	19190	1001 ± 39	6705	1347 ± 117
101	13208	705 ± 14	5665	725 ± 18

4 Conclusions

The results presented in this paper serve as an initial evaluation of the performance of Apache Hadoop running on the EGI federated cloud infrastructure. They show that FedCloud is especially suitable for small and medium-size Hadoop jobs where the data set is already pre-deployed in the Hadoop HDFS filesystem. Considering the two use cases selected, we see that the federated cluster is even able to outperform the local one if the remote nodes are faster than the ones available locally.

This results show that small VM instances are good enough to run Hadoop workloads assuming an appropriate tuning of the configuration is done, of course using larger instances would simplify the configuration and it would allow to reach larger problems. In our benchmarks the *tasktrackers* run out of memory when running the Wikipedia's use case if we used a small number of reduce tasks, additionally when using a small *dfs.block.size* it was the *jobtracker* the service that run out of memory because it was not able to cope with the large amount of tasks.

The initial results showed very disappointing results about the startup times and reliability of the system. Almost 3 hours were needed to start the larger cluster with 101 nodes, a high penalty for small and medium size jobs that usually finish in less than an hour. Additionally, failures started to appear when trying to start more than 20 VMs. After analyzing in detail the startup process we saw these failures could be mainly attributed to limitations in the resource provider cloud management backend. We were able to appropriately tune both the cloud provider backend and our marketplace image, reducing the startup time to less than 20 minutes for the 101-node cluster. This optimization also lead to an improvement of the reliability of the startup process, so no more instances failed during the cluster startup process.

The startup times were compared with those of Amazon EC2. In order to run the measurements, a custom tool had to be developed for deployment of cluster larger than 20 nodes because of the request rate limit imposed by Amazon that causes Whirr to fail in such scenarios. Our measurements showed startup times of around 5 minutes for the larger 101-node cluster, but there were reliability issues in Amazon's infrastructure that caused several nodes not to work properly.

There is also a large overhead introduced by the upload–*put* times–of the data sets to the HDFS filesystem. In the medium-size use case it takes 10 times more to upload the data set than to run the actual job. This aspect should be taken into account when considering Hadoop on demand services.

Considering the scalability of the system we were able to run the TeraGen benchmark up to 2TB and the TeraSort benchmark up to 500GB. Even if larger targets could be reached increasing the size of the system, we found networking issues when trying to add additional instances running at additional sites. These issues seemed site-specific.

There are some aspects that could improve the usability of FedCloud like adding a central workload management system, an automated way to distribute and synchronize the image between all sites, or a mechanism to query the resources available at a given site.

Acknowledgements

We would like to thank BIFI, IFCA, CESNET and CESGA for providing the resources for the benchmarks as well as for their support during all this work.

This work is partially funded by EGI-InSPIRE (European Grid Initiative: Integrated Sustainable Pan-European Infrastructure for Researchers in Europe), a project co-funded by the European Commission (contract number INFSO-RI-261323) as an Integrated Infrastructure Initiative within the 7th Framework Programme. Full information is available at: `http://www.egi.eu/`.

References

[1] FedCloud: EGI Federated Cloud Task Force. `https://wiki.egi.eu/wiki/Fedcloud-tf:FederatedCloudsTaskForce` (June 2014)
[2] FedCloud: EGI Federated Cloud. `https://wiki.egi.eu/wiki/Fedcloud-tf:Main` (June 2014)
[3] Simón, A., Freire, E., Rosende, R., Díaz, I., Feijóo, A., Rey, P., López-Cacheiro, J., Fernández, C.: Egi fedcloud task force. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on. (2012) –
[4] Hadoop: Apache hadoop. `http://hadoop.apache.org/` (June 2014)
[5] Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. In: Sixth Symposium on Operating System Design and Implementation. (2004) –
[6] EGI: EGI Marketplace. `http://marketplace.egi.eu` (June 2014)
[7] EGI: rOCCI API. `https://github.com/gwdg/rOCCI` (June 2014)
[8] EGI: Hadoop on fedcloud deployment tools. `https://github.com/grid-admin/hadoop` (June 2014)
[9] Gray, J.: Sort benchmark. `http://sortbenchmark.org` (June 2014)
[10] Graves, T.: Graysort and minutesort at yahoo on hadoop 0.23. `http://sortbenchmark.org/Yahoo2013Sort.pdf` (June 2014)
[11] Britannica: Encyclopædia Britannica. `http://www.britannica.com` (June 2014)
[12] Wikipedia: Wikipedia. `http://www.wikipedia.org` (June 2014)
[13] Apache: Apache whirr. `https://whirr.apache.org` (June 2014)
[14] Apache: Apache jclouds. `http://jclouds.apache.org` (June 2014)
[15] López Cacheiro, J.: Hadoop on-demand. `https://github.com/alcachi/hadoop-on-demand` (June 2014)