TPCx-HS v2: Transforming with Technology Changes

Tariq Magdon-Ismail[1(✉)], Chinmayi Narasimhadevara[2], Dave Jaffe[1], and Raghunath Nambiar[2]

[1] VMware, Inc., 4301 Hillview Ave, Palo Alto, CA 94304, USA
{tariq,djaffe}@vmware.com
[2] Cisco Systems, Inc., 275 East Tasman Drive, San Jose, CA 95134, USA
{cnarasim,rnambiar}@cisco.com

Abstract. The TPCx-HS Hadoop benchmark has helped drive competition in the Big Data marketplace and has proven to be a successful industry standard benchmark for Hadoop systems. However, the Big Data landscape has rapidly changed since its initial release in 2014. Key technologies have matured, while new ones have risen to prominence in an effort to keep pace with the exponential expansion of datasets. For example, Hadoop has undergone a much-needed upgrade to the way that scheduling, resource management, and execution occur in Hadoop, while Apache Spark has risen to be the de facto standard for in-memory cluster compute for ETL, Machine Learning, and Data Science Workloads. Moreover, enterprises are increasingly considering cloud infrastructure for Big Data processing. What has not changed since TPCx-HS was first released is the need for a straightforward, industry standard way in which these current technologies and architectures can be evaluated. In this paper, we introduce TPCx-HS v2 that is designed to address these changes in the Big Data technology landscape and stress both the hardware and software stacks including the execution engine (MapReduce or Spark) and Hadoop Filesystem API compatible layers for both on-premise and cloud deployments.

Keywords: TPC · Big Data · Benchmark · Hadoop · Spark
Cloud · Performance

1 Introduction

Since its release on August 2014, the TPCx-HS Hadoop benchmark [1] has helped drive competition in the Big Data marketplace, generating 24 publications spanning 5 Hadoop distributions, 3 hardware vendors, 2 OS distributions, and 1 virtualization platform [2] (as of 2017/06/20). By all measures, it has proven to be a successful industry standard benchmark for Hadoop systems. However, the Big Data landscape has rapidly changed over the last three years. Key technologies have matured, while new ones have risen to prominence in an effort to keep pace with the exponential expansion of datasets. Moreover, enterprises are increasingly considering cloud infrastructure for Big Data processing. What has not changed, however, is the need for a *straightforward*, industry standard way in which these current technologies and architectures can be evaluated with workloads and metrics that are well understood and easily relatable to the end user.

R. Nambiar and M. Poess (Eds.): TPCTC 2017, LNCS 10661, pp. 120–130, 2018.
https://doi.org/10.1007/978-3-319-72401-0_9

In keeping with these important industry trends, we introduce TPCx-HS v2 for Hadoop and Spark that support not only traditional on-premise deployments but also cutting edge cloud deployments.

The rest of the paper is organized as follows. Section 2 briefly discusses the Hadoop ecosystem and the emergence of Spark. In Sect. 3, we present the changes made to TPCx-HS in version 2 of the benchmark specification and kit. Section 4 follows with an experimental comparison of Hadoop MapReduce and Spark. Finally, we conclude in Sect. 5.

2 Hadoop Ecosystem

At its core, Apache Hadoop is a software library that provides a framework for distributed processing of large datasets using a simple programming model. The popularity of Hadoop has grown in the last few years because it meets the needs of many organizations for flexible data analysis. Because of the increased deployment of Hadoop in production, a rich ecosystem of tools and solutions has developed around it. The number of official Apache open source projects alone, that are related to Hadoop, have increased from just 1 in 2008 [5] to 26 today [6]. Commercial Hadoop offerings are even more prolific and diverse, and include platforms and packaged distributions from vendors such as Cloudera, Hortonworks, and MapR, plus a variety of tools for specific Hadoop development, production, and maintenance tasks. Today, Apache Spark represents an increasingly important piece of this ecosystem.

2.1 Emergence of Spark

Apache Spark is an open source cluster computing framework that provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance. It was developed to overcome some of the bottlenecks of Apache Hadoop, one of which is around the use of intermediate persistent storage. Spark provides an alternative to MapReduce that enables workloads to execute in memory, instead of on disk. Spark accesses data from HDFS but bypasses the MapReduce processing framework, and thus eliminates the resource-intensive disk operations that MapReduce requires. By using in-memory computing, Spark workloads typically run significantly faster compared to disk execution.

According to a Big Data survey report published by the Taneja Group [3], performance was cited as one of the main drivers of Spark adoption. Within the report, more than half of the respondents mentioned actively using Spark, with a notable increase in usage over the twelve months following the survey. Clearly, Spark is an important component of any Big Data pipeline today. Interestingly, but not surprisingly, there is also a significant trend towards deploying Spark in the cloud.

3 TPCx-HS v2

The TPCx-HS benchmark now stresses both the hardware and software stack including the execution engine (MapReduce or Spark) and Hadoop Filesystem API compatible

layers for both on-premise and cloud deployments. The workload can be used to assess a broad range of system topologies and implementations of Hadoop/Spark clusters. In this new version of the kit, there have been changes made to support not only Spark but also Hadoop 2 APIs. The following sections discuss the need for these changes and describe what they are.

3.1 Hadoop 2 Support

With Hadoop 2, MapReduce from Hadoop 1 (MRv1) has been split into two components. The cluster resource management capabilities have become YARN (Yet Another Resource Negotiator) [11], while the MapReduce-specific capabilities remain MapReduce (MRv2)—albeit with a newer API. This is a significant upgrade to the way scheduling, resource management, and execution occur in Hadoop. It divides resource management and job lifecycle management into separate components.

The new YARN ResourceManager manages the global assignment of compute resources to applications, and the per-application ApplicationMaster manages the scheduling and coordination of an application. An application is either a single job (in the sense of Hadoop 1 MapReduce jobs) or a Directed Acyclic Graph (DAG) of such jobs. The ResourceManager and per-machine NodeManager daemon, which manages the user processes on that machine, form the computation unit of the job. The per-application ApplicationMaster is the framework-specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks. One of the primary issues with MRv1 is that the Map and Reduce slot configuration is static. This inflexibility can lead to the underutilization of resources [10]. There is no slot configuration in YARN, allowing it to be more dynamic and hence more efficient [11]. Another limitation of MRv1 is that the Hadoop framework only supports MapReduce jobs. YARN supports both MapReduce and non-MapReduce applications.

3.1.1 MapReduce Kit Changes

While TPCx-HS v1 used the MRv1 API, the MapReduce code in the TPCx-HS v2 kit has been rewritten to conform to the MRv2 Java API. Since the MRv2 API is not backward compatible, a side effect of this change is that TPCx-HS v2 will not run on Hadoop 1. As before, job configuration options can be specified on the command line or in the mapred-site.xml file on the client. The vast majority of job configuration options that were available in MRv1 work in MRv2/YARN as well. For consistency and clarity, many options have been given new names. The older names are deprecated, but will still work for the time being. One more notable difference is the change in record format between TPCx-HS v1 and v2. The MRv1 code used a 64-bit Linear Congruential Generator (LCG) based random number generator, while the new MRv2 code uses a 128-bit LCG random number generator. As a result, keys now remain in the binary format. TPCx-HS v2 data is also less compressible as it was changed to reflect the changes in the GraySort benchmark [8] on which it is based [7]. Results of running the kit with the new API are detailed in Sect. 4.

3.2 Spark Support

Spark is a unified engine for distributed data processing. It enables batch, real-time, and advanced analytics on the Hadoop platform. Spark has a programming model similar to MapReduce but extends it with a data-sharing abstraction called "Resilient Distributed Datasets," or RDDs [9]. RDDs enable Spark to perform fault tolerant distributed in-memory computations.

Spark can be run in standalone mode or on YARN, both of which are supported by the TPCx-HS kit. In standalone mode, Spark manages its own cluster and uses a master/worker architecture. Here, a single driver (master) manages the workers on which the executors run. When run on YARN, YARN is responsible for allocating resources to Spark. Spark on YARN supports data locality for data residing in HDFS.

3.2.1 Spark Kit Changes

The TPCx-HS kit utilizes the Spark Scala API, for running the three phases of data generation, data sorting, and data validation required by the benchmark. The record generation in the Spark code is similar to the MRv2 code, so both of these results are comparable. The settings needed for Spark can be added to the Spark default configuration, for additional tuning. The kit supports running Spark using YARN or in standalone mode. The YARN configuration settings can be changed as needed for running Spark applications. The new jar file for Spark is also part of the kit and the user can choose to run with either Spark or MapReduce as the framework for running the benchmark. The results of the Spark framework tests are outlined in Sect. 4.

3.3 Cloud Support

The TPCx-HS specification now allows for cloud services to be part of the System Under Test. Moreover, the disclosure requirements have been amended to support public cloud environments where there is limited visibility into the underlying technology platform.

While there was nothing inherent in the workload or kit that prevented TPCx-HS from running on public or private cloud infrastructure, changes to the TPCx-HS specification were required in order to make the results compliant with the new TPC pricing policies outlined in the TPC Pricing Specification version 2.0 [4]. In particular, for a *measured configuration* the benchmark driver and the System Under Test must all reside in the same region and for a *priced configuration* the benchmark driver and the System Under Test must all reside in the same region. The region of the priced configuration may be different from the region of the measured configuration. The price of the priced configuration must include all hardware, software, cloud services, and maintenance charges over a period of 3 years.

4 Experimental Results

4.1 Configuration

The 13 HPE ProLiant DL 380 Gen 9 servers used in the test were configured identically, with two Intel Xeon E5-2683 v4 ("Broadwell") processors with 16 cores each and 512 GiB of memory. Hyper-Threading was enabled so each server showed 64 logical processors.

Each server was configured with two 1.2 TB spinning disks in a RAID 1 mirror for the server operating systems, as well as four 800 GB Non-Volatile Memory Express (NVMe) solid state disks connected to the PCI bus, and twelve 800 GB SAS Solid State Disks (SSDs) connected through the HPE Smart Array P840ar/2G raid controller.

Full server configuration details are shown in Table 1.

Table 1. Server configuration. In this document notation such as "GiB" refers to binary quantities such as gigibytes (2**30 or 1,073,741,824) while "GB" refers to gigabytes (10**9 or 1,000,000,000).

Component	Quantity/Type
Server	HPE ProLiant DL380 Gen 9
Processor	2× Intel Xeon CPU E5-2683 v4 @ 2.10 GHz w/16 cores each
Logical Processors (including HyperThreads)	64
Memory	512 GiB (16× 32 GiB DIMMs)
NICs	2× 1 GbE ports + 4 × 10 GbE ports
Hard Drives	2× 1.2 TB 12G SAS 10 K 2.5in HDD – RAID 1 for OS
NVMes	4× 800 GB NVMe PCIe – NodeManager traffic
SSDs	12× 800 GB 12G SAS SSD – DataNode traffic
RAID Controller	HPE Smart Array P840ar/2G Controller
Remote Access	HPE iLO Advanced

Three of the servers were virtualized with VMware vSphere 6.5 and ran virtual machines that managed the Hadoop cluster. On the first server, a VM hosted the Gateway node, running Cloudera Manager and several other Hadoop functions as well as the gateway for the Hadoop Distributed File System (HDFS), YARN, Spark, and Hive services. The second and third servers each hosted a Master VM, on which the active and passive HDFS NameNode and YARN ResourceManager components and associated services ran. ZooKeeper, running on all three VMs, provided high availability.

The other 10 servers ran only the worker services, HDFS DataNode, and YARN NodeManager. Spark executors ran on the YARN NodeManagers.

The full assignment of roles is shown in Table 2. Key software component versions are shown in Table 3.

Table 2. Hadoop/Spark roles.

Node	Roles
Gateway	Cloudera Manager, ZooKeeper Server, HDFS JournalNode, HDFS gateway, YARN gateway, Hive gateway, Spark gateway, Spark History Server, Hive Metastore Server, Hive Server2, Hive WebHCat Server, Hue Server, Oozie Server
Master1	HDFS NameNode (Active), YARN ResourceManager (Standby), ZooKeeper Server, HDFS JournalNode, HDFS Balancer, HDFS FailoverController, HDFS HttpFS, HDFS NFS gateway
Master2	HDFS NameNode (Standby), YARN ResourceManager (Active), ZooKeeper Server, HDFS JournalNode, HDFS FailoverController, YARN JobHistory Server,
Workers (10)	HDFS DataNode, YARN NodeManager, Spark Executor

Table 3. Key software components.

Component	Version
Operating System	Centos 7.3
Cloudera Distribution of Hadoop	5.10.0
Cloudera Manager	5.10.0
Hadoop	2.6.0+cdh5.10.0+2102
HDFS	2.6.0+cdh5.10.0+2102
YARN	2.6.0+cdh5.10.0+2102
MapReduce2	2.6.0+cdh5.10.0+2102
Hive	1.1.0+cdh5.10.0+859
Spark	1.6.0+cdh5.10.0+457
ZooKeeper	3.4.5+cdh5.10.0+104
Java	Oracle 1.8.0_111-b14
MySQL	5.6.35 Community Server

With the NVMe storage providing the highest random read/write IOs per second, the four NVMe devices in each server were assigned to handle the NodeManager temporary data, which consists of Hadoop map spills to disk and reduce shuffles. SAS SSDs provide very high speed sequential reads and writes, so the twelve SSDs in each server were assigned to the DataNode traffic, consisting of reads and writes of permanent HDFS data.

The Hadoop and Spark parameters used in the test are shown in Table 4. They fall into two categories. Parameters such as yarn.nodemanager.resource.cpu-vcores and yarn.nodemanager.resource.memory-mb assign the resources available to YARN (to provide to containers running Hadoop map or reduce tasks or Spark executors), while the rest are application-dependent parameters.

Table 4. Key Hadoop/Spark cluster parameters used in tests.

Parameter	Default	Configured
dfs.blocksize	128 MiB	1 GiB
dfs.replication	3	3
mapreduce.task.io.sort.mb	256 MiB	2047 MiB
yarn.nodemanager.resource.cpu-vcores		64
mapreduce.map.cpu.vcores	1	1
mapreduce.reduce.cpu.vcores	1	1
yarn.nodemanager.resource.memory-mb		448 GiB
mapreduce.map.memory.mb	1 GiB	6.25 GiB
mapreduce.reduce.memory.mb	1 GiB	6.25 GiB
Maximum map/reduce tasks per node		64/64
Number of map/reduce tasks used in tests		639/639
spark.executor.cores	1	4
spark.executor.memory	256 MiB	25 GiB
spark.driver.memory	1 GiB	10 GiB
spark.executor.instances		149
Maximum number of Spark executors per node		16
Log Level on HDFS, YARN, Hive	INFO	WARN

For yarn.nodemanager.resource.cpu-vcores, all 64 logical processors were assigned to YARN vcores. For yarn.nodemanager.resource.memory-mb, the 512 GiB server memory was reduced by about 12% to provide memory for the operating system, as well as the Java heap size required for the DataNode and NodeManager processes, resulting in 448 GiB usable for containers.

The dfs.blocksize was set at 1 GiB to take advantage of the large memory available to YARN, and the mapreduce.task.io.sort.mb was consequently set to the largest possible value, 2047 MiB, to minimize spills to disk during the map processing of each HDFS block.

The number of vcores assigned to map and reduce processes (mapreduce.map.cpu.vcores and mapreduce.reduce.cpu.vcores) were left at the default value of 1, meaning that a maximum of 64 map or reduce task containers could run at any one time on the 64-vcore cluster. It was found through experimentation that using all 64 vcores per server provided the fastest performance, but optimum performance was achieved by lowering the per-task memory (mapreduce.map.memory.mb and mapreduce.reduce.memory.mb) from the maximum sustainable by the 448-GiB cluster (7 GiB) down to 6.25 GiB. With each of the 10 nodes running 64 YARN containers, a maximum of 640 task containers could be run simultaneously. One YARN container was needed to run the YARN Application Master, leaving 639 for maps or reduces.

For Spark the calculations are similar: 16 Spark executors were enabled per node, each using 4 vcores (spark.executor.cores) and 25 GiB (spark.executor.memory). Spark automatically adds 10% of Spark executor memory overhead, so the total memory consumed by 16 Spark executors (16×27.5 GiB or 440 GiB) would fit within the 448-GiB cluster. However, it was found that the best Spark performance was obtained while

allowing more free memory and vcores, so the final setup used 15 executors per node or 149 per cluster (again with one YARN container left over for the ApplicationMaster task).

Spark was run in yarn-client mode, meaning that the Spark master process ran on the Spark gateway on the Gateway VM. 10 GiB was assigned to this process (spark.driver.memory).

Finally, the log level of most Hadoop processes was lowered from INFO to WARN to reduce the amount of log traffic being written on each server.

4.2 Results

The results for the three versions of the code are shown in Table 5. Both the consolidated benchmark metric (HSph@3TB) in which larger is faster, and the elapsed times for the three TPCx-HS phases (smaller is better) are shown.

Table 5. Results

Test	TPCx-HS Performance Metric (HSph@3TB)	HSGen Elapsed Time (S)	HSSort Elapsed Time (S)	HSValidate Elapsed Time (S)
MRv1	7.8843	372	918	123
MRv2	7.2974	370	979	127
Spark	8.6281	368	799	79

Utilization of CPU, disks, and network are shown for the three tests in Figs. 1, 2, 3, 4, 5, 6, 7, 8 and 9. One can see how the various resources are utilized through the phases of the benchmark, as well as the small differences between the two MapReduce versions and the Spark version.

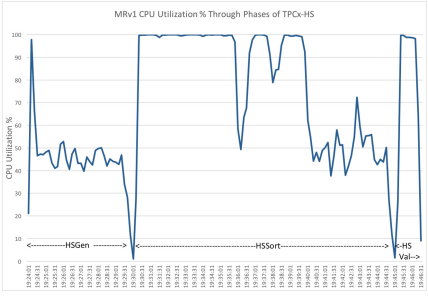

Fig. 1. MRv1 CPU Utilization on a single worker node.

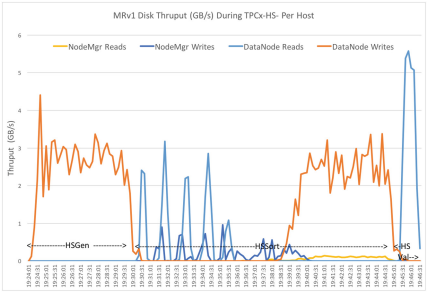

Fig. 2. MRv1 Disk Throughput on a single worker node.

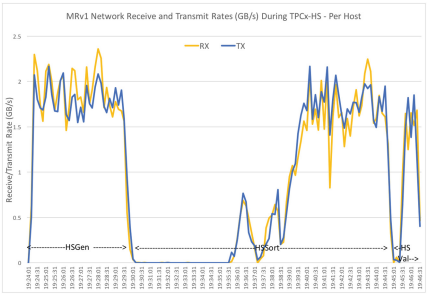

Fig. 3. MRv1 Network Receive and Transmit Rates on a single worker node.

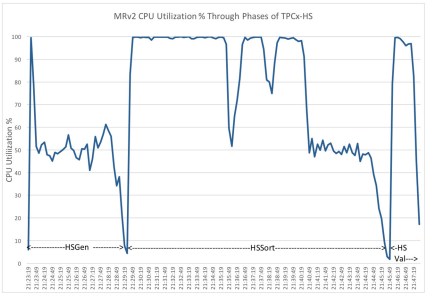

Fig. 4. MRv2 CPU Utilization on a single worker node.

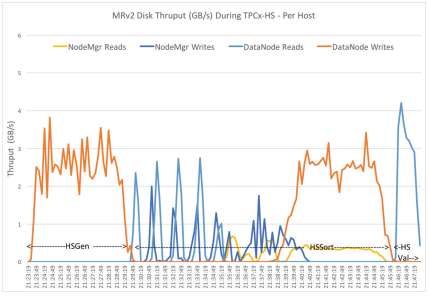

Fig. 5. MRv2 Disk Throughput on a single worker node.

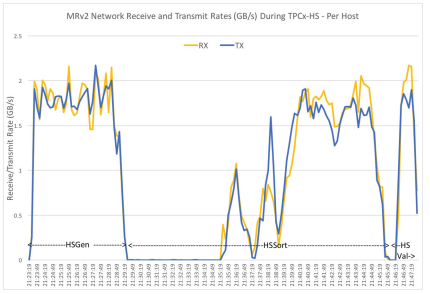

Fig. 6. MRv2 Network Receive and Transmit Rates on a single worker node.

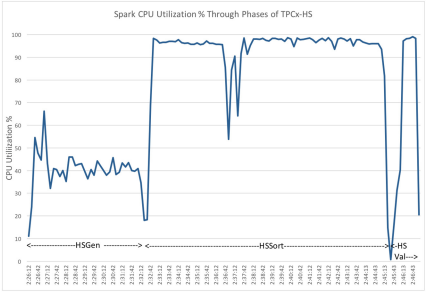

Fig. 7. Spark CPU Utilization on a single worker node.

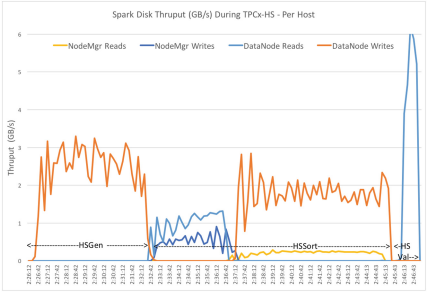

Fig. 8. Spark Disk Throughput on a single worker node.

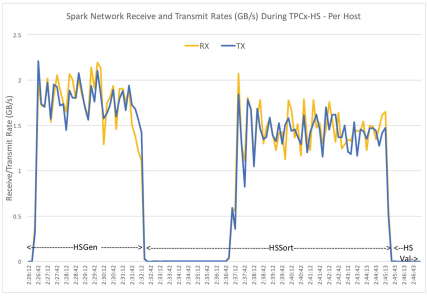

Fig. 9. Spark Network Receive and Transmit Rates on a single worker node.

5 Conclusion

The TPC has played a crucial role in providing the industry with relevant standards for total system performance, price-performance, and energy efficiency comparisons. TPC benchmarks are widely used by database researchers and academia. As Big Data became an integral part of enterprise IT, TPCx-HS was the TPC's first major step in creating a set of industry standards for measuring various aspects of hardware and software systems dealing with Big Data. It has helped drive competition in the Big Data marketplace and has proven to be a successful industry standard benchmark for Hadoop systems.

However, the Big Data technology landscape has rapidly changed since the benchmark's initial release, and in keeping with these changes TPCx-HS has also transformed. TPCx-HS v2 has advanced not only by supporting the significant leaps in technology, namely Hadoop 2 (MapReduce v2/YARN) and Spark, but also by accommodating major new infrastructure and deployment options such as the cloud.

Acknowledgements. Developing a TPC benchmark for a new environment requires a huge effort to conceptualize, research, specify, review, prototype, and verify the benchmark. The authors acknowledge the work and contributions made by Da Qi Ren, David Grimes, Jamie Reding, John Poelman, Karthik Kulkarni, Matthew Emmerton, Meikel Poess, Mike Brey, Paul Cao, and Reza Taheri.

References

1. Nambiar, R., Poess, M., Dey, A., Cao, P., Magdon-Ismail, T., Ren, D.Q., Bond, A.: Introducing TPCx-HS: the first industry standard for benchmarking Big Data systems. In: Nambiar, R., Poess, M. (eds.) TPCTC 2014. LNCS, vol. 8904, pp. 1–12. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15350-6_1
2. TPCx-HS Results. http://www.tpc.org/tpcx-hs/results/tpcxhs_results.asp. Accessed 20 June 2017
3. Taneja Group Spark Market Adoption Report. https://www.cloudera.com/content/dam/www/marketing/resources/analyst-reports/taneja-group-spark-survey-exec-summary-Oct-2016.pdf.landing.html. Accessed 20 June 2017

4. TPC Specifications, http://www.tpc.org//tpc_documents_current_versions/current_specifications.asp. Accessed 20 June 2017
5. Apache Hadoop Project Page, 02 July 2008. http://web.archive.org/web/20080702015052/http://hadoop.apache.org. Accessed 20 June 2017
6. Apache Hadoop Ecosystem and Open Source Big Data Projects. https://hortonworks.com/ecosystems/. Accessed 20 June 2017
7. Getting MapReduce 2 Up to Speed. http://blog.cloudera.com/blog/2014/02/getting-mapreduce-2-up-to-speed/. Accessed 21 June 2017
8. Sort Benchmark Home Page. http://sortbenchmark.org. Accessed 21 June 2017
9. Zaharia, M. et al.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the Ninth USENIX NSDI Symposium on Networked Systems Design and Implementation, San Jose, CA (2012)
10. Guo, Z., Fox, G., Zhou, M., Ruan, Y.: Improving resource utilization in MapReduce. In: 2013 IEEE International Conference on Cluster Computing (CLUSTER) (2013)
11. Vavilapalli, V.K. et al.: Apache Hadoop YARN: yet another resource negotiator. In: Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC), Santa Clara, CA (2013). Article No. 5