A publication of ISCA*:
International Society for Computers
and Their Applications

INTERNATIONAL JOURNAL OF COMPUTERS AND THEIR APPLICATIONS

TABLE OF CONTENTS

Page

*** "International Journal of Computers and Their Applications is abstracted and indexed in INSPECT and Scopus."**

Volume 20, No. 4, Dec. 2013 **ISSN 1076-5204**

International Journal of Computers and Their Applications

A publication of the International Society for Computers and Their Applications

ISCA Headquarters···•···64 White Oak Court, Winona, MN 55987···•···Phone: (507) 458-4517
E-mail: isca@ipass.net • URL: http://www.isca-hq.org

Guest Editorial: Advances in Cloud Computing

Dunren Che*, Guest Editor
Southern Illinois University, Carbondale, IL 62901, USA

Parisa Ghodous[†], Guest Editor
University Lyon 1, Villeurbanne, 69622, FRANCE

Hassan Badir[‡], Guest Editor
Abdelmalek Essaadi University, Tangier, 9000, MOROCCO

Cloud Computing [1, 3] represents a major paradigm shift in computing and information technology strategy. The "Cloud" is a natural evolution of distributed computing and of widespread adoption of the virtualization technology and SOA. In Cloud Computing, IT-related capabilities and resources are provisioned as services, via the Internet and with the essential characteristics such as *on-demand*, *elasticity*, *metered services*, and *rapid provision* (without requiring possession of detailed knowledge of the underlying technology). The International Journal of Computers and Their Applications (IJCA) has thus scheduled this special issue in response to the fast development and increased application of Cloud Computing. This issue includes five selected articles on various topics of Cloud Computing:

1. "Performance Evaluation of Distributed Storage Systems for Cloud Computing," by S. Shirinbab *et al*.
2. "Budget Constrained Dataflow Scheduling for Minimized Completion Time on the Cloud," by D. Ding *et al*.
3. "A Cooperative Game Theory-based Approach for Energy-Aware Job Scheduling in Cloud," by M. Khaleel *et al*.
4. "Moving energy consumption control into the cloud by coordinating services," by G. Vargas-Solar *et al*.
5. "Data Warehouse Systems in the Cloud: Rise to the Benchmarking Challenge," by R. Moussa *et al*.

Load balancing (and load rebalancing) is a critical management task in Cloud Computing. If properly done, it helps to achieve the promised QoS (in contrast to otherwise deteriorated performance especially on congested server machines) and avoiding quick wearing out of heavily used servers. The task of load balancing relates to many other issues in Cloud Computing, for example, if properly done, it may facilitate "green computing" – that is, when the task is carried out toward consolidating sparse computing jobs (which happens typically at non-peak times) onto fewer physical server machines, this will result in more idle servers that can be shut down in favor of reducing energy consumption. Load balancing inevitably requires live migration of virtual servers, which in turn requires the provision of large shared storage systems accessible to all the physical servers involved in a cloud. Distributed storage systems offer reliable and cost-effective storage for large amounts of data and thus become a favored choice for supporting live migration of virtual servers in a Cloud. In article 1 of this special issue, the authors evaluated four large distributed storage systems, and provided insight that are helpful for potential cloud providers in future consideration of a distributed storage systems for supporting live migration of virtual servers in their clouds. The article concluded that in general the multicast approach outperforms another popular approach – Distributed Hash Table.

Cloud Computing has emerged as a promising computing paradigm for large-scale data intensive applications and as an ideal platform to face the unprecedented challenges of Big Data and Big Data Analytics [2], which is currently an exhortation in the discipline of Commuter Science and the IT industry. Many such data intensive applications are best modeled as complex Directed Acyclic Graphs (DAGs) [5], which in essence are structured processing data flows with arbitrary data operators being modeled as nodes and producer-consumer interactions modeled as directed edges in the DAGs. The optimization problem of dataflow scheduling on clouds is a very challenging task. The optimization must satisfy a variety of objectives and constraints, including fitting into the particular characteristics of an underlying cloud environment. Job completion time and user's budget constraint (especially under the current global economic atmosphere) are the two most prominent parameters in the optimization of dataflow scheduling on clouds. In article 2, the authors formulated dataflow scheduling problem in a cloud environment toward the objective of minimizing the job completion time under a certain budget constraint. A heuristic scheduling algorithm, called LRA-B (Layer-oriented Resource Allocation within Budget constraint) was proposed and experimentally

* Department of Computer Science. Email: dche@cs.siu.edu.
† Department of Computer Science, LIRIS UMR 5205. Email: ghodous@liris.cnrs.fr.
‡ LabTIC Lab., Departement of Computer Science SIC. Email: hassan.badir@uae.ma.

evaluated.

To a great extent, green computing means less power consumption and higher utilization of other resources [1, 4, 6]. Article 3 addresses the problem of energy-aware job scheduling on underlying cloud nodes using a cooperative game theory. This work inspects a bi-objective, maximization of resource utilization and minimization of power consumption under the constraint of not sacrificing a module's latest completion time (Make span). Cloud providers always have the keen interest in an efficient and cost-effective job scheduling strategy with low power consumption and high job throughput. This article presents an energy-aware job scheduling algorithm given a bag of tasks based on the premise of Nash Bargaining Solution (NBS). The article also demonstrates the effectiveness of the proposed algorithm via simulation-based evaluation and comparison with related work.

Continuing on the same theme as article 3, i.e., energy-efficiency, the authors of article 4 presented a cloud-based and service-oriented approach for collecting, integrating, storing, and analyzing energy consumption data. In their work, energy sensors are utilized and modeled as cloud services that carries information regarding various aspects of energy consumption and can be composed into distinct (monitoring and controlling) scenarios at different granularity levels best suiting users' particular needs and requirements, such as home-owners, energy providers, local and regional planning authorities, etc., which all concern about energy consumption.

While Big Data and Big Data Analytics [2], though being the buzzwords for a couple of years, still remain at their fledging stage of research and development, migrating data warehouse systems into the clouds appears to be a practical and immediately deliverable approach. Accordingly, there emerges the necessity for benchmarking data warehouse systems running in the clouds. Although there are popular benchmarks for cloud computing such as Terasort and YCSB, and prominent benchmarks for decision support systems such as the Transaction Processing Council's TPC-H and TPC-DS benchmarks, however, specialized benchmarks for cloud-hosted data warehouse systems remain to be developed. Such benchmarks must take into account the specific rationale of clouds (e.g., scalability, elasticity, pay-per-use, QoS, and fault-tolerance) and that of data warehouse systems and related OLAP technologies. The last article in this special issue, article 5, discusses the new requirements for implementing a benchmark for data warehouse systems in clouds and sets a preliminary foundation with the potential of facilitating fair comparisons of data warehouse systems hosted and running on different cloud providers' platforms.

Acknowledgement

As guest editors, we would like to express our genuine appreciation for the encouragement and support from the former and current editor-in-chiefs, Qiang Zhu and Fred Harris, of the journal. Our appreciation shall well extend to Professor Aris M. Ouksel, who, as the general chair of AICCSA'13, helped bridging the conference and this special issue (We accepted two recommended papers from AICCSA'13 and included their extended versions in this special issue). We also owe many thanks to our authors and reviewers who contributed to this special issue.

References

[1] D. Che and W.-C. Hou, "A Novel 'Credit Union' Model of Cloud Computing," *Proc. of DICTAP*, Dijon, France, pp. 714-727, June 21-23, 2011.

[2] D. Che, M. Safran1, and Z. Peng, "From Big Data to Big Data Mining: Challenges, Issues, and Opportunities," *Proc. of DASFAA-BDMA* Workshop, Wuhan, China, pp. 1-15, April 2013.

[3] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," *Proc. of Grid Computing Environments Workshop*, Austin, TX, pp. 1-10, Nov. 2008.

[4] S. Khan and I. Ahmad, [A3-2], "A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids," *IEEE Transactions on Parallel and Distributed Systems*, 20(3):346-360, Oct. 2009.

[5] H. Kllapi, E. Sitaridi, M. M. Tsangaris, and Y. Loannidis, "Schedule Optimization for Data Processing Flows on the Cloud," *Proc. of SIGMOD'11*, Athens, Greece, pp. 289-300, June 12-16, 2011.

[6] Y. C. Lee and A. Y. Zomaya, [A3-1], "Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling," *Proc. of the 9th IEEE/ACM Inter. Symposium on Cluster Computing and the Grid (CCGRID '09)*, Shanghai, China, pp. 92-99, May 18-21, 2009.

Performance Evaluation of Distributed Storage Systems for Cloud Computing

Sogand Shirinbab*, Lars Lundberg*, and David Erman*
Blekinge Institute of Technology, 371 79 Karlskrona, SWEDEN

Abstract

The possibility to migrate a virtual server from one physical computer in a cloud to another physical computer in the same cloud is important in order to obtain a balanced load. In order to facilitate live migration of virtual servers, one needs to provide large shared storage systems that are accessible for all the physical servers that are used in the cloud. Distributed storage systems offer reliable and cost-effective storage of large amounts of data and such storage systems will be used in future Cloud Computing. We have evaluated four large distributed storage systems. Two of these use Distributed Hash Tables (DHTs) in order to keep track of how data is distributed, and two systems use multicasting to access the stored data. We measure the read/write/delete performance, as well as the recovery time when a storage node goes down. The evaluations are done on the same hardware, consisting of 24 storage nodes and a total storage capacity of 768 TB of data. These evaluations show that the multicast approach outperforms the DHT approach.

Key Words: Cloud computing, compuverde, distributed storage system, file system, gluster, OpenStack (Swift).

1 Introduction

The possibility to migrate a virtual server from one physical computer in a cloud to another physical computer in the same cloud is important in order to obtain a balanced load. In order to facilitate live migration of virtual servers, one needs to provide large shared storage systems that are accessible for all the physical servers that are used in the cloud. This is an important reason why the demand for storage capacity has increased rapidly during the last years.

One problem with traditional disk drives is that data losses are common due to hardware errors. A solution to this is Redundant Array of Independent Disks (RAID) storage. RAID storage systems can automatically manage faulty disks without losing data, and scale by attaching new disk drives. However, the scalability of RAID is too limited for large cloud systems; this limitation is the main reason for using distributed storage systems.

Distributed storage systems should be capable of sustaining rapidly growing storage demands, avoid loss of data in case of hardware failure, and they should provide efficient distribution of the stored content [33]. Two examples of distributed storage systems are OpenStack's Swift[1] and Gluster[2]. We have evaluated the performance of three distributed storage systems: Compuverde, OpenStack's Swift, and Gluster. Openstack's Swift and Gluster are both open-source distributed storage systems that are available for downloading and testing.

Some distributed storage systems use Distributed Hash Tables (DHTs) for mapping data to physical servers. In the DHT approach file names and addresses are run through a hashing function in order to indentify the nodes that have the requested data. Two examples of systems that use DHTs are Gluster and OpenStack's Swift [15]. An alternative approach to using DHTs is to use multicasting where data requests are sent to multiple storage nodes and the nodes that have the requested data answer. Compuverde uses the multicast approach. The architectural advantage of DHTs compared to multicasting is that we do not need to broadcast requests; the hash table gives us the address of the nodes that store the requested data and we avoid communication overhead. However, the obvious disadvantage with DHTs is that we need to run a hash function to obtain the address of the data, which introduces processing overhead. This means that the architectural decision, whether to use DHTs or multicasting will introduce different kinds of overhead: processing overhead for DHTs and communication overhead for multicasting. Using DHTs or multicasting is a key architectural decision in distributed storage systems for Cloud Computing and this performance evaluation will give important insights regarding the performance implications of this decision.

2 Background

In distributed storage systems, the most common interfaces are Web Service APIs (Application Programming Interface) like Internet Small Computer System Interface (iSCSI) [38]; REpresentational State Transfer (REST)-based [19, 25] and Simple Object Access Protocol (SOAP)-based [14]. REST is a HTTP-based architectural style to build networked

* Department of Computer Science, School of Computing. E-mail: {Sogand.Shirinbab, Lars.Lundberg, David.Erman}@bth.se.

[1] http://openstack.org/.
[2] http://www.gluster.org/.

applications that allows access to stored objects by an Object Identifier (OID), i.e., no file or directory structures are supported [17]. We will refer to object-based storage systems as *unstructured storage systems*.

There are other access methods like Network File System (NFS) and Common Internet File System (CIFS) which are used for accessing storage on a private network or LAN and Web-based Distributed Authoring and Versioning (WebDAV) which is based on HTTP. These APIs are file-based (variable-size) and use a path to identify the data; we denote these as *structured storage systems*. The architecture of structured storage systems is similar to Network Attached Storage (NAS) which provide file system functionality, i.e., structured storage systems support variable file and directory structures [9, 22].

The most well-known distributed storage systems are Amplistor [2, 13], Caringo's CAStor [7-8], Ceph [6], Cleversafe[3] , Compuverde[4] , EMC Atmos [16], Gluster [23], Google File System [21], Hadoop [11, 27], Lustre [32], OpenStack's Swift [29], Panasas [1], Scality[5] and Sheepdog[6]. Some of the distributed file systems could be used by other applications, i.e., BigTable is a distributed storage for structured data and it uses GFS to store log and data files [10].

As shown in Table 1 AmpliStor, CAStor, Ceph, Cleversafe and Scality are unstructured distributed storage systems. Amplistor is designed to work with HTTP/REST. Just as in Amplistor, CAStor's Simple Content Storage Protocol (SCSP) is based on HTTP using a RESTful architecture [26]. Ceph provides an S3-compatible REST interface that allows applications to work with Amazon's S3 service. Cleversafe provides an iSCSI device interface, which enables users to transparently store and retrieve files as if they were using a local hard drive.

EMC Atmos is a structured distributed storage system that provides CIFS and NFS interfaces, as well as web standard interfaces such as SOAP and REST. Other distributed file systems such as Google File System, Hadoop Distributed File System (HDFS), Lustre and Panasas provide a standard POSIX API. Sheepdog is the only distributed storage system which is based on Linux QEMU/KVM and is used for virtual machines.

Some of the distributed file systems are also used for computing purposes, e.g., the Hadoop Distributed File System (HDFS) which distributes storage and computation across many servers. HDFS stores file system metadata and application data separately and users can reference files and directories by paths in the namespace (a HTTP browser can be used to browse the files of an HDFS instance) [18]. Lustre is an object-based file system used mainly for computing purposes. The Lustre architecture is designed for HighPerformance Computing (HPC). Panasas is also used for computing purposes and similar to Lustre, it is designed for HPC.

Scality uses a ring storage system which is based on a Distributed Hashing Mechanism with transactional support and failover capability for each storage node. The Sheepdog architecture is fully symmetric and there is no central node such as a meta-data server (Sheepdog uses the Corosync cluster engine [4] to avoid metadata servers). Sheepdog provides an object (variable-sized) storage and assigned a global unique id to each object. In Sheepdog's object storage, target nodes calculated based on consistent hashing algorithm which is a schema that provides hash table functionality and each object is replicated to 3 nodes to avoid data loss [35].

The remaining distributed storage systems in Table 1 are Compuverde, Gluster and OpenStack's Swift. We have ported these three systems to the same hardware platform (see Section 3), thus making it possible to compare their performance (see Sections 4 and 5). In Subsections 2.1, 2.2, and 2.3, we discuss these three systems in detail.

Distributed storage systems use either multicasting or Distributed Hash Tables (DHTs). Data redundancy is obtained by either using multiple copies of the stored files or by so called striping using Reed-Solomon coding [20]. When using striping the files are split into stripes and a configurable number of extra stripes with redundancy information are generated. The stripes (in case of Striping) and file copies (in case of Copying) are distributed to the storage nodes in the system.

2.1 Compuverde

Compuverde has no separate metadata. The system uses its own proprietary caching mechanism (SSD Caching that employs Write-back policy) [5] in the storage nodes. The solution uses multicasting, and supports geographical dispersion, heartbeat monitoring, versioning, self-healing and self-configuring. Compuverde supports a flat 128 bit addresses space (for unstructured storage) and NFS/CIFS (for structured storage). The system supports both Linux and Windows. Compuverde's storage solution consists of two parts: The first part is unstructured and it contains all storage nodes (clusters). The other part is the structured part of the storage solution. This part contains gateways (this corresponds to what OpenStack calls proxy servers) to communicate with storage nodes. The communication is based on TCP unicast and UDP multicast messages. Structured data storage is achieved by storing information about the structure in envelopes. An envelope is an unstructured file that is stored on the storage nodes and contains information about other envelopes and other files.

The storage cluster provides mechanisms for maintaining scalability and availability of the structured data by replicating the envelopes a (configurable) number of times within the cluster as well as providing access to them by the use of IP-multicast technology.

The communication between the structured and the unstructured layers starts with an IP-multicast of a key from the gateway; this key identifies the requested envelope. All nodes that have the requested envelope reply with information about the envelope and what other nodes contain the requested envelope, with the current execution load on the storage node. The gateway collects this information and waits until it has received answers from more than 50 percent of the listed

[3] http://www.cleversafe.com/.
[4] http://compuverde.com/.
[5] http://www.scality.com/.
[6] http://www.osrg.net/sheepdog/.

Table 1: Overview of different distributed storage systems

	INTERFACE				**SOLUTION**		**REPLICATION**		**METADATA**	
	Unstructured		Structured							
	Web Service APIs (REST, SOAP)	*Block-based APIs (iSCSI)*	*File-based APIs (CIFS, NFS)*	*Other APIs (WebDAV, FTP, Proprietary API)*	DHT	Multicast	Copy-ing	Striping	Centralized	Distributed
AmpliStor	X	-	-	-	-	-	-	X		X
Caringo's CAStor	X	-	X	-	-	X	X	-	X	-
Ceph	X	-	-	-	X	-	-	X	-	X
Cleversafe	-	X	-	-	-	-	-	X	X	-
Compuverde	X	-	X	X	-	X	X	-	-	X
EMC Atmos	X	-	X	-	X	-	-	X	-	X
Gluster	-	-	X	X	X	-	X	-	-	-
Google File System (GFS)	X	-	X	-	-	-	-	X	X	-
Hadoop	-	-	X	-	X	-	-	X	X	-
Lustre	-	-	X	-	X	-	-	X	X	-
OpenStack's Swift	X	-	-	-	X	-	X	-	-	X
Panasas	-	-	X	-	-	-	-	X	-	X
Scality	X	-	-	-	X	-	X	-	-	X
SheepDog	-	X	-	-	X	-	X	-	-	X

storage nodes that contains the identifier before it makes a decision on which one to select for retrieval of the file.

2.2 Gluster

Gluster is a structured distributed storage system. Storage servers in Gluster support both NFS and CIFS. Gluster does not provide a client side cache in the default configuration [34]. Gluster only provides redundancy at the server level, not at the individual disk level. For data availability and integrity reasons Gluster recommends RAID 6 or RAID 5 for general use cases. For high-performance computing applications, RAID 10 is recommended.

Distribution over mirrors (RAID 10) is one common way to implement Gluster. In this scenario, each storage server is replicated to another storage server using synchronous writes. In this strategy, failure of a single storage server is transparent, and read operations are spread across both members of the mirror.

Gluster uses the Elastic Hash Algorithm (EHA). EHA determines where the data are stored and is a key to the ability to function without metadata. A pathname/filename is run through the hashing algorithm. After that, the file is placed on the selected storage. When accessing the file, the Gluster file system uses load balancing to access replicated instances. Gluster offers automatic self-healing [23, 37].

2.3 OpenStack's Swift

OpenStack's Swift is an unstructured distributed storage system. A number of "zones" are organized in a logical ring which represents a mapping between the names of entities stored on disk and their physical location. Swift is configurable in terms of how many copies (called "replicas") are stored, as well as how many zones are used. The system tries to balance the writing of objects to storage servers so that the write and read load is distributed.

The mapping of objects to zones is done using a hash function. Swift does not do any caching of actual object data but Swift-proxys can work with a cache (Memcached[7]) to reduce authentication, container, and account calls [30]. In Swift, there are separate rings for accounts, containers, and objects. When other components need to perform any operation on an object, container, or account, they interact with the appropriate ring to determine its location in the cluster. OpenStack's Swift's rings are responsible for determining which devices to use in failure scenarios [3, 28-29, 31, 36].

OpenStack's Swift divides the storage space into partitions. In our case, 18 bits of the GUID are used to decide on which partition a certain file should be stored, i.e., there are 2^{18} = 262 144 partitions. These partitions are divided into 6 zones. Zone 0 is mapped to storage nodes 0 to 3, zone 1 is mapped storage nodes 4 to 7, and zone 5 is mapped to storage nodes 20 to 23. Storage nodes 0 to 7 are handled by one switch, nodes 8 to 15 by one switch and nodes 16 to 23 by one switch (see Figure 1). There are 24*16 = 384 disks in the system and the 262 144 partitions are spread out with 682 or 683 partitions on each

[7] Memcached is a distributed memory object caching system.

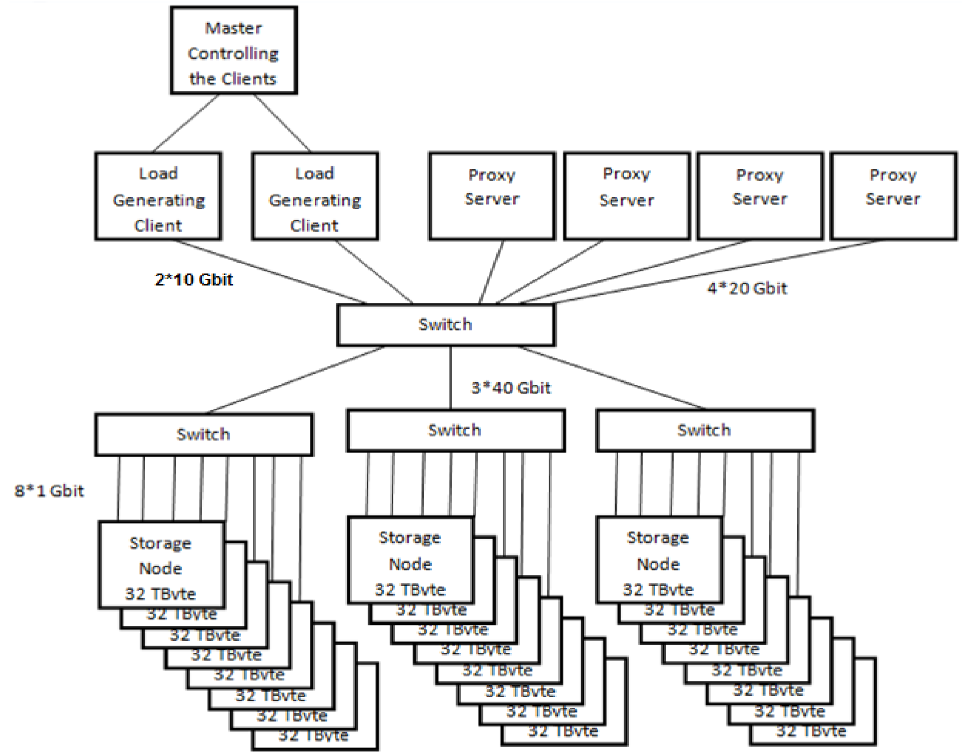


Figure 1: The physical structure of the test configuration

disk (262144/384 = 682.666…). If a file is stored on partition X, the two extra copies of the file (there are three copies of each file) are stored on partitions (X + 87 381) mod 262 144, and (X + 2 * 87 381) mod 262 144 (262 144 / 3 = 87 381.333…).

3 Experimental Setup

3.1 Test Configurations

Four different storage system configurations have been evaluated:

1. Compuverde Unstructured
2. Compuverde Structured
3. OpenStack's Swift (an unstructured storage system)
4. Gluster (a structured storage system)

The measurements use two load generating clients (see Figure 1). We use the same load for each configuration; the only part that has been changed is the interface. The clients work synchronously and report the result to the master controlling the clients (see Figure 1), which is responsible for monitoring the throughput.

In the configurations 1 and 2, Compuverde 0.9 has been installed on CentOS 6.2. In the configuration 3, version 1.4.3 of the OpenStack's Swift (release name: Diablo) has been installed on Linux Ubuntu 10.04 and in the configuration 4, Gluster 3.2.5 has been installed on CentOS 6.2.

The same hardware is used in each configuration. The storage system consists of 24 storage nodes, each containing sixteen 2 TB disks, i.e., a total of 32 TB for each node and 768 TB storage for all 24 nodes. With the exception of configuration 1 (Compuverde Unstructured), all accesses to the storage system are routed through four proxy (gateway) servers. In configuration 1 the clients communicate directly with the storage system.

Each proxy server has an Intel Quad processor, 16 GB RAM, and two 10 Gbit network cards. Each storage node has an Intel Atom D525 processor, 4 GB RAM, and a 1 Gbit network card. All storage nodes and proxy servers run the Linux operating system. There are four switches that are used to transmit data from four proxy servers and two load generating clients to the 24 storage nodes. The central switch is a Dell 8024F and the other three switches are Dell 7048Rs. Four proxy servers are connected to the central switch via four 20 Gbit fibers. Two load generating clients are connected to a central switch via two 10 Gbit fibers and the central switch is connected to the other three switches via three 40 Gbit fibers.

The four test configurations will now be described.

3.1.1 Compuverde Unstructured. In this configuration three copies of each file are created. The proxy servers are not used, and the load generating clients communicate directly with the storage nodes.

3.1.2 Compuverde Structured. In this case two copies of each file are created. The reason for this is that this case will be compared with Gluster, and Gluster only supports two copies of each file. The two load generating clients communicate with two proxy servers each. The communication protocol between the load generating clients and the proxy servers is NFS/CIFS.

3.1.3 OpenStack's Swift. OpenStack's Swift has three copies of each file, and the two load generating clients communicate with two proxy servers each.

3.1.4 Gluster. Gluster dedicates a volume to the lock file. In Gluster the storage nodes are arranged in pairs to obtain fault tolerance. This means that there are only two copies of each file. The communication protocol between the load generating clients and the proxy servers is NFS/CIFS.

3.2 Test Cases

Two kinds of tests are considered in this study: performance tests and recovery tests.

3.2.1 Performance Tests. In these test cases the read, write and delete performance are measured:

There are four test cases for each test configuration:

1. We measure write performance. In these tests, a number of clients (implemented as full speed threads, i.e., as threads that issue write requests in a tight loop without any delay and with only minimal processing done between each request) running on two servers (see Figure 1) create files of size 0 KB, 10 KB, 100 KB, 1 MB and 10 MB, respectively. Writing 0 KB corresponds to creating a file and will be reported separately. We vary the number of clients using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. A write operation is a combination of Open, Write and Close. We measure MB/s and operations/s.
2. We measure read performance. In these tests, a number of clients (implemented as full speed threads) running on two servers (see Figure 1) read files of size 0 KB, 10 KB, 100 KB, 1 MB and 10 MB, respectively. Reading 0 KB corresponds to opening a file and will be reported separately. We vary the number of clients using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. A read operation is a combination of Open, Read and Close. We measure MB/s and operations/s.
3. We measure delete performance. In these tests, a number of clients (implemented as full speed threads) running on two servers (see Figure 1) delete files of size 10 KB, 100 KB, 1 MB and 10 MB, respectively. We vary the number of clients using the steps 2, 4, 8, 16, 32, 64, 128, and 256 clients. We measure operations/s.
4. For the structured storage case, we use the SPECsfs2008 performance evaluation tool[8]. The tool can be configured to issue a number of I/O Operations per Second (IOPS), and it then measures the actual achieved throughput in terms of IOPS and the average response time.

[8] http://www.spec.org/sfs2008.

The performance tests for small file sizes (0 KB and 10 KB) have been done by writing/reading/deleting 1,000,000 files to/from the storage nodes, but for larger file sizes (100 KB, 1 MB and 10 MB) the test has been continued by writing/reading/deleting files (between 50,000 and 100,000 files) until the results become stable.

Gluster and OpenStack's Swift do not use caching. In order to get fair results, the test has been done for Compuverde for two cases: caching and No Caching (NC). We limited the NC tests to 1 MB files

3.2.2 Recovery Tests. In these tests we measure how long it takes for the storage system to reconfigure itself after a node failure. We measure recovery performance by reformatting one storage node. When a storage node is reformatted the file copies stored on that node are lost. We measure the time until the system has created new copies corresponding to the copies that were lost.

4 Read and Write Performance

In this section we look at the read and write performance of each of the four configurations. In Section 5 we compare the different configurations.

4.1 Compuverde Unstructured

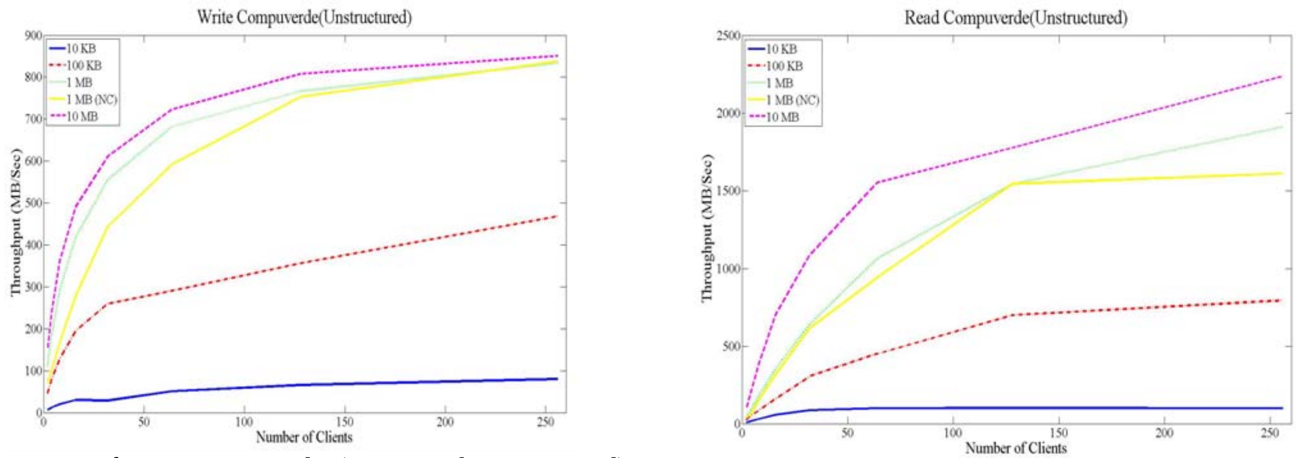
Figures 2a and 2b show that the throughput is low when the number of clients and the size of the files are small; the throughput increases when the number of clients and the size of the files increase. It can also be noted that the performance in case of using cache in the storage nodes, e.g., 1 MB files, does not differ much compared to the case that using NC, i.e., 1 MB (NC).

4.2 Compuverde Structured

Figures 3a and 3b show that the data transfer rate is low when the number of clients and the size of the files are small and it increases when the number of clients and size of files increase. It can also be noted that the performance difference between using caching in the storage nodes, e.g., 1 MB files, and using NC, i.e., 1 MB (NC), is approximately a factor of 1.5 when writing; there is no significant difference between caching and NC when reading.

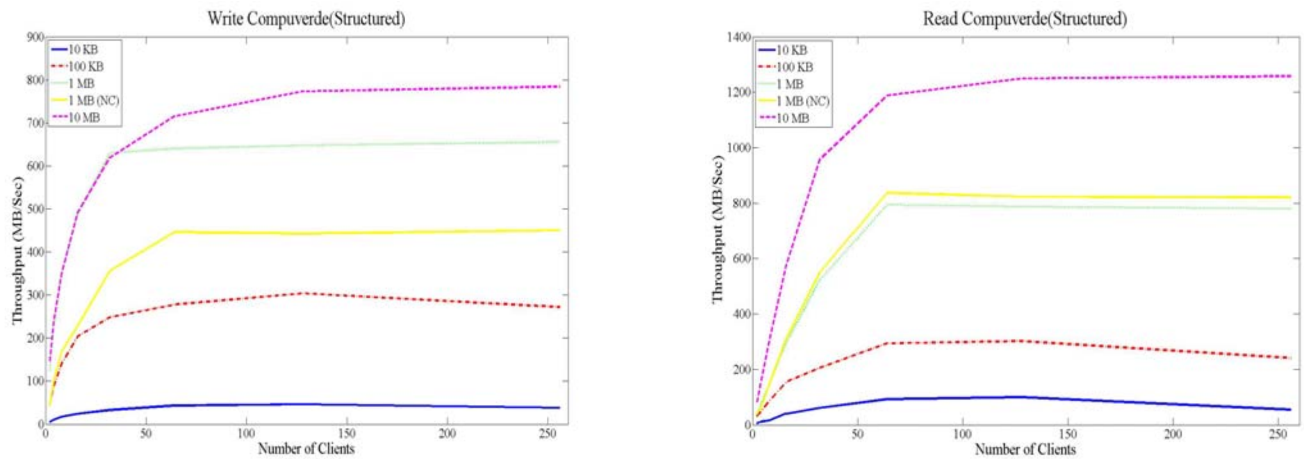
4.3 OpenStack's Swift

Figures 4a and 4b show that in cases of writing/reading the files of files of large size (10 MB), the data transfer rate increases rapidly when the number of the clients increases. While in case of writing files with size of 1 MB and less the curve is quite stable.



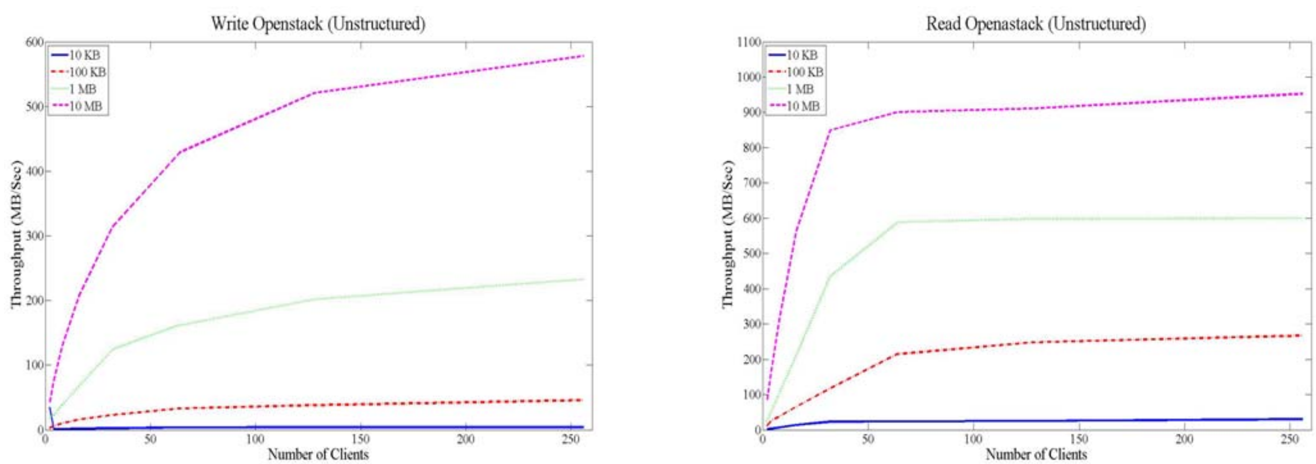
(a) Write performance test results (compuverde unstructured) (b) Read performance test results (compuverde unstructured)

Figure 2: In figures (a) and (b) the *y*-axis denotes the data transfer rate in MB/s, while the *x*-axis denotes the number of clients that are writing/reading simultaneously



(a) Write performance test results (compuverde structured) (b) Read performance test results (compuverde structured)

Figure 3: In figures (a) and (b) the *y*-axis denotes the data transfer rate in MB/s, while the *x*-axis denotes the number of clients that are writing/reading simultaneously



(a) Write performance test results (openstack) (b) Read performance test results (openstack)

Figure 4: In figures (a) and (b) the *y*-axis denotes the data transfer rate in MB/s, while the *x*-axis denotes the number of clients that are writing/reading simultaneously

4.4 Gluster

Figures 5a and 5b show that the data transfer rate for large files increases when the number of clients increases. However, for smaller files the transfer rate does not increase so much when the number of clients increases.

In fact, when the number of clients exceeds a certain value the transfer rate starts to decrease. The reason for this is probably that Gluster contains contention bottlenecks internally. According to the performance test results, the utilization for the storage nodes never exceeds 50 percent for Gluster. For the other test configurations we get much higher utilization values. This is an indication that there are internal performance bottlenecks in Gluster.

5 Comparing the Distributed Storage Systems

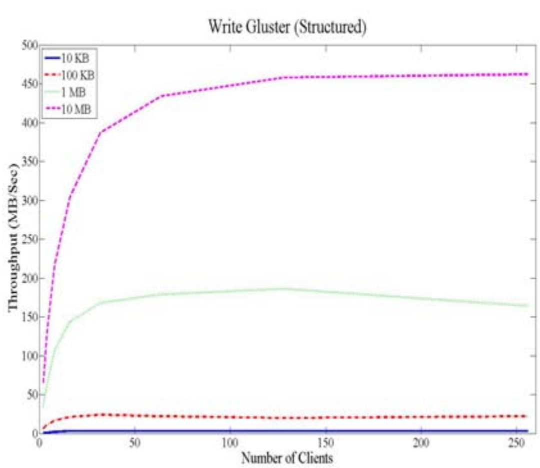
We have evaluated two unstructured storage systems (OpenStack's Swift and Compuverde Unstructured) and two structured storage systems (Gluster and Compuverde Structured). In Section 5.1 we compare the performance of the two unstructured systems and in Section 5.2 we compare the performance of the two structured systems. In Section 5.3 we compare the time to recreate all the file copies in a storage system in case one of the storage nodes fails.

5.1 Compuverde Unstructured vs. OpenStack's Swift

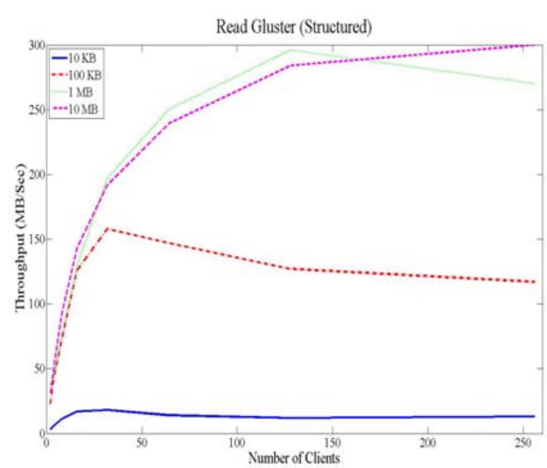
We talked to several cloud storage providers and it turned out that most of their users store small files with an average size of 1 MB. Therefore, the performance tests are compared only for 1 MB. Figure 6a shows that the write performance of Compuverde Unstructured for 256 clients (both when using caching and NC) was roughly 800 MB/s, while for OpenStack's Swift it was around 200 MB/s. Figure 6b shows that the read performance of Compuverde Unstructured for 256 clients (both when using caching and NC) was 1600 MB/s to 1900 MB/s, while for OpenStack's Swift it was around 600 MB/s. Figure 6c shows that the create files performance of Compuverde Unstructured for 256 clients was 10,118 operations/s in case of caching and 6,500 operations/s in case of NC; for OpenStack's Swift it was 600 operations/s. Figure 6d show that the open files performance of Compuverde Unstructured for 256 clients was 11,153 operations/s in case of caching and 12,826 operations/s in case of NC; for OpenStack's Swift it was 4,500 operations/s. The delete files performance test has been done by deleting files with a size of 1 MB. Figure 6e shows that the delete files performance of Compuverde Unstructured for 256 clients was 9956 operations/s in case of caching and 8,145 operations/s in case of NC; for OpenStack's Swift it was 498 operations/s.

5.2 Compuverde Structured vs. Gluster

The write/read/delete performance tests have been done for 1 MB file size. Figure 7a shows that the write performance of Compuverde Structured for 256 clients was 655 MB/s in case of caching and 450 MB/s in case of NC; for Gluster it was 164 MB/s. Figure 7b shows that the read performance of Compuverde Structured for 256 clients was 780 MB/s in case of caching and 821 MB/s in case of NC; for Gluster it was 270 MB/s. Figure 7c shows that the performance for Compuverde Structured for 256 clients was 7,370 operations/s in case of caching and 1,239 operations/s in case of NC; for Gluster it was 241 operations/s. Figure 7d shows that the performance for Compuverde Structured for 256 clients was 11,116 operations/s in case of caching and 12,458 operations/s in case of NC; for Gluster it was 1,029 operations/s. The delete files performance test has been done by deleting files of 1 MB size. Figure 7e shows that the performance for Compuverde Structured for 256 clients was 3,548 operations/s in case of caching and 3,367 operations/s in case of NC; for Gluster it was 441 operations/s. The test results using the Spec2008sfs tool are shown in Figures 8a and 8b. Figure 8a shows that

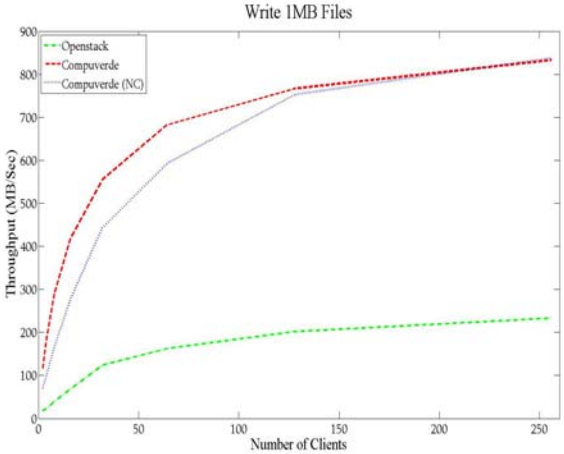


(a) Write performance test results (gluster)

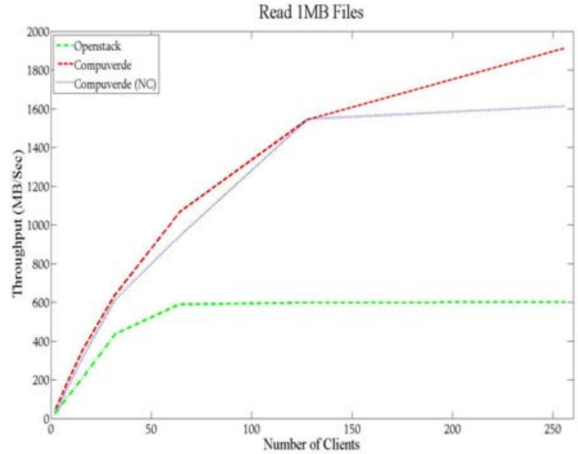


(b) Write performance test results (gluster)

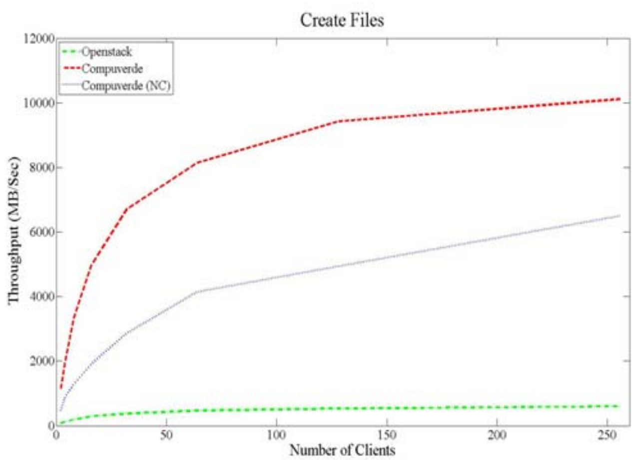
Figure 5: In figures (a) and (b) the y-axis denotes the data transfer rate in MB/s, while the x-axis denotes the number of clients that are writing/reading simultaneously



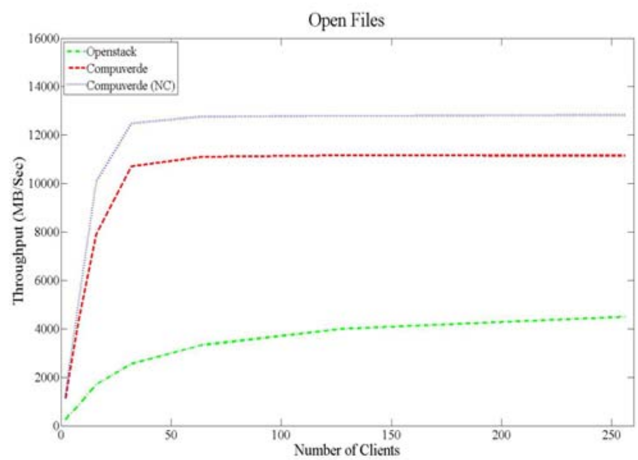
(a) Write performance compuverde unstructured vs. openstack's swift



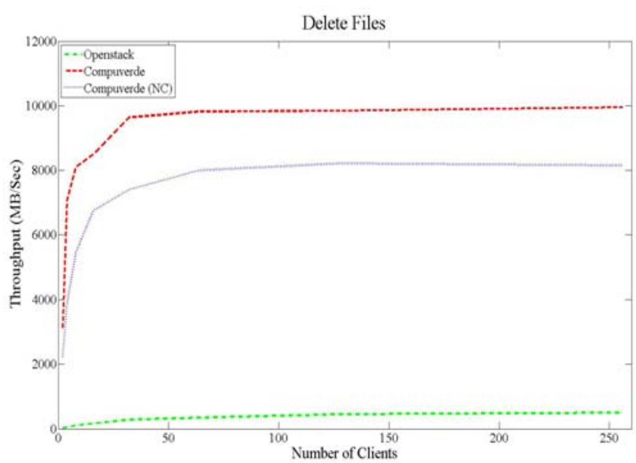
(b) Read performance compuverde unstructured vs. openstack's swift



(c) Create files performance compuverde unstructured vs. openstack's swift

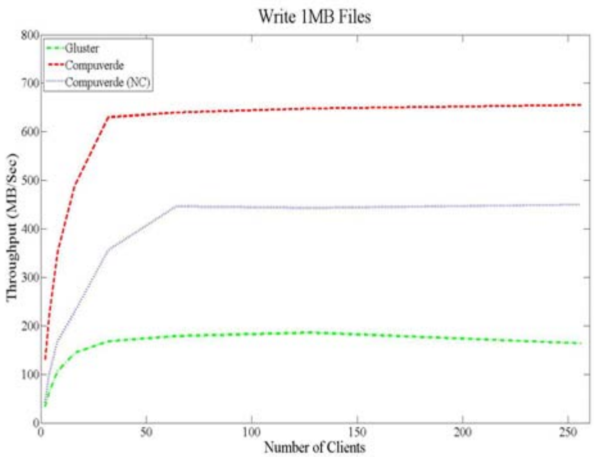


(d) Open files performance compuverde unstructured vs. openstack's swift

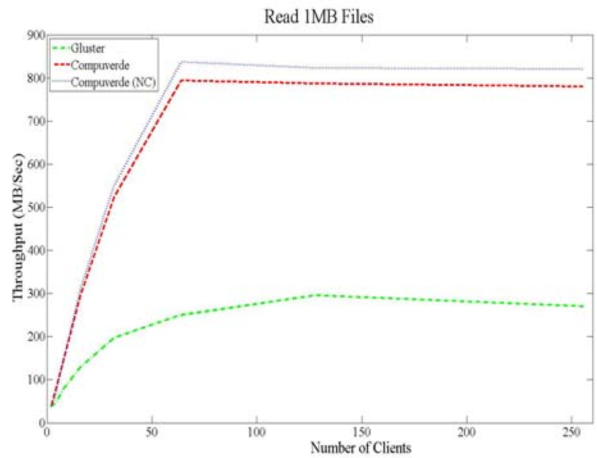


(e) Delete files performance compuverde unstructured vs. openstack's swift

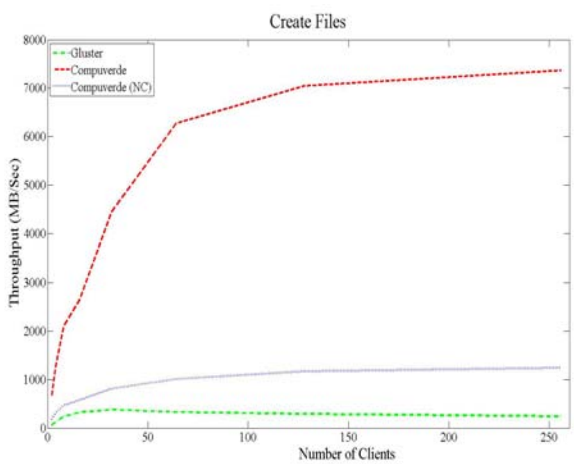
Figure 6: Comparison between the performance of compuverde unstructured and openstack's swift for files of 1 MB



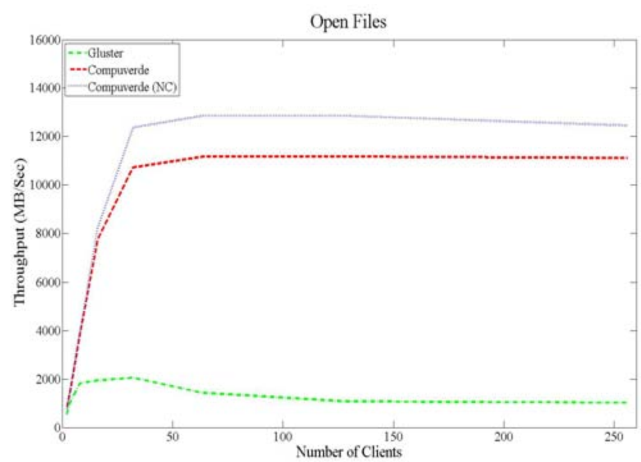
(a) Write performance compuverde structured vs. gluster



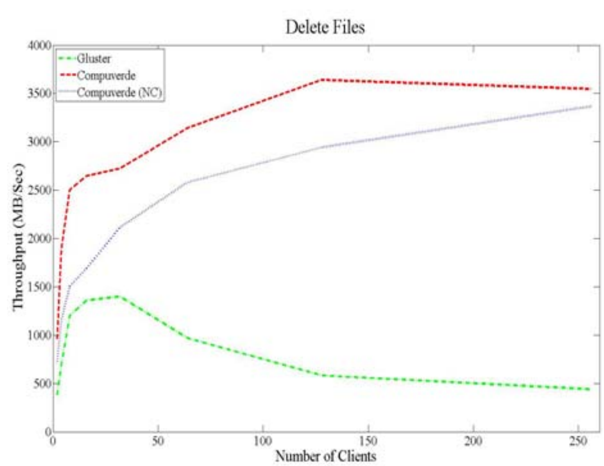
(b) Read performance compuverde structured vs. gluster



(c) Create files performance compuverde structured vs. gluster

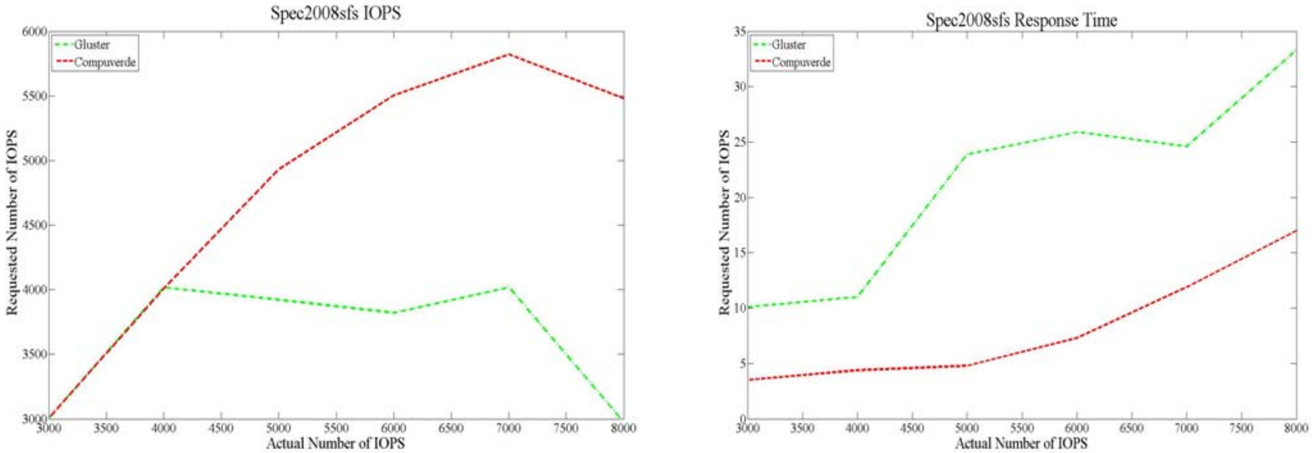


(d) Open files performance compuverde structured vs. gluster



(e) Delete files performance compuverde structured vs. gluster

Figure 7: Comparison between the performance of compuverde structured and gluster for files of 1 MB



(a) Performance evaluation compuverde structured vs. gluster (b) Performance evaluation compuverde structured vs. gluster

Figure 8: Comparison between the performance of compuverde structured and gluster when using the spec2008sfs tools

both Compuverde Structured and Gluster meet the number of requested IOPS for 3000 IOPS and 4000 IOPS. However, when the requested numbers of IOPS increased to 5000 and above, Compuverde Structured delivered a number of IOPS relatively near to the requested one, while Gluster delivers a number of IOPS that is significantly lower than the requested number. Figure 8b shows the result of response time test that has been obtained using the Spec2008sfs performance evaluation tool. Compuverde's response time is in the range of 3.5 ms to 17 ms, while for Gluster the response time is between 10.1 ms and 33.3 ms.

5.3 Recovery Test

We did the recovery test for all four different configurations. The same recovery test has been run twice for each configuration.

As shown in Table 2, the recovery time for Compuverde Unstructured was 18-19 minutes and the recovery time for OpenStack's Swift was approximately 10 hours. This means that the recovery time for Compuverde Unstructured is approximately 30 times faster than that of OpenStack's Swift. One reason for this difference is that Compuverde uses multicasting whereas OpenStack's Swift uses DHT. Another reason could be that OpenStack uses the rsync[9] command that is responsible for maintaining object replicas, consistency of objects and perform update operations. It seems that using rsync command introduces a significant overhead which causes a performance decrease. The situation is similar for Compuverde Structured with a recovery time of 22 minutes compared to Gluster with recovery time of approximately 18.5 hours. Compuverde Structured recovery time is thus approximately 50 times faster than Gluster recovery time. As discussed before, Gluster uses DHTs instead of multicasting. Gluster also uses rsync for replication. Another reason for the low performance of Gluster compared to Compuverde Structured is the architecture that is used by Gluster for replication. In Gluster the proxy servers are doing the self-healing while in Compuverde Structured storage nodes are performing the self-healing by themselves without involving any proxy servers which results in a many-to-many replication pattern.

Table 2: Recovery test results

Compuverde Unstructured	19 minutes (1140 s)	18 minutes (1080 s)
Compuverde Structured	22 minutes (1320 s)	22 minutes (1320 s)
OpenStack	9 hours 27 minutes (34020 s)	10 hours 16 minutes (36960 s)
Gluster	18 hours 27 minutes (66420 s)	18 hours 29 minutes (66540 s)

6 Discussion and Related Work

Compared to conventional centralized storage systems, a distributed storage system allows for not only increased performance and redundancy, but also affords improved energy efficiency and lowering the carbon footprint of the system. For instance, by removing the need for a central, high-powered storage controller and replacing it with low wattage storage nodes, such as the ones used in the experiments presented in this paper. Furthermore, a distributed storage systems built from standard hardware components also makes it possible to exchange the individual nodes with nodes with a lower carbon footprint as technology advances. Reducing the carbon footprint and enabling green computing are two important aspects of Cloud Computing.

In recent years, many research and development efforts have been done in cloud computing, specifically on distributed file systems. In [24] the authors have done a performance comparison between several distributed file systems such as Hadoop, MooseFS (MFS) and Lustre. They have compared functional-

[9] rsync is a file transfer program for Unix-like systems.

ities as well as I/O performance of these three file systems.

In [12] the authors have done a performance comparison between Google File System (GFS) and MFS in terms of reliability, file performance and scalability. According to their comparison GFS and MFS are both reliable since resource files are backed up. But they found a single point of failure in master on GFS while it does not exist on MFS. In MFS there is a need for manual backup after a problem has occurred. Their comparison of the file performance indicates that GFS is used for large GB file size while MFS supports small files better.

7 Conclusion

We have compared two unstructured storage systems for Cloud Computing (Compuvede Unstructured and Openstack's Swift) and two structured storage systems for Cloud Computing (Compuverde Structured and Gluster). Compuverde uses multicasting and Openstack's Swift and Gluster use Distributed Hash Tables (DHTs). The architectural advantage of DHTs compared to multicasting is that we do not need to broadcast requests; the hash table gives us the address of the nodes that store the requested data and we avoid communication overhead. However, the obvious disadvantage with DHTs is that we need to run a hash function to obtain the address of the data, which introduces processing overhead. This means that the architectural decision, whether to use DHTs or multicasting will introduce different kinds of overhead: processing overhead for DHTs and communication overhead for multicasting.

We have compared the performance using a large storage system and realistic workloads, including the well-known Spec2008sfs test tool. Our experiments show that Compuverde has higher performance than the two systems that use DHTs. The performance advantage of Compuverde is particularly clear when the number of clients that issue simultaneous accesses to the system is high, which is typical in Cloud Computing. The performance advantage of Compuverde is not a result of caching in the storage nodes, i.e., the performance of Compuverde using NC is still significantly higher than that of the other two systems. We believe that the main reason for the higher performance is that Compuverde uses multicast instead of DHTs. The communication overhead introduced by multicasting does obviously not affect the performance as negatively as the processing overhead introduced by DHTs.

The recovery tests show that Compuverde recovers from a storage node failure much faster than OpenStack's Swift and Gluster. Again, we believe that the use of multicast instead of DHTs is the main reason. However, this cannot be the only reason for the significant difference in recovery times. One additional reason for Gluster to perform slower than Compuverde Structure could be that Gluster involves proxy servers in self-healing while Compuverde uses the many-to-many replication pattern and only involves storage nodes in self-healing. Another reason could be that Compuverde has built its own recovery protocol from scratch, whereas OpenStack's Swift and Gluster base their protocols on existing applications (e.g., rsync). Moreover, the processor utilization for Gluster never exceeds 50 percent, even for high loads. This indicates that there are internal performance bottlenecks in Gluster, which probably contributes to the relatively long time for self-healing.

References

[1] Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Marc Unangst, Brent Welch, Jim Zelenka, and Bin Zhou. "Scalable Performance of The Panasas Parallel File System," *FAST'08 Proceedings of the 6th USENIX Conference on File and Storage Technologies*, USENIX Assosiation Berkeley, CA, USA, pp. 17-33, 2008.

[2] Amplidata, "Amplistor: Unbreakable Object Storage for Petabyte-Scale Unstructured Data," White Paper, April 2011.

[3] Joe Arnold, Dr. Jaesuk Ahn, and Dr. Jinkyung Hwang, "Commercialization of OpenStack: Object Storage," OpenStack Conference Commercialization of Object Storage, Korea, April 2010.

[4] Andrew Beekhof, Christine Caulfield, and Steven C. Dake, "The Corosync Cluster Engine," *Proceedings of the Linux Symposium*, Ottawa, Ontrio, Canada, July, pp. 85-99, 2008.

[5] Angelos Bilas, Michail D. Flouris, Yannis Klonatos, Thanos Makatos, and Manolis Marazkis, "Using Transparent Compression to Improve SSD-Based I/O Caches," *EuroSys'10 Proceedings of the 5th European Conference on Computer Systems*, ACM NewYork, NY, USA, pp. 1-14, 2010.

[6] Scott A. Brandt, Darrell D. E. Long, Carlos Maltzahn, Ethan L. Miller, and Sage A. Weil, "Ceph: A Scalable, High-Performance Distributed File System," University of California, Santa Cruz, *Proceeding of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*, pp. 307-320, November 2006.

[7] Caringo CAStor, "CAStor: The Market Leading Object Storage Engine," Product Brief, September 2011.

[8] Caringo CAStor, "Castor the Market Leading Object Storage Engine" [Online], Available: http://www.caringo.com/downloads/datasheets/Caringo-CAStor-Object-Storage, pdf, September 15, 2011.

[9] Lei Chai, Ranjit Noronha, Dhabaleswar K. Panda, and Thomas Talpey, *Designing NFS with RDMA for Security, Performance and Scalability*, Technical Report OSU-CISRC-6/07-TR47, The Ohio State University, 2007.

[10] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "BigTable: A Distributed Storage System for Structured Data." *Journal: ACM Transactions on Computer Systems (TOCS)*, New York, USA, Volume 26, Issue 2, Number 4, June 2008.

[11] Robert Chansler, Hairong Kuang, Sanjay Radia, and Konstantin Shvachko, "The Hadoop Distributed File System " Yahoo! Sunnyvale, California USA, 2010.

[12] Ping Chen, Xuerong Gou, and Jianwei Li, "Research of Distributed File System Based on Massive Resource and Application in the Network Teaching System," *Proceedings of the International Conference on Advanced Intelligence and Awareness Internet*, Beijing, China, pp. 154-158, 2011.

[13] Santa Clara, "Amplidata Demonstrates Highly Scalable and Reliable Storage Solutions for Massive Cloud Deployments at Intel Development Forum," Article at PRNewswire, Calif., September 2011.

[14] F. Curbera, M. Duftler, R. Khalaf, N. Mukhi, W. Nagy, and S. Weerawarana, "Unraveling the Web Services web: An Introduction to SOAP, WSDL, and UDDI," *Proceedings of the IEEE Internet Computing*, NY, USA, pp. 86-93, 2002.

[15] Julian Dymcek, *Survey of Distributed Hash Tables*, Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV, 2011.

[16] EMC Atmos, "EMC Atmos Cloud Optimize Storage for Web Services," Whitepaper, April 2010.

[17] Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, and Julian Satran, "Object Storage: The Future Building Block for Storage Systems," 2nd International IEEE Symposium on Mass Storage Systems and Technologies, Sardinia, 2005.

[18] Bin Fan, Garth Gibson, Wittawat Tantisiriroj, and Lin Xiao, "DiskReduce: Replication as a Prelude to Erasure Coding in Data-Intensive Scalable Computing," Parallel Data Laboratory, Carnegie Mellon University, Pittsburgh, PA, 2011.

[19] Roy T. Fielding and Richard N. Taylor, "Principles Design of the Modern Web Architecture," *ICSE'00 Proceedings of the 22nd International Conference on Software Engineering*, ACM New York, NY, USA, pp. 115-150, 2000.

[20] William Geisel, "Tutorial on Reed-Solomon Error Correction Coding," Technical Memorandum, NASA, TM-102162, August 1990.

[21] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System," 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2006.

[22] Garth A. Gibson and Rodney Van Meter, "Network Attached Storage Architecture," *Magazine, Communications of the ACM*, New York, November 2000.

[23] Gluster Inc. "An Introduction to Gluster Architecture" Whitepaper, 2011.

[24] Wu Hao and Bai Songlin, "The Performance Study on Several Distributed File Systems," *Proceedings of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, Beijing, China, pp. 226-229, 2011.

[25] Michael Jakl, *REST: Representational State Transfer*, University of Technology Vienna, 2008.

[26] Roberto Lucchi and Michel Millot, "Resource Oriented Architecture and REST," JRC Scientific and Technical Reports, European Communities, Luxembourg, 2008.

[27] Shunsuke Mikami, Kazuki Ohta, and Osamu Tatebe, "Using the Gfarm File System as a POSIX Compatible Storage Platform for Hadoop MapReduce Applications", *GRID'11 Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, IEEE Computer Society Washington, DC, USA, pp. 181-189, 2011.

[28] OpenStack, "OpenStack Object Storage: An Overview," White paper, 2010.

[29] OpenStack, LLC, "Welcome to Swift's Documentation!" Swift v1.4.8-dev Documentation, 2011.

[30] OpenStack Object Storage Admin Manual, OpenStack, "Consideration and Tunning," 2011.

[31] OpenStack, *OpenStack Compute Admin Manual* Manual, November 2011.

[32] Sarp Oral, Galen Shipman, and Feiyi Wang, "Understanding Lustre FileSystem Internals," Technical report, Oak Ridge National Laboratory (ORNL), Center of Computational Science, USA, 2009.

[33] Liuis Pamies and I. Juarez, *On the Design and Optimization of Heterogeneous Distributed Storage Systems*, PHD Thesis, Department of Engineering Information in Mathematic, University Rovira in Virgili, July 2011.

[34] Dhabaleswar K. Panda and Ranjit Noronha, *IMCa: High Performance Caching Front-end for GlusterFS on InfiniBand*, Network-Based Computing Laboratory, Computer Science and Engineering, The Ohio State University, 2008.

[35] George Parisis, "DHTbd: A Reliable Block-Based Storage System for High Performance Clusters," *Proceedings of the IEEE/ACM CCGRID*, UK, pp. 392-401, 2011.

[36] Ken Pepple, "Deploying OpenStack", O'Reilly Media, ISBN 1449311059, August 2011.

[37] Drew Robb, "Gluster Brings Open Source to Unstructured Data," Storage Technology Features Article Published, August 2010.

[38] P. Wang, "IP SAN- from iSCSI to IP-addressable Ethernet Disks," Mass Storage Systems and Technologies, *Proceedings, 20th IEEE/11th NASA Goddard Conference*, pp. 189-193, 2003.

Sogand Shirinbab (photo not available) is a Ph.D. student at the School of Computing at Blekinge Institute of Technology. She received her M.S. from Kristianstad University in Embedded Systems. She is currently contributing in a project on virtualized and cloud-based test environments.

Lars Lundberg (photo not available) has a Ph.D. in Computer Science from Lund University. He has more than 14 years as full professor in Computer Systems Engineering and is currently the head of research at the School of Computing at

Blekinge Institute of Technology. He has worked with performance and capacity issues in a number of projects. One of his main interest areas is efficient resource allocation, and he has developed a number of results in real-time scheduling for both single- and multi-processors. Professor Lundberg is currently leading a project on virtualized and cloud-based test environments. Dr. Lundberg has published more than 100 articles in peer reviewed journals and conferences, and been the advisor for 11 Ph.D. students that have taken their doctoral degrees.

David Erman (photo not available) received his Ph.D. degree in Telecommunication Systems at Blekinge Instititute of Technology in March 2008, where he is currently employed as a senior lecturer. His academic background is in signal processing and computer science, before turning to computer networking, mainly focusing on P2P systems and overlay networks. His research interests include distributed systems, large-scale media distribution and streaming, P2P communication, overlay networking, network coding, virtualization, cloud computing and cognitive networking. David has participated in several national and international research projects.

Budget Constrained Dataflow Scheduling for Minimized Completion Time on the Cloud

Dabin Ding*, Fei Cao*, Dunren Che*, Michelle M. Zhu*, and Wen-Chi Hou*
Southern Illinois University, Carbondale, Illinois 62901, USA

Abstract

Cloud computing provides high-end computing capabilities so that users can access data and applications anywhere in the world on demand and pay for what they use. It is emerging as a promising computing paradigm for large-scale data intensive queries, which are usually modeled as complex Directed Acyclic Graph (DAG)-structured data processing dataflows with arbitrary data operators as nodes and producer-consumer interactions as directed edges. The optimization problem of scheduling dataflows on the Cloud is a very complex and challenging task which is similar to query optimization. Optimization must satisfy a variety of objectives and constraints, while taking into account the particular characteristics of the underlying Cloud environment. In addition to achieving minimum query completion time, the commercialization of Clouds requires policies to take users' economic concerns as well. In this paper, we formulate scheduling of dataflows onto Cloud resources toward the objective of minimizing the query completion time under certain budget constraint. A heuristic scheduling algorithm, Layer-oriented Resource Allocation within Budget constraint (LRA-$\mathscr{B}$) is proposed and evaluated. Experiments are conducted on numerous dataflows and Cloud environment configurations, and the overall results are quite promising and indicate the effectiveness of our algorithm.

Key Words: Cloud computing, dataflows, scheduling, query completion time, budget constraint.

1 Introduction

Complex on-demand data retrieval and processing combining the notions of query & search, information filtering & retrieval, data transformation & analysis, and other data manipulations [14] are typically represented by DAG-structured data processing graphs (i.e., dataflows) whose nodes are arbitrary data operators and directed edges are producer-consumer interactions. Assume that terabytes of aerial imagery have been collected for intelligence purposes and algorithms to detect tanks, planes, or missile silos are available, it is a complex and time-consuming task to find these weapons if run in a conventional manner. This query could be expressed in SQL as follows:

SELECT count(Tanks), count(Planes), count(Missiles)
FROM Raw_Aerial_Imagery **AND** GPS_Signal **AND** WorldMap
WHERE [Analytical Requirements]
GROUP BY Location

The SQL query is optimized [15] and transformed into an execution plan represented as a DAG-structured dataflow. Scheduling the dataflow graph onto the resources of the underlying distributed environment (i.e., Grid, Cloud, etc.) is a well-known NP-complete problem [12]. Moreover, the heterogeneity and dynamic status of distributed environments complicate the scheduling optimization problem in order to achieve objectives such as the completion time and monetary cost.

Cloud computing has attracted much attention from the research community [1] that evolved from a paradigm of basic IT infrastructures to Grid computing, and to resource provisioning services: infrastructures (IaaS), platforms (PaaS), and software (SaaS) [11]. Meanwhile, Cloud computing data centers are becoming increasingly popular for providing high-end computing capabilities to end users as pay-as-you-go services. Clouds offer their users the ability to lease resources as long as needed, and charge based on a per time quantum pricing policy. Moreover, data centers are making heavy use of virtualization which allows a single server to run multiple operating instances simultaneously [28] to achieve efficient computing resource usage. A Virtual Machine (VM) is a software based machine emulation technique that executes other software in the same manner as the physical machine for which the software is developed and executed [23]. The normal process of a data center operating with the use of VMs for executing jobs (e.g., dataflows) is shown as follows:

(1) A data center provides various VM templates.
(2) When a job arrives at the data center, the scheduler allocates the job with pre-configured VMs then starts it on

*Department of Computer Science, Email: {dabin, vicky}@siu.edu and {dche, mzhu, hou}@cs.siu.edu

proper servers.

(3) The job is executed in the VMs.

(4) After the job finishes execution, the VMs are shutdown.

To run dataflows on Clouds, dataflow characteristics (e.g., execution time of operators, amount of data generated, etc.), Cloud network characteristics (e.g., bandwidth, etc.), Cloud pricing policies, and more need to be considered. The optimal trade-off between Quality of Service (QoS) and money spent depends on the needs of the particular user concerned. Scientific dataflow applications usually have the primary objective of optimizing the completion time which depends on both the data transfer time involved in staging the input and output data and the computation time to execute them. However, users with budget or quota constraints may not always desire the highest possible level of QoS such as completion time.

Motivated by the above practices and concerns, we focus on developing a dataflow scheduling algorithm on the Cloud based on both time and money, namely, how to minimize completion time under a budget constraint.

The key contributions of this paper are:

(1) Complex DAG-structured dataflow model intermixed with different operator types.

(2) Novel time modeling with different operator types and dynamic Cloud resource consideration.

(3) Novel monetary cost modeling considering both execution cost and data transfer cost.

(4) Time-dependent virtual machine allocation policy.

(5) Comprehensive comparison using various experimental setups to show the effectiveness of our algorithm.

The paper is organized as follows. Section 2 gives an overview of related works. Section 3 conducts analytical models and Section 4 formulates the scheduling problem. In Section 5, our scheduling algorithm design is described in details. Section 6 explains the evaluation methodology, simulation setup and the analysis of results. Section 7 presents the conclusion and future work.

2 Related Works

Typically, some middleware are used to execute user-defined code in distributed environments [27]. The Condor/DAGMan/Stock set [17] is a representative technology of High Performance Computing. It is a robust and easily scalable mechanism for exploiting extensive scientific infrastructures of mostly computational resources due to its scheduling, monitoring and failure resilience capabilities. Condor [24, 4, 5] is a specialized workload management system for compute-intensive jobs and is designed to harvest CPU cycles on idle machines. Directed Acyclic Graph Manager (DAGMan) [5] is a meta-scheduler for Condor jobs which manages dependencies between jobs at a higher level than the Condor Scheduler. Running data-intensive workflows with DAGMan is very inefficient [27, 21]. Many systems such as Pegasus [8] and GridDB [18] use DAGMan as middleware. Extensions of Condor to deal with data-intensive workflows have been proposed [21], but they have not been materialized yet to the best of our knowledge.

Middleware technologies such as Pegasus Workflow Manage System [8], Gridbus Workflow Management System [29] and so forth, are used to schedule the DAG-structured workflows onto the distributed environments. Pagasus supports a higher level of abstraction for both data and operations, and maps workflows onto the Cloud to generate executable workflows using a clustering approach to group short duration tasks as a single task in order to reduce data transfer overhead and number of VMs created. Therefore, it offers true optimization features, as opposed to simple matching of operators to a fixed set of resources. Nefeli [26] is a Cloud gateway that uses deployment hints for efficient execution of workloads, being aware of the resources and the actual locations of VMs. However, this information may not be generally available especially in commercial Clouds. Hadoop is a popular platform that follows the Map-Reduce [6] paradigm to achieve fault-tolerance and massive parallelism [27]. It is being used in companies like Yahoo, Facebook, etc. to store and process extremely large data sets on commodity hardware [25]. However, the Map-Reduce programming model is very low level that requires developers to write custom programs. Therefore, several high level query languages have been developed on top of Hadoop, such as Hive [25] and PigLatin [19]. Hive supports queries expressed in a SQL-like declarative language (i.e., HiveQL), which are compiled into Map-Reduce jobs that are executed using Hadoop. In addition, HiveQL enables users to plug in custom Map-Reduce scripts into queries [25].

Cloud computing environments facilitate applications by providing virtualized resources that can be dynamically provisioned [20]. Clouds are primarily driven by economics, the pay-per-use pricing model is very appealing for both Cloud providers and users [16]. However, dataflow applications may incur large data retrieval and monetary cost when they are scheduled taking into account only the completion time. Therefore, in addition to optimizing completion time, data transfer costs between resources as well as execution costs must also be taken into account. There are several efforts that move in the same direction as our work but solve a simpler version of the problem. Kllapi et al [14] studied the space of alternative schedules that arose from the optimization problem between completion time and monetary cost, and investigated the time-money trade-off for different types of dataflows and Cloud environments based on greedy and exhaustive algorithms. In [20], Pandey et al. presented a particle swarm optimization based heuristic to schedule general dataflows with one-dimensional weighted average parameter of several metrics as the optimization criterion. Silva et al. proposed a heuristic optimization of independent tasks (no communication between tasks) having the number of resources that should be allocated to maximize speedup as the optimization criterion with a given predefined budget [22]. Moreover, parallelism and resource sharing models for optimal scheduling of relational operators of query execution plans with time-shared (e.g., CPUs, disks, etc.) and space-shared (e.g., memories) resources and communications are generalized to arbitrary operators [27, 10]. Our difference with the aforementioned efforts falls on the following aspects: (i) A new methodology, layer-oriented resource allocation algorithm, is adopted to solve the scheduling problem. (ii) A new time modeling in accordance with dynamic virtual machine allocation policy in Cloud infrastructure is considered. (iii) a more thorough experiment is conducted to study the impact of different factors on our scheduling algorithm. Those factors include data center and dataflow size, operator types, data transfer sizes and computing and link unit cost.

3 Analytical Models

We construct the dataflow scheduling model as the dataflow operator graph and the underlying Cloud environment (i.e., data center) to facilitate the mathematical formulation of the scheduling problem.

A dataflow is constructed as a Directed Acyclic Graph (DAG) $G(ops, flows)$. Vertices represent arbitrary concrete operators (ops) and edges represent data transferred between two operators ($flows$). An operator in ops receives a data input from each of its preceding operators, and is modeled as $op(exec, tran, Z, type)$, where $exec$ is the execution time of an operator, $tran$ is the data transfer time between two connected operators, Z denotes the aggregated and complexity normalized input data size, and $type$ is a flag either equal to $pipeline$ (PL) or $store\text{-}and\text{-}forward$ ($S\&F$). PL type (e.g., from databases, $select$ operator) means execution can start as soon as some data inputs from its preceding operators ($producer$) is available, whereas $S\&F$ type (e.g., from databases, $sort$ operator) means execution cannot start until all data inputs from its preceding operators ($producer$) arrive. A flow between two operators, $producer$ and $consumer$ [14], is modeled as $flow(producer, consumer, data)$, where $data$ is the size of data transferred. To generalize our model, if a dataflow has multiple starting or ending operators, a virtual starting or ending operator of complexity zero can be created and connected to all starting or ending operators without any data transfer along the edges. The parameters of a dataflow are given in Table 1.

The Cloud environment (i.e., a data center) is where the VMs will be reserved, deployed and run on physical servers. We consider a general Cloud environment where both prior VM reservation and on-demand requests are supported. Thus, our resource allocation status for a data center is time-dependent, which means that the available resources of each server and bandwidth of each network link are changing from time to time due to the in-advance reservation requests. The parameters of a Cloud network are given in Table 2.

For general purposes, we construct a data center as a complete network graph $G(servers, links)$ consists of a set of servers and network links. A server in $servers$ is modeled as $server(cpu, vms)$, where cpu is its computing power and vms is the set of VMs allocated on the server. A VM is modeled as $vm(p, t_{start}, t_{shut}, size)$, where p is its computing power, t_{start} and t_{end} are its start time and shut down time, respectively, and $size$ is the size of the VM. We assume 5 different sizes of VMs: small, small-medium, medium, medium-large, and large which consume 20%, 40%, 60%, 80%, and 100% of their allocated server's capacity, respectively. The unit execution price of a VM depends on its size, the larger the size, the higher the charge. A link between two servers is modeled as

Table 1: Parameter of a dataflow model

Parameters	Definitions
$G(ops, flows)$	dataflow
ops	set of arbitrary concrete operators
$flows$	data transferred between two operators
$op(exec, tran, Z, type)$	operator
$exec$	execution time of an operator
$tran$	data transfer time between two connected operators
$type$	operator type
Z	aggregated and complexity normalized input data size
$flow(producer, consumer, data)$	a $flow$ between two operators (producer & consumer)
$data$	size of data transferred between two connected operators (producer & consumer)

Table 2: Parameter of a Cloud network model

Parameters	Definitions
$G(servers, links)$	the Cloud environment
$servers$	set of servers in a data center
$links$	network links in a data center
$server(cpu, vm)$	a server in a data center
cpu	computing power of a server
vms	set of VMs allocated on a server
$vm(p, t_{start}, t_{shut}, size)$	a VM allocated on a server
p	computing power of a VM
t_{start}	the start time of a VM
t_{end}	the end time of a VM
$size$	size of a VM
$link(bw, delay)$	network link between two servers
bw	network bandwidth
$delay$	minimum link delay
ξ_M	unit executing price of a VM M ($/hour)
λ_{XY}	unit executing price of network link from server X to server Y ($/hour)

$link(bw, delay)$ where bw is the network bandwidth between them, and $delay$ is the minimum link delay.

A schedule S_G of a dataflow G is an assignment of its operators onto VMs on servers in a data center (Figure 1). An assignment is modeled as $assign(op, server)$. Time $T(S_G)$ and cost $C(S_G)$ denote the completion time and the monetary cost of a schedule S_G of a dataflow G. The parameters of a schedule are given in Table 3.

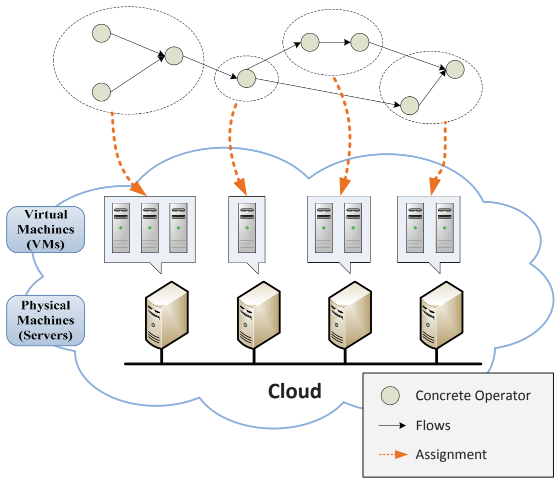


Figure 1: Dataflow scheduling with assignment of operators to virtual machines on cloud servers

Table 3: Parameter of a schedule

Parameters	Definitions
$assign(op, server)$	assignment of op to $server$
$T(S_G)$	completion time of a schedule S_G
$C(S_G)$	monetary cost of a schedule S_G

3.1 Time Modeling

As usable Cloud resources are dynamic, the allocable computing power of server X at time t is represented as $cpu_{X,t}$. Whenever the allocable capacity of a particular server changes, a new time t' will be used. In other words, the computing power during time slot $[t, t']$ is constant. For example, during time interval $[t_1, t_n]$, the allocable computing power can be set is $CPU_{X,t_1,t_n} = (cpu_{X,t_1,t_2}, cpu_{X,t_2,t_3}, ..., cpu_{X,t_{n-1},t_n})$. The maximum computing power we can reserve during time interval $[t_1, t_n]$ will be the minimum value of all time slots: $cpu_{X,t_1,t_n} = min(CPU_{X,t_1,t_n})$. As in the Cloud environment, the provider always has some types of pre-configured VMs, so the largest VM we can allocate on a server should be the largest one which is smaller than cpu_{X,t_1,t_n}, defined as $p_{X,t_1,t_n} = max(p_{X,t_1,t_n} < cpu_{X,t_1,t_n})$.

Figure 2 shows an example of three VMs scheduled on one server during different time slots. For example, VM1 reserves 60% of the server's general capacity from t_0 to t_2; VM2 reserves 20% from t_1 to t_4; VM3 reserves 40% from t_3 to t_4. The available computing power of this server from t_0 to t_4 will be sets of $P_{X,t_0,t_4} = (40\%, 20\%, 80\%, 40\%)$, thus the maximum allocable computing power from t_0 to t_4 would be 20%. Similarly, the available bandwidth of a network link is defined in the same way: the maximum link bandwidth bw_{XY,t_1,t_n} during time interval t_1 and t_n will be $min(BW_{XY,t_1,t_n})$ where $BW_{XY,t_1,t_n} = (bw_{XY,t_1,t_2}, bw_{XY,t_2,t_3}, ..., bw_{XY,t_{n-1},t_n})$.

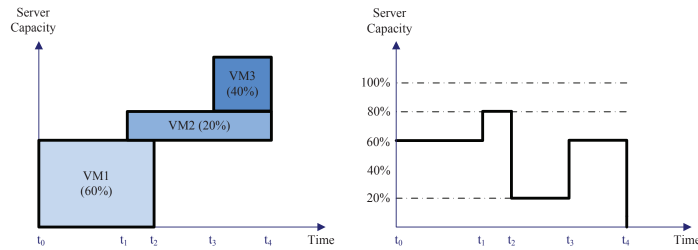


Figure 2: Cloud server allocated capability during the time interval $[0, t_4]$

The network communications which perform data transfers are injected between the operators of a $flow(producer, consumer, data)$ if $producer$ and $consumer$ are assigned to different servers. According to [14], always two data transfers are injected, one attached after $producer$, and another attached before $consumer$. To calculate the completion time of a dataflow, two types of operators, namely PL and $S\&F$ must be addressed separately as discussed in the following paragraphs.

Pipeline: Let A and B be two connected PL operators with $flow(A, B, D_{A \to B})$ (where B's preceding operators are all PL operators). We assume that the execution time of both operators are fully overlapped. Let the assignments of A and B be $assign(A, X)$ and $assign(B, Y)$, respectively, with $X \neq Y$. Let Z_A denote the aggregated and complexity normalized input data size on A,

(1) the execution time of A during time interval $[t_1, t_n]$ is computed as:

$$exec_{A,X,t_1,t_n} = \frac{Z_A}{p_{X,t_1,t_n}} \quad (1)$$

(2) the data transfer time, which is injected into the execution of A at various time slots $[t_p, t_q]$ during time interval $[t_1, t_n]$ as shown in Figure 3 is computed as:

$$tran_{AB,XY,t_1,t_n} = \sum_{\forall [t_p,t_q] \in [t_1,t_n]} \left(\frac{D_{A \to B}}{bw_{XY,t_p,t_q}} + delay_{XY}\right) \quad (2)$$

(3) the running time of A is:

$$t_{A,t_1,t_n} = max(A.exec + tran, B.exec + tran) \quad (3)$$

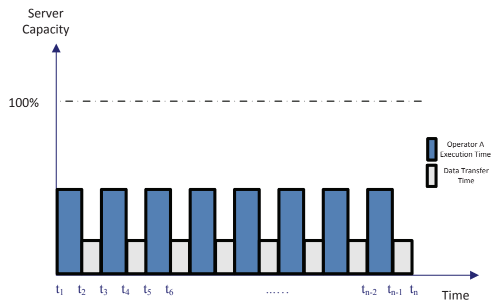


Figure 3: PL operator A intermixed with data transfer time during the time interval $[t_1, t_n]$

Store-and-Forward: Let A be a $S\&F$ operator with $assign(A, X)$. For every operator B (either PL operator or $S\&F$ operator) with $flow(A, B, D_{A\to B})$ and $assign(B, Y)$ with $X \neq Y$,

(1) the execution time of A during time interval $[t_1, t_n]$ is computed as:

$$exec_{A,X,t_1,t_n} = \frac{Z_A}{p_{X,t_1,t_n}} \tag{4}$$

(2) the data transfer time, which is attached after the execution of A as shown in Figure 4, is computed as:

$$tran_{AB,XY,t_1,t_n} = \frac{D_{A\to B}}{bw_{XY,t_1,t_n}} + delay_{XY} \tag{5}$$

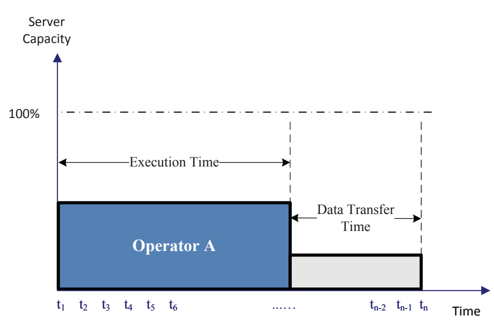


Figure 4: $S\&F$ operator A with data transfer time attached during the time interval $[t_1, t_n]$

(3) the running time of A is:

$$t_{A,t_1,t_n} = A.exec + tran \tag{6}$$

3.2 Cost Modeling

Cloud providers lease resources that are typically charged based on a per time quantum pricing policy which is typically one hour [14], and Cloud resources are charged for exactly the time being used. We define the total monetary cost $C(S_G)$ of a schedule S_G as the sum of costs of executing each operator and the sum of costs of all the data transfers:

(1) Cost of executing operator A on VM M:

$$C_{exec}(A) = \xi_M \times exec(A) \tag{7}$$

Note that VMs with different sizes have different unit execution prices.

(2) Cost of data transfer of $flow(A, B, D_{A\to B})$ from VM M located on server X to VM N located on server Y:

$$C_{tran}(D_{A\to B}) = \lambda_{XY} \times tran(AB) \tag{8}$$

(3) Total monetary cost $C(S_G)$ of a schedule S_G:

$$C(S_G) = \sum_{\forall A \in ops} C_{exec}(A) + \sum_{\forall D_{A\to B} \in flows} C_{tran}(D_{A\to B}) \tag{9}$$

4 Problem Formulation

4.1 Query Language Abstractions

Generally, user requests take the form of queries in some high-level declarative or visual language such as SQL, Hive [25], etc. The optimization process examines all execution plans that could answer the original query(s) and choose the one that is optimal and satisfies user's quality of service requirements. As introduced in [27], our optimization process has the following three different layers of abstractions:

Operator Graphs: These are the query(s) decomposed into data operators as nodes, and operator interactions in the form of producing and consuming flows as directed edges. Operators encapsulate data processing algorithms and could be custom-made by end users. These processing algorithms include compositions, aggregations and partitions, and be more specific like filtering, ranking, sorting and so on.

Concrete Operator Graphs: Similar to operator graphs except that their nodes are concrete operators, i.e., software components that implement operators in a particular way and carry all necessary details for their execution. For this layer, the critical step is to determine an operator's implementation. In general, there might be multiple alternative implementation for an operator, e.g., a fast but limited by memory version and a slow but only limited by disk size one. A more specific example is the $JOIN$ operator, which has multiple implementations: $hash$ join has short execution time but limited by memory; $nested - loops$ join has little memory consumption but long execution time.

Execution Plans: Similar to concrete operator graphs except that their nodes are concrete operators that have been allocated resources for execution and have their initialization parameters set. The modeling and methodology presented in this paper belong to this stage of optimization. The main focus is to allocate the resource needed for execution of operators and flows.

4.2 The Scheduling Problem

We define the scheduling problem as follows:

Definition 1. *Cloud users can submit dataflow applications G (e.g., queries) from anywhere around the world. Our objective is to find the scheduling such that the completion time of the dataflow $T(S_G)$ is minimized within a pre-specified financial budget constraint (Figure 5).*

$$\min_{all\ possible\ schedules} T(S_G),\ such\ that\ C(S_G) \leq budget \tag{10}$$

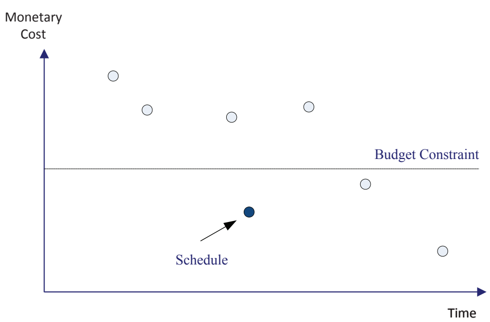


Figure 5: The optimization problem (the chosen schedule is shown with an arrow)

5 Algorithm Design

We develop a Layer-oriented Resource Allocation within Budget constraint (LRA-$\mathscr{B}$) for budget constrained users. The general steps of our algorithm are as follows:

Step 1. Create Virtual Execution Plan (VEPlan).
Step 2. Adjust VEPlan to satisfy the budget constraint if necessary.
Step 3. Map the VEPlan to the Cloud, and generate VM allocations.

The pseudocode of LRA-$\mathscr{B}$ is provided in Algorithm 1 and the details of the algorithm will be discussed in the following sections.

5.1 Virtual Execution Plan

A VEPlan is a virtual schedule for the execution of the given dataflow based on the basic configuration of the Cloud environment, such as the node power, link bandwidth, etc. It is called virtual because it does not consider the mapping to VMs and creating of VM allocations. For example, after acquiring the VM power, we can calculate the time required to compute each operator, and by acquiring the link bandwidth and link delay, we can calculate the data transfer time of each link. With this information, an execution plan is constructed and query completion time and cost can be estimated. As the Cloud environment usually provides different VM templates, the VEPlan can be constructed with different VM templates, which leaves the space for adjusting VEPlan to satisfy the budget constraint. A Max-VEPlan is a virtual execution created with the maximum processing power, i.e., 100% of server capacities. On the contrary, a Min-VEPlan is created with the minimum processing power, i.e., 20% of server capacities. A Min-VEPlan gives out the longest query completion time and usually requires least cost for estimation purpose. When the VEPlan is mapped onto the actual servers in the Cloud, the final query completion time and cost might vary a lot from the estimation got from this step: firstly, operators may share servers which save the data transfer cost; secondly, when an operator is scheduled to be executed in a certain time slot, the query completion time might be prolonged due to time slot unavailability. The technique we adopted to make precise estimation of the cost of the VEPlan will be discussed in the following section.

Algorithm 1 LRA-$\mathscr{B}$

Input:
$G(ops, flows)$: The dataflow graph
$G(servers, links)$: The Cloud environment
$budget$: The budget constraint

Output:
S_G: The schedule that minimizes the $T(S_G)$ under budget constraint

1: $C(S_G) = createMinVEPlan()$;
2: **if** $C(S_G) > budget$ **then**
3: **return** NO_POSSIBLE_SCHEDULE;
4: **end if**
5: $C(S_G) = createMaxVEPlan()$;
6: **if** $C(S_G) > budget$ **then**
7: $adjustVEplan()$;
8: **end if**
9: $mapVEPlan()$;

5.2 Cost Estimation

A Min-VEPlan is used to estimate the minimum cost required to execute the dataflow. If the budget is lower than the minimum cost, then there is no possible schedule within this budget (Line 1-4 in Algorithm 1). The ideal case is that the estimated cost of the VEPlan is the same as the actual execution cost. However, it is impossible since the scheduling problem is NP-Hard.

Proposition 1. *To achieve better query completion time, an operator with indegree > 1 can share the server with at least one of its preceding operators to save the data transfer time.*

By taking the savings on the data transfer time into consideration, to make the estimation more precise, the cost of at least one of data transfer cost for each operator with indegree > 1 can be deducted from the estimated cost of the VEPlan.

5.3 Adjust Virtual Execution Plan

As shown in Algorithm 1, a Max-VEPlan will be created at the beginning. If the estimated cost is greater than budget, then a swapping procedure is invoked to adjust the VEPlan and lower the cost, as shown in Algorithm 2. The objective of this procedure is to reassign those operators on non-critical paths to result in the minimum increase in $T(S_G)$ for the largest cost savings under the budget limit. The operators on the critical path will not be reassigned. In each iteration, an operator is selected and the VM template is swapped to a lower one if available. The iteration ends up with a reduced total cost with similar $T(S_G)$. To determine the swapping strategy, $Increase$ for operator A as the iteration increase between the current and new configurations are computed in (11):

Algorithm 2 adjustVEPlan

Input:
$G(ops, flows)$: The dataflow graph
$G(servers, links)$: The Cloud environment
$VEP(ops, flows)$: The VEPlan
$Budget$: The budget constraint

Output:
$VEP'(ops, flows)$: The new VEPlan that satisfies the budget constraint

```
VEP' = VEP;
Calculate C(S_G)_Cur;
while C(S_G)_Cur > Budget do
   findCriticalPath(VEP');
   for all operator A ∈ G do
      if A ∈ CriticalPath then
         Continue;
      end if
      Calculate Increase_A by assigning A to smaller VM;
   end for
   A = min{Increase};
   VEP' = updateVEP(VEP', A.smallerVM());
   Calculate C(S_G)_Cur;
end while
return VEP';
```

$$Increase_A = \frac{T(S_G)_{New} - T(S_G)_{Cur}}{C(S_G)_{Cur} - C(S_G)_{New}} \tag{11}$$

where $T(S_G)_{Cur}$ and $C(S_G)_{Cur}$ are the query completion time and cost of current schedule, respectively; $T(S_G)_{New}$ and $C(S_G)_{New}$ are the query completion time and cost of A reassigned with a smaller VM size, respectively. The algorithm keeps reassigning by considering the smallest values of $Increase$. Our selection criteria of having large cost saving and small query completion time increase will result in small value of $Increase$.

5.4 Mapping Virtual Execution Plan to the Cloud

This procedure is to find the available resources on Cloud servers to assign dataflow operators. The goal is to achieve the minimum query completion while mapping the VEPlan generated in the previous step to the Cloud environment. It starts with a layer-oriented sorting and then schedules the operators layer-by-layer that optimizes the query completion time $T(S_G)$.

5.4.1 Layer-oriented Sorting. A layer-oriented sorting of a DAG is a linear ordering of its vertices constrained by the edge dependencies [3]. By applying layer-oriented sorting to the DAG-structured dataflow, we can separate operators into different layers starting from layer 1 based on happen-before dependencies and operator types. Operators in the same layer can be executed simultaneously. To decide which layer should an operator A belong to, as there are two types of operators:

Case 1: if A is a PL operator and all its preceding operators $pre(A)$ are PL operators, then A will be assigned to the same layer as its preceding operators, e.g., operator 3 in Figure 6;
Case 2: if A is a PL operator but at least one of its preceding operators $pre(A)$ is a $S\&F$ operator, then A will be assigned to the next layer, e.g., operator 9 in Figure 6;
Case 3: if A is a $S\&F$ operator then A will be assigned to the next layer, e.g., operator 7 in Figure 6;

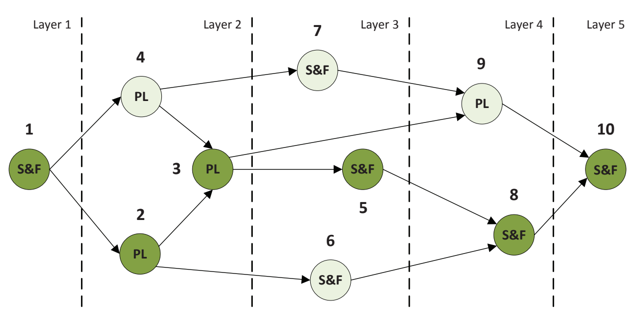


Figure 6: Layer-oriented sorting of DAG-structured dataflow

Such layer-based sorting can be done in linear time. Each operator will be given a priority value depending on their computing and communication loads. Operators on the CP (shown in dark shade in Figure 6) will be given the highest priority value compared with other operators from the same layer. If there are more than one CP operators in one layer, the higher priority gives to $producer$. An example is shown in Figure 6 (both operator 2 and 3 are CP operators, but higher priority will be given to operator 2).

5.4.2 The Resource Allocation Procedure. The resource allocation procedure seeks to assign operators to servers with the goal of minimizing query completion time. Operators starting from layer 1 will be scheduled on the appropriate VMs allocated on different servers with the lowest partial earliest completion time from the starting operator. If there are multiple starting/ending operators, a virtual starting/ending operator with zero complexity is inserted and connected to all starting/ending operators. The shaded operators in Figure 6 compose the CP. The order to schedule these operators is indicated by the numbers. Whenever we start to schedule operators from a new layer, operators with higher priorities will be scheduled first to have a better chance to utilize good resources.

Algorithm 3 mapVEPlan

Input:

$G(ops, flows)$: The dataflow graph
$G(servers, links)$: The Cloud environment
$VEP(ops, flows)$: The VEPlan
$Budget$: The budget constraint

Output:

S_G: The schedule that minimizes the $T(S_G)$ under budget constraint

1: Apply layer-oriented sorting to $G(ops, flows)$;
2: **for** i = layer 1 to $MaxLayer$ **do**
3: Sort operators in current with descending order of priority.
4: **for all** operator $A \in$ current layer **do**
5: **for all** server $S_k \in$ available servers **do**
6: Calculate partial query completion time $pT(S_G)_k$ and partial cost $pC(S_G)_k$ if A is assigned to S_k;
7: **end for**
8: Select the schedule(s) with minimum $pT(S_G)$, if there are several schedules with the same $pT(S_G)$, choose the one with minimum partial cost;
9: **end for**
10: **end for**
11: Calculate $T(S_G)$ and $C(S_G)$;
12: **return** $T(S_G), C(S_G)$;

In Algorithm 3, line 1 applies layer-oriented sorting to all operators. Lines 2-10 aim to schedule operators layer-by-layer. Line 3 sorts the operators in the current layer according to their priorities. For all the operators in the current layer, lines 5-7 seek to find the allocation that will give the best partial query completion time as well as minimum partial cost. For example, for operator A, the computing power requirement and time span needed can be obtained from the VEPlan and schedules for preceding operators. To allocate operator A to a server S, we need to find an appropriate time slot on S to satisfy the time requirement of A. After looping through all the severs, A will be assigned to the server that achieves the minimum partial query completion time. Line 11 calculates the final query completion time $T(S_G)$ and cost $C(S_G)$. Line 12 returns the $T(S_G)$ and $C(S_G)$.

6 Experimental Evaluation

In this section, we describe the overall experimental setup and the analysis of results.

6.1 Experimental Setup

6.1.1 Data Center and Dataflow Configurations. The experiments conducted are characterized by three elements:

Cloud Environment: In our experiments, we realistically assume that all the servers within a data center are homogeneous, i.e., they have the same resources (CPUs, memories, and network, etc.).

Dataflow Structure: There are several commonly used families of dataflows such as: Montage [13], Ligo [7], Cybershake [9] and more generally Lattice [14], etc. The first three are abstractions of actual dataflows that are used in real applications, and Lattice is a purely synthetic dataflow family that generalize the typical Map-Reduce dataflow [14]. To ensure the generality of our model, dataflows with multiple starting or ending operators can be converted to our general model (as discussed in Section 3) by creating a virtual starting or ending operator of complexity zero and connect it to all starting or ending operators without any data transfer along the edges.

We experiment with several sizes of dataflows which are represented by a two-tuple (m, n) in Table 4, where m is the number of operators, and n is the number of flows as defined in the dataflow model.

Table 4: Dataflow Configurations

Dataflow ID	Dataflow Size m, n
1	10, 21
2	20, 43
3	40, 79
4	50, 96
5	50, 500
6	100, 205
7	200, 438
8	300, 601
9	400, 784
10	500, 1138

We develop a dataflow generator to randomly generate our test dataflows. By giving two attributes of dataflows, m for number of operators and n for number of flows, our dataflow generator can automatically generate random dataflows with varying parameters within a suitably predefined range of values: (i) aggregated and complexity normalized input data size;

(ii) operator type; (iii) the number of flows and the data transfer size between two operators. This generator will ensure that each operator has at least one input edge and one output edge.

Operator Types: In our experiments, we examine complex dataflows with intermixed PL and $S\&F$ operators. For the dataflows with the same size, we examine the following five cases:

(1) all operators being $S\&F$ operators

(2) 25% operators being PL operators, and 75% being $S\&F$ operators

(3) 50% operators being PL operators, and 50% being $S\&F$ operators

(4) 75% operators being PL operators, and 25% being $S\&F$ operators

(5) all operators being PL operators

6.1.2 Performance Metrics and Experimental Scenarios.

The following performance metrics are considered:

- Dataflow query completion time
- Monetary cost

We evaluate our algorithm from the following experimental scenarios:

- Impact of data center size
- Impact of unit execution price of servers and network links
- Impact of operator types
- Impact of data transfer sizes
- Impact of budget constraint
- Impact of dataflow size

By default for plotting figures, if not specify otherwise, the dataflow used for demonstration is the 4th one G(50,96) in Table 4 and the data center contains 100 nodes. The ratio of server and link unit cost is set as 10:1.

6.2 Analysis of Results

6.2.1 Impact of Data Center Size To study the impact of data center size, we create two data centers of different sizes. Data center 1 contains 10 nodes and data center 2 contains 1000 nodes. Individual node and link capacities are the same in both data centers. Impact of data center size on query completion time (bar) and cost (plot) is given in Figure 7. We evaluate two dataflows running on two data centers. Case 1 and Case 2 are the results of dataflow 6 running on data center 1 and Case 2 is dataflow 6 running on data center 2. Case 3 is the result of dataflow 10 running on data center 1 and Case 4 is on data center 2. We can conclude from the figure that for smaller sized dataflows the performance is irrelevant to the size of data center, but for larger dataflows (e.g., dataflow 10 contains 500 operators compared to 100 in dataflow 6) the performance is better on larger data center although the costs are almost the same in different sized data centers for a same dataflow.

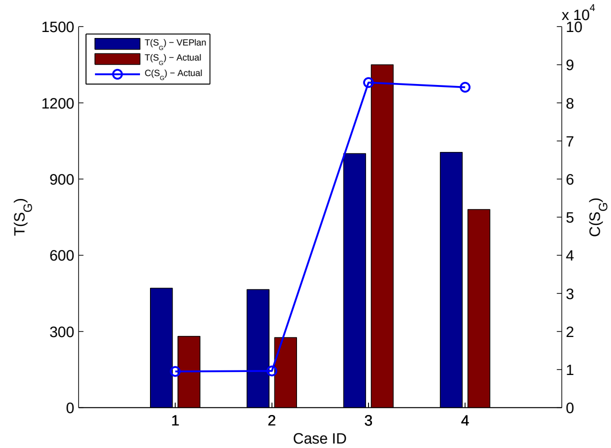


Figure 7: Impact of data center size

6.2.2 Impact of Unit Execution Price of Servers and Network Links. We consider the following seven different ratios of server unit execution prices vs. network link unit execution prices (as shown in Table 5). For all cases, the sum of server and link unit price remains the same. Impact of server and link price on query completion time (plot) and cost (bar) are given in Figure 8.

Table 5: Unit Execution Price Ratios: Server vs. Network link

Case ID	Ratio
1	1 : 100
2	1 : 10
3	1 : 5
4	1 : 1
5	5 : 1
6	10 : 1
7	100 : 1

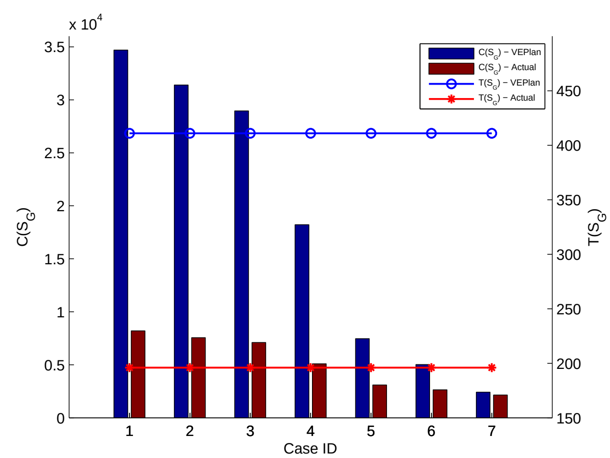


Figure 8: Impact of unit execution price of servers and network links (Dataflow ID = 4)

From the figure, we can see that the cost of dataflow is sensitive to link cost, as the ratio of link vs. server decreases, the total cost decrease. The difference between VEPlan and actual cost is huge for the first few cases, and decreases as the ratio of link vs. server decreases. The reason for this is the node sharing between operators, as the link transfer cost composes a great potion in VEPlan while it does not in actual plan. Meanwhile, the changing of ration have no effect on the completion time.

6.2.3 Impact of Operator Types. For the dataflows with the same size intermixed with different percentages of PL and $S\&F$ operators, we examine the five cases indicated in the experimental setup section. As we can observe from Figure 9, generally, the completion time decreases as percentage of PL increases, but the cost increases since the PL operators will consume more server time while they are waiting for data inputs from predecessors. For Case 4, where there are 75% of PL operators and 25% of $S\&F$, there is a huge increase for the actual cost and completion time. The few scattered $S\&F$ operators might be the bottleneck for execution and cause the increase in time and cost.

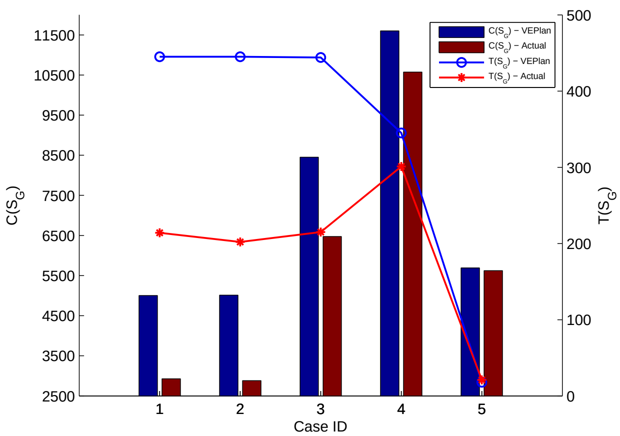


Figure 9: Impact of operator types

6.2.4 Impact of Data Transfer Sizes. We examine two dataflows, 4 and 5, for the impact of data transfer sizes and the results are shown in Figure 10. Dataflow 4 contains 50 nodes and 96 links, while dataflow 5 contains 50 nodes and 500 links. For each link in dataflow, take the original transfer size as 1, we double the transfer size as the test case increases. For Case 1 to 5, the transfer size increase by 1, 2, 4, 8 and 16 times. Although the transfer size increases exponentially, for dataflow with smaller number of links, the cost (bar) and query completion time(plot) only have linear increases; but for dataflow with relatively large number of links, the cost (bar) and query completion time (plot) have polynomial increase.

6.2.5 Impact of Budget Constraint. The impact of budget constraint is studied for dataflow 4. The result is shown in

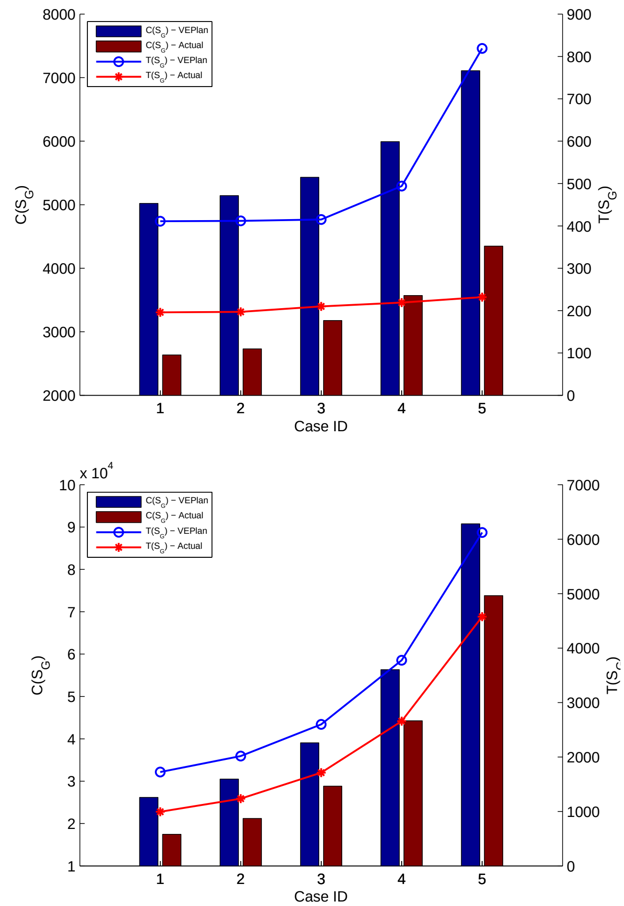


Figure 10: Impact of data transfer sizes (top: Dataflow ID = 4, bottom: Dataflow ID = 5)

Figure 11 and the budgets are set to the cost of Min-VEPlan adding 0%-100% of the difference between the cost of Min-VEPlan and Max-VEPlan).

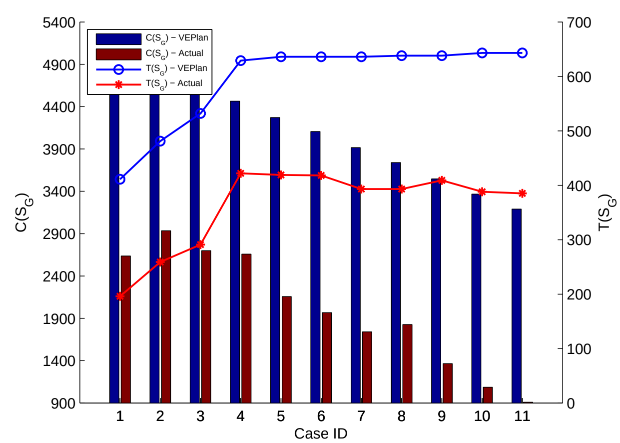


Figure 11: Impact of budget (Dataflow ID = 4)

From Figure 11 we can conclude that the completion time(plot) remains the same after a certain point, while the actual cost still decreases as the budget decreases. From the user's point of view, if the completion time is not a big concern, the budget should be set as close to minimum as possible, otherwise the budget should be set as close to maximum as possible.

6.2.6 Impact of Dataflow Size. To evaluate the impact of dataflow size on query completion time and monetary cost, we experiment with 9 dataflows of sizes from small to large as indicated in Table 4. The fifth dataflow in Table 4 is excluded in this study since its links number is incomparable with others. The actual query completion time (plot) and cost (bar) are given in Figure 12. The budget for each dataflow is set to 80% of its Max-VEPlan. Generally, as the size of dataflow increases linearly, the query completion time and cost increase linearly too.

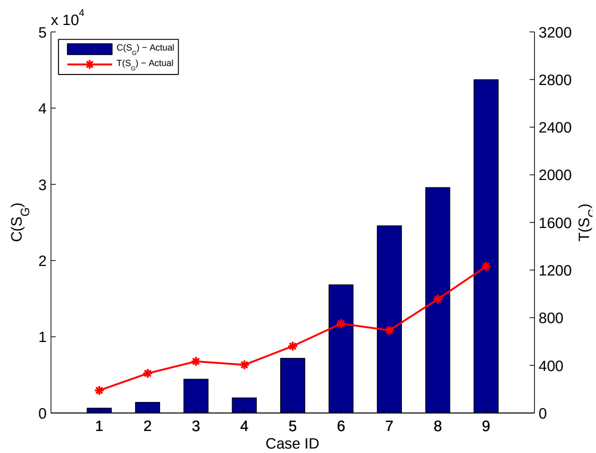


Figure 12: Impact of dataflow size

7 Conclusion

In this paper, we formulate scheduling of dataflows onto Cloud resources under the objective of minimizing the query completion time under certain budget constraints. A heuristic scheduling algorithm, Layer-oriented Resource Allocation within Budget constraint (LRA-$\mathscr{B}$) is proposed and evaluated. LRA-$\mathscr{B}$ first calculates a Min-VEPlan to check if scheduling is available under the given budget constraint, and then calculates a Max-VEPlan and adjust the VEPlan by keeping reassigning the operators on non-critical paths to results in the minimum increase in query completion time for the latest cost savings until the cost is within the budget constraint. Finally the VEPlan is mapped to the Cloud while minimizing the query completion time by adapting a layer-oriented mapping strategy and keeping selecting the minimum partial query completion time for each operator. Experiments are conducted on numerous dataflows and Cloud environment configurations, and the overall results are quite promising and indicate the effectiveness of our algorithm. Our future plan is to run our experiments on a local private Cloud, called Saluki Cloud established and managed by Eucalyptus with a few Beowulf clusters. We also would like to extend our model to consider various factors such as energy consumption and resource utilization of data centers.

References

[1] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. G. Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, G. Weikum, *The Claremont Report on Database Research*, SIGMOD Record, 37(3):9-19, 2008.

[2] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-Hui Su, K. Vahi, *Characterization of Scientific Workflows*, Third Workshop on Workflows in Support of Large-Scale Science, pp. 1-10, 2008.

[3] F. Cao and M. Zhu. A Fault-tolerant Workflow Mapping Algorithm under End-to-end Delay Constraint, *Proceeding of the Symposium on Advances of High Performance Computing and Networking (AHPCN-2011) in conjunction with the 13th IEEE international conference on High Performance Computing and Communications (HPCC-2011)*, Banff, Canada, September 2-4, 2011.

[4] Condor, *http://www.cs.wisc.edu/condor*.

[5] Condor, *http://www.cs.wisc.edu/condor/dagman*.

[6] J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, 6th Symposium on Operating System Design and Implementation, pp. 137-150, 2004.

[7] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, S. Koranda, *GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists*, *Proceedings* of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), pp. 225-234, 2002.

[8] E. Deelman, G. Singh, M. Su, J. Blythe, and Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. Laity, J. Jacob, D. Katz, *Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems*, in Sci Program., 13:219-237, July 2005.

[9] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta,T. H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, L. Zhao, *Managing Large-scale Workflow Execution from Resource Provisioning to Provenance Tracking: The Cybershake Example*, Second IEEE International Conference on e-Science and Grid Computing (e-Science '06), pp. 14, 2006.

[10] M. N. Garofalakis and Y. E. Ioannidis, *Parallel Query Scheduling and Optimization with Time- and Space-Shared*

Resources, VLDB, pp. 296-305, 1997.

[11] L. M. V. Gonzalez, L. R. Merino, J. Caceres, and M. Lindner, *A Break in the Clouds: Towards a Cloud Definition*, Computer Communication Review, 39(1):50-55, 2009.

[12] R. M. Karp, *Reducibility among Combinatorial Problems*, Plenum Press, 1972.

[13] J. Kleinberg and E. Tardos, *Algorithm Design*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[14] H. Kllapi, E. Sitaridi, M. M. Tsangaris and Y. Loannidis, *Schedule Optimization for Data Processing Flows on the Cloud*, SIGMOD'11, Athens, Greece, June 12-16, 2011.

[15] D. Kossmann, *The State of the Art in Distributed Query Processing*, ACM Comput. Surv., 32(4):422-469, 2000.

[16] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, *Profit-Driven Service Request Scheduling in Clouds*, *Proceedings* of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID '10), pp. 15-24, 2010.

[17] M. J. Litzkow, M. Livny, and M. W. Mutka, *Condor - A Hunter of Idle Workstations*, 8th International Conference on Distributed Computing Systems, pp. 104-111, 1972.

[18] D. T. Liu and M. J. Franklin, *The Design of GridDB: A Data-Centric Overlay for the Scientific Grid*, In VLDB, pp. 600-611, 2004.

[19] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, *Pig Latin: A Not-so-foreign Language for Data Processing*, In SIGMOD Conference, pp. 1099-1110, 2008.

[20] S. Pandey, L. Wu, S. M. Guru, and R. Buyya, *A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments*, IEEE AINA, pp. 400-407, 2010.

[21] S. Shankar and D. J. DeWitt, *Data Driven Workflow Planning in Cluster Management Systems*, HPDC, pp. 127-136, 2007.

[22] J. N. Silva, L. Veiga, and P. Ferreira, *Heuristic for Resources Allocation on Utility Computing Infrastructures*, In B. Schulze and G. Fox, editors, MGC, pp. 9. ACM, 2008.

[23] J. Smith and R. Nair, *Virtual Machines: Versatile Platforms for Systems and Processes*, The Morgan Kaufmann, 2003.

[24] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, *Condor - A Distributed Job Scheduler*, Beowulf Cluster Computing with Linux, The MIT Press, MA, USA, 2002.

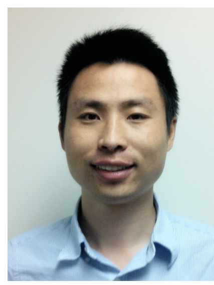
[25] A. Thusoo, IEEE 26th International Conference on Data Engineering (ICDE), pp. 996-1005, 2010.

[26] K. Tsakalozos, M. Roussopoulos, V. Floros, and A. Delis, *Nefeli: Hint-based Wxecution of Workloads in Clouds*, IEEE ICDCS, pp. 74-85, 2010.

[27] M. Tsangaris, G. Kakaletris, H. Kllapi, G. Papanikos, F. Pentaris, P. Polydoras, E. Sitaridi, V. Stoumpos, and Y. Ioannidis, *Dataflow Processing and Optimization on Grid and Cloud Infrastructures*, IEEE Data Eng. Bull., 32(1):67-74, 2009.

[28] A. Weiss, *Computing in the Clouds*. NetWorker, 11(4):16-25, Dec. 2007.

[29] J. Yu and R. Buyya, *A Novel Architecture for Realizing Grid Workflow using Tuple Spaces*, in GRID 04: *Proceedings* of the 5th IEEE/ACM International Workshop on Grid Computing, pp. 119-128, 2004.



Dabin Ding received his B.S. and M.S. degree in Computer Science from Northeastern University, China in 2006 and 2008. He is currently a Ph.D. student in the Department of Computer Science at Southern Illinois University Carbondale. His research interests include XML Query Processing/Optimization, and Cloud Computing.

Fei Cao received her B.S. degree in Software Engineering from Zhejiang University, China in 2007, the M.S. degree in Computer Science from California State University Fullerton in 2009. She is currently a Ph.D. student in the Department of Computer Science at Southern Illinois University Carbondale. Her research interests include Distributed Networking, and High Performance Computing.

Dunren Che is currently a professor in Computer Science at Southern Illinois University Carbondale. He earned his Ph.D. in Computer Science from the Beijing University of Aeronautics and Astronautics, Beijing, China in 1994. His main research interests include Database (especially XML Database and Query Processing/Optimization) and Data Mining, as well as Cloud Computing and Big Data as his most recent emphasis.

Michelle M. Zhu received her Ph.D. degree in Computer Science from Louisiana State University in 2005. She spent two years in the Computer Science and Mathematics Division at Oak Ridge National Laboratory for her PhD dissertation from 2003 to 2005. She is an Associate Professor in the Department of Computer Science at Southern Illinois University Carbondale. Her research interests include distributed and high-performance computing, remote visualization and sensor networks.

Wen-Chi Hou received the M.S. and Ph.D. degrees in Computer Science and Engineering from Case Western Reserve University, Cleveland Ohio, in 1985 and 1989, respectively. He is a professor of Computer Science at Southern Illinois University Carbondale. His interests are in Statistical Databases, Mobile and XML Databases, Data Streams Processing, and Query optimization.

A Cooperative Game Theory-based Approach for Energy-Aware Job Scheduling in Cloud

Mustafa Khaleel*, Saad Alqithami*, Michelle M. Zhu*, Dunren Che* and Wen-chi Hou*
Southern Illinois University, Carbondale, IL, 62901, USA

Abstract

This paper addresses the problem of energy-aware job scheduling for underlying cloud nodes using cooperative game theory. The objectives are on resource utilization maximization and the power consumption minimization without violating the job's latest completion time (Makespan). Cloud computing can deliver platform, software, storage and data services through web browsers as a metered service. Due to the skyrocketed electricity cost and a large number of active users, Cloud service providers are highly motivated to adopt a performance guaranteed and cost-effective job scheduler with low power consumption and high job throughput. Therefore, an energy-aware job scheduling algorithm is proposed for a bag of tasks based on the premise of Nash Bargaining Solution (NBS), which can ensure Pareto-optimality. In such a cooperative theoretical gaming, each job seeks to locate a cloud machine that can both guarantee the low energy under certain makespan constraint. Simulation results show that our approach significantly reduces the power consumption by strategically selecting appropriate mapping nodes for prioritized task modules. Our approach consistently achieves lower energy consumption and higher resource utilization than some comparable methods.

Key Words: Cloud computing; game theory; NBS; power consumption; makespan.

1 Introduction

High Performance Computing (HPC) systems are playing an ever-increasing important role for large-scale scientific applications that are collaborated among a group of distributed scientists [17]. In order to meet the intensive data and computing needs, HPC system together with the managing software needs to be designed as a highly flexible, scalable and cost-effective platform [49]. Cloud infrastructure provides users with on-demand and pay-on-the-go services realized through virtualization technology [9]. There are three basically main types of cloud services, namely *Infrastructure-as-a-Service* (IAAS), *Platform-as-a-Service* (PAAS) and *Software-as-a-Service* (SAAS). SAAS allows users of the cloud to run different software over web browsers remotely. PAAS provides users with an environment to run their applications using specific development environments. Furthermore, IAAS has virtual machines (VMs) that can be setup and configured on physical nodes to execute the assigned job modules. Gartner estimated that he market opportunity for Cloud computing will be worth around $150 billion by 2014 [18]. In recent years, the electricity cost on managing data centers for clouds have skyrocketed [3, 5]. For example, a typical data center with 1,000 racks consumes about 10 Megawatt of power during normal operation [17]. Over the past decade, the cost of servers running and cooling systems has increased by 400 percent [15]. Thus, the design and development of a power efficient cloud infrastructure have become a critical research area in today's HPC system. From the cloud provider's view, high job throughout is desired to satisfy as many user requests as possible with the limited computing and networking resources. The Service level Agreement (SLA) between provider and customers must also be met to provide some guaranteed Quality of Service (QoS). Many researchers have been working on efficient resource management/job scheduling strategies to reduce the energy consumption. Hardware manufactures on the other hand, focus more on the power-efficient chip and technology design (e.g., [12, 29, 44, 52]). Incorporating power management to scheduler design adds complexity due to the difficulty of balancing power optimization with other objectives [20, 49]. Several techniques have been applied to decrease energy consumption [46]. Dynamic Voltage Scaling (DVS) technique has been used as one of the effective techniques that scale the CPU frequencies without compromising the execution end-time [8, 19, 50]. The cloud scheduling tackles the energy issue from a higher software level and such optimized problem has been proven to be NP-complete [51]. Thus, we propose a heuristic energy-aware job scheduling algorithm that takes into account both makespan and energy consumption as well as higher utilization rate. Since the minimized energy consumption and execution time are two contradicting objectives, a tradeoff is sought by using cooperative game theory to find a better payoff for both factors (makespan and power consumption). We formulate our problem as a min-min-max optimization problem. Since min-min-max optimization has a high complexity, we convert the min-min-max problem optimization into the max-max-min problem optimization based on previous work [25]. In addition to the

* Computer Science Department, Email:{mkhaleel, salqithami, mzhu, dche and hou}@cs.siu.edu.

low complexity of a max-max-min problem, Pareto optimality from the Nash Bargaining Solution can be guaranteed [25].

The rest of the paper is organized as follows. A survey of cloud scheduling algorithms is given in Section 2. Mathematical models for cloud meta-modules, cloud environments, and energy consumption are constructed in Section 3. The makespan and power consumption as two conflicting objectives is proved in Section 4, thus optimizing both at the same time is not possible. The problem is then formulated as min-min-max problem optimization. The mathematical model and a cooperative game-based approach with the objective function of reducing both makespan and power consumption of cloud jobs are proposed in Sections 5 and 6. The details of the algorithm are presented in Section 7. Simulation results are given in Section 8. Finally, the conclusion can be found in Section 9.

2 Related Works

Many researchers have studied the problem of scheduling a bag of tasks onto heterogeneous computing nodes with guaranteed completion time and low power consumption. Some of these studies proposed approaches using the DVS model to adjust the frequency of processor while others incorporated methods to optimize the dynamic power management [21]. Energy and time optimization using game theory has been gently investigated in the cloud. Young Choon Lee and Albert Y. Zomaya [30] proposed a scheduler named Energy-Conscious Scheduling (ECS) to minimize the energy consumption for precedence-constrained applications with DVS. Samee Ullah Khan and Ishfaq Ahmad [25] developed a cooperative technique for mutli-constrained multi-objective Generalized Assignment Problem (GAP) with DVS technique in computational grids. Kim et al. [26] proposed an energy-aware scheduling algorithm for bag-of-tasks applications with each subjected to specific deadline constraint. Garg et al. [17] proposed near-optimal energy efficient scheduling polices to determine the scheduling order of data center to minimize some factors such as CO_2, cooling system, and power consumption. Chen et al. [10] proposed three online solutions strategies to control the power consumption for running servers based on steady state querying analysis, feedback control theory, and a hybrid mechanism. Zhu et al. [50] proposed two novel power-aware scheduling algorithms for task sets with and without precedence constraints for multiprocessor systems. Their algorithm was based on the concept of slack sharing among a set of processors. The scheduling techniques reclaimed the time unused by a task to reduce the execution speed. Bradley et al. [7] presented a solution for power consumption problem via workload history. Lawson and Smirni [28] designed an algorithm that can dynamically scale the number of processors in order to decrease the power consumption by turning the nodes into sleep modes. Huang et al. [22] proposed a near optimal solution for heterogeneous processors to minimize the power consumption of the system and complete all tasks by their deadline. Duy et al. [13] proposed a green scheduling algorithm to optimize server power consumption in cloud computing. Their algorithm focused on how to turnoff unused servers and restart them to minimize the number of active servers. Borgetto et al. [6] presented an integrated approach for VM migration, reconfiguration, and Physical Machine (PM) power management. They proposed a method to unify all three above-mentioned methodologies. The goals of the approach were to minimize energy consumption and minimize SLA violations. Pinheiro et al. [37] proposed an idea of categorizing the servers in a cluster system into two groups, namely one group with high capacity executing the applications with intensive data while the other one can be switched-off to save the power. Rountree et al. [38] proposed a technique to bound optimal energy saving using linear programming. Ge et al. [19] proposed a distributed performance-directed DVS scheduling strategy to reduce the power consumption during parallel applications. hey aim to decrease the power when the peak CPU performance is not necessary. In general, power managements were heuristic [23, 33, 43] or stochastic approaches [39, 41]. There were several commonly used job scheduling policies including Greedy (First Fit) and Round Robin algorithms in open-source cloud computing management systems such as Eucalyptus [35]. Queuing system, advanced reservation and preemption scheduling were adopted by Open Nebula [36]. Nimbus uses some customizable tools such as PBS and SGE [34]. The Greedy and Round Robin were heuristic approaches that select adaptive physical resources for the VM to deploy without considering the maximum usage of the physical resource. The queuing system, advanced reservation and preemption scheduling did not consider any balanced overall system utilization either.

To our best knowledge, the work is different from most existing works in these two aspects: (a) time dependency on cloud infrastructure: the underlying Cloud infrastructure/Virtual Machine (VM) resource availability is time-dependent because of the dual operation modes namely on-demand and reservation instances at various Cloud data centers. (b) Game theory in cloud management: using game theory to calculate the Pareto optimality at a point that guarantees the best utilization rate for cloud management. Some previous game theory work only considers grid environment.

3 Analytical Models

We construct the analytical cost models for cloud meta-modules, underlying cloud computer network graph, and energy consumption model to facilitate a mathematical formulation of the performance constrained optimization cloud scheduling problem.

3.1 Cloud Task Model

Cloud users submit their job modules via a job scheduler to be executed by cloud infrastructure. To generalize our model, we consider N concurrent modules represented as

$T = \{u_1, u_2, ..., u_N\}$. Each module characterized by specific deadline d_{ui} has to schedule by a VM on a particular node v_j. Before the mapping process, modules are sorted in decreasing order of their deadline.

3.2 Cloud Network Model

Figure 1 shows the cloud environment which is comprised of a set of M heterogeneous nodes that are fully interconnected. Since each node may support multiple virtual machines which can be reserved, deployed and run, each VM can use DVS to adjust the frequency needed in a certain (i.e., discrete clock frequencies starting from ($f_{\min}$ to $f_{\max}$). Scaling the frequency of processor v_j from up to down and vice versa depends on whether an assigned module is processor bound or not [17]. Overhead of clock frequency transition are not considered in this paper because it takes only (10ms-150ms) [30]. We consider a general cloud environment where VM reservation and on-demand requests are both supported, which means resource allocation status for the cloud network graph is time dependent. It implies that available computing resources on each node and the bandwidth on each vary over time as shown in Figure 2. We model the underlying cloud network as an arbitrary fully directed network graph $G_{cm} = (V_{cm}, E_{cm})$, where V_{cm} consists of a set of computing nodes $V_{cm} = (v_1, v_2, ..., v_M)$ as well as directed edges between each pair. Node v_j is featured by its normalized computing power including CPU and memory as $p_{v_j,t}$. The communication link $L_{i,j}$ between nodes v_i to v_j is featured by bandwidth $b_{v_i,v_j,t}$ and the minimum link delay d_{v_i,v_j}.

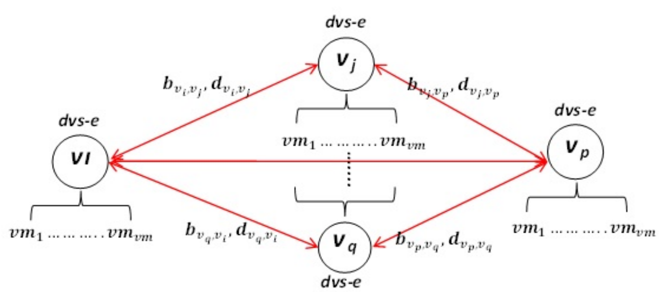


Figure 1: Cloud network graph

Figure 3 shows an example of three reservation requests made on one cloud node during different time slots. Lets assume that 30 percent of the node's general capacity is reserved for request 1 from t_0 to t_2; request 2 reserves 20 percent from t_1 to t_4; request 3 reserves 50 percent from t_3 to t_4. Taking this into consideration, we can get $p_{v_j,t_0,t_4} = \min(50\%, 30\%, 70\%, 50\%)$ where p_{v_j,t_0,t_4} is the maximal available computing power of node v_j from t_0 to t_4. Each node is occupied by one or a set of VMs to execute

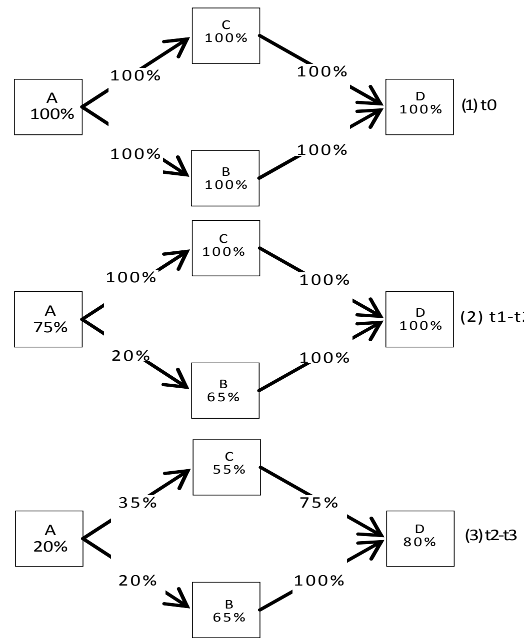


Figure 2: Shows several allocable resource graphs for a cloud network at different time points due to resource allocation

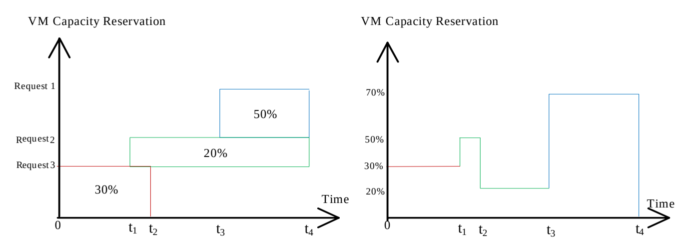


Figure 3: An example of three reservation requests

assigned modules. The largest VM instance that can be allocated on v_j from time t_0 to t_n is computed as the maximum VM instance that can be launched using p_{v_j,t_0,t_n}. The execution time of module u_i on node v_j during time slot t_0 and t_n is then computed as:

$$et_{v_j,t_0,t_n}(u_i) = \frac{C_{u_i}}{P_{vj,t0,tn}} \tag{1}$$

Where C_{u_i} denotes the computational cost of module u_i. Similarly, the maximum link bandwidth along L_{v_i,v_j} during time slot t_m and t_n is $\min(b_{v_i,v_j,t_m,t_n})$.

3.3 Energy Model

The energy model is based on the power consumption in complementary metal-oxide semiconductor (CMOS). Dynamic and static power are two factors that contribute to

COMS circuit power consumption [10, 17, 24, 47]. As reported in [2, 30], dynamic power has the main rule in adjusting power consumption of the system, which can be reduced by lowering the supply voltage using the DVS technique. Dynamic power consumption of a CMOS-based microprocessor is defined to be:

$$P_{vj} = V^2 \times f \times C_{ef} \tag{2}$$

Where V denotes the supply voltage, f is the frequency, and C_{ef} is the effective switched capacitance of circuit. From Equation (2), we can see that power consumption will be reduced by lowering supply voltage which is linearly proportional to CPU frequency [1, 14]. This implies that reducing supply voltage will also reduce the frequency of the processor. From this point, we consider that the frequency of the processor for each computing node in cloud infrastructure can be scaled down from $f_{\max}$ to $f_{\min}$ using DVS model.

3.4 Meta-Module Execution Cost

The cost of running meta-modules over cloud infrastructure is measured by the sum of the total time, during which virtual machines are setup on node v_j multiplied by the power consumption by node v_j to execute parallel modules via deployed VMs. Power of node v_j, p_{v_j,t_0,t_n}, that shares between all virtual machines deployed on node v_j from time slot t_0 to t_n consider as the major contributor to adjust the total running cost for meta-modules which also has a significant effect on the execution cost of cloud systems. The time spent on deploying VMs on node v_j consists of the following components: 1) The startup time for the virtual machines includes selecting a virtual node and transferring a virtual image as well as the boot-up time, and is assumed to be a fixed value of t_{start}. 2) The running time for every assigned module on that VM. Suppose that a set U of modules are assigned to node v_j to be executed on the *Kth*VM, and start to run from time t_s and end at time t_e in a sequential manner. The running time for assigned modules on this VM is computed as:

$$Running\ time = \sum_{u_i \in U, k \in K} \frac{C_{u_i}}{p_{v_j,k,ts,te}} \tag{3}$$

3) When two modules run on the same VM, there could be some idle time after one module is completed and before the next module starts. The total idle time for $VM_{v_j,k}$ can be calculated as:

$$Idle\,(VM_{v_j,k}) = \sum_{u_i \in U_{j,k}} (St_i - et_i - 1) \tag{4}$$

4) The time to shut down that virtual machine is assumed to a constant of t_{shut}. Consequently, we can define the Total Energy Cost as the summation of cloud modules computation cost T_c and cloud underlying network cost E_c. Mathematically, we can formulate:

$$T_c = \sum_{i=1}^{N} C_{u_i} \tag{5}$$

$$E_c = \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{Sj} C_{p_{vm_{v_j,k}}}(u_i) \times \left(t_{start} + Idle\left(VM_{v_j,k}\right) + t_{shut}\right) \tag{6}$$

$$TEC = T_c + E_c \tag{7}$$

Where N is the total number of modules from a particular job, M represents the total number of nodes that have been allocated in the system, and $vm_{v_j,k}$, where k ranges from 1 to Sj, which denotes the total number of VMs that have been set up on a computing node vj. The Utilization Rate for one job with single module is defined as (8-1) or for multiple modules as (8-2):

$$UR = \frac{C_{u_i}}{TEC} \tag{8-1}$$

$$UR = \frac{\sum_{i=1}^{N} C_{u_i}}{TEC} \tag{8-2}$$

It is understood from equation (7) that the cloud networkcost is linearly proportional to *TEC* of the system. This implies that by lowering E_c, *TEC* will be reduced and this results in maximizing *UR* of the cloud provider due to the goal that we achieved from equation (8), which states that *UR* is inversely proportional to *TEC*. From this point, we can state that total energy cost considers the dominating factor in equation (8-1) or (8-2) that leads us to get a higher throughput. For convenience, we provide a summary of the notations used in the cost models in Table 1.

4 Problem Formulations

We first consider a bi-objective scheduling problem to minimize the total energy that is required by the computational nodes to setup VMs and execute the parallel assigned modules over cloud infrastructure and also minimize the makespan (i.e., completion time) of cloud modules at the same time. However, these are two conflicting objectives and cannot be achieved at the same time, as stated in Theorem 1, and then we propose a novel solution to find a Pareto-optimality point from NBS that balances these two conflicted objectives at point that guarantees the minimum power consumption with the minimum acceptable makespan of assigned modules.

Table 1: Notations used in the analytical models

Parameters	Definition
N	The number of modules
u_i	The $i-th$ computing module
C_{u_i}	The computational cost of module u_i
st_i	The start time of module u_i
et_i	The end time of module u_i
$G_{cn} = (V_{cn}, E_{cn})$	The cloud network
M	The total number of nodes in the cloud
v_j	The $j-th$ computer node
v_s	The source node
v_d	The destination node
p_{v_j}	The total computing power of node v_j
p_{v_j,t_1,t_n}	The maximal percentage of computing power of VM on node v_j from t_1 to t_n
$L_{i,j}$	The network link between nodes v_i and v_j
b_{v_i,v_j,t_1,t_n}	The bandwidth of link $L_{i,j}$ from t_1 to t_n
d_{v_i,v_j}	The minimum link delay of link $L_{i,j}$
t_{start}	The time spent on setting up a virtual machine
t_{shut}	The time spent on shutting down a virtual machine
$et_{v_j}(u_i)$	The execution time of module u_i running on node v_j
$VM_{v_j,k}$	The $k-th$ VM on the $j-th$ node
vm_{v_j}	The total number of VMs on the $j-th$ node
$PVM_{v_j,k}$	The computing power of $VM_{v_j,k}$
$U_{j,k}$	set of modules scheduled on node *vj*'s *k* th VM

Theorem 1: The bi-objective problem of minimizing the makespan and minimizing the power consumption is non-approximable within a constant factor.

Proof: Assume: (1) there are two different scheduling strategies each with a different objective function. A schedule S has the objective of minimizing the power consumption while schedule Q has the objective of minimizing the completion time (2) arrange of frequencies ($f_{\min} - f_{\max}$) that operates the processor of computing node v_1 to execute the assigned module u_1 via virtual machine vm_1. Thus, two cases exist.

Case 1: (Schedule S with the objective of minimizing power). Schedule S starts with executing module u_1 over node v_1 using ($f_{\min}$) to satisfy the objective of minimizing power. According to [25], the frequency of CPU is proportional to the energy consumption per operation in the system which means that operating the processor of computing node v_1 at a lower frequency ($f_{\min}$) to execute the assigned modules u_1 over virtual machine vm_1 results in decreasing the system's energy due to the fact that ($E \propto f^2$). On the other hand, the time required to finish the execution process for a particular module u_1 is inversely proportional to the frequency of CPU [49]. This means that running CPU at lower frequencies will incur more time to complete an operation.

Case 2: (Schedule Q with the objective of minimizing makespan). In the second case, Schedule Q operates the processor of computing node v_1 at the maximum level of frequencies and because the frequency of CPU for node v_1 is inversely proportional to the time required to execute and finish module u_1 due to ($f \propto t^{-1}$) [25], the makespan of assigned module u_1 will decrease satisfying the deadline constraint. But due to the fact that frequency is cubic proportional to power consumption ($P \propto f^3$) [40], this will result in increasing/maximizing power consumption.

Case 1 has the minimum power consumption with the maximum makespan while case 2 results in the minimum makespan and maximum power consumption. None of the above cases is considered in our simulation because both cases contradict our assumption, which focuses on finding a tradeoff between the power consumption and the makespan for improved utilization rate. Our algorithm tries to balance these two conflicting objectives using the Nash Bargaining Solution (NBS) from a theoretical cooperative game, which guarantees a bargaining point that results in high utilization throughput. In other words, the point that gives a minimum acceptable makespan operates at the minimum CPU frequency with the constraint of module's deadline.

Definition (1): *Given:*

1- *A module's computation cost:*

$$T_c = \sum_{i=1}^{N} C_{u_i} \tag{9}$$

2- *A module's execution time:*

$$et_{v_j,vm_k(u_i)} = \frac{\sum_{u_i \in U} C_{u_i}}{p_{VM_{v_j,k}}} \tag{10}$$

3- *An arbitrary computer network in a cloud environment* $G_{cn} = (V_{cn}, E_{cn})$:

$$E_c = \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{Sj} C_{p_{vm_{v_j,k}}}(u_i) \times (t_{start} + Idle(VM_{v_j,k}) + t_{shut}) \tag{11}$$

4- *Total Energy Cost TEC:*

$$TEC = T_c + E_c \tag{12}$$

5- *Cloud Utilization Rate for N jobs:*

$$UR = \frac{\sum_{i=1}^{N} C_{u_i}}{TEC} \tag{13}$$

With time-dependent link bandwidth and node computing power, we formulate the energy-aware job scheduling algorithm as:

$$\text{Minimize } E_c = \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{Sj}\Big(C_{p_{vm_{v_j,k}}}(u_i)\times(t_{start} + Idle(VM_{v_j,k}) + t_{shut})\Big)x_{i,j,k} \tag{14}$$

$$\text{Minimize } et_{v_j,vm_k}(u_i) = \left(\frac{\sum_{u_i\in U} C_{u_i}}{p_{VM_{v_j,k}}}\right)x_{i,j,k} \tag{15}$$

$$\text{Maximize}\, UR = \frac{\sum_{i=1}^{N} C_{u_i}}{TEC} \tag{16}$$

Subject to:

(1) $[(et_{v_j,k}(u_i))] < (d_{u_i})$

(2) $f_{v_j}^{\min} < f_{i,v_j,k} < f_{v_j}^{\max}$

(3) $x_{i,j,k}\begin{cases}1 & \text{if module } u_i \text{ schedules to node } v_j \text{ over } VMvm_k\\ 0 & \text{otherwise}\end{cases}$

Constraint (1) is a deadline of each module and execution time of module u_i should be less than its deadline. Constraint (2) is near-optimal frequency between $(f^{\min})$ and $(f^{\max})$ such that the utilization rate is maximized and the minimum makespan of the system is guaranteed.

5 Cooperative Game Theory for Scheduling Cloud Modules

In ordinary game, a finite number of players perform different strategies based on their payoffs' matrices [11]. Their set of strategies can be a compact or a convex subset of a finite dimension of Euclidean space [16, 32]. A game, in general, can be either cooperative or non-cooperative as proposed by John Nash in [16]. Cooperative game has several more primitive advantages while non-cooperative game has a generalization of *min-max theorem* aimed at zero-sum games [16]. Cooperative games [25], a) do not require specific details of the players' movement, b) are more powerful since their convergence to solution is stable and will not drift away from the equilibrium. Nevertheless, non-cooperative games are highly susceptible for any changes in the strategy, which may lead to different results, and c) achieve a better performance of each player than in anon-cooperative game at the Nash equilibrium stage. In light of this, a cooperative game has been used in our model since the focus is on minimizing the total energy required by the computational nodes to setup VMs and execute the assigned modules and maximizing the resource utilization over cloud infrastructure. In particular, higher efficiency of the collective benefits can be reached through the NBS. The players usually interact through bargaining of a partial desire of some payoffs in NBS, and they will keep interacting unless they reach their goal. NBS ensures the Pareto optimality. Thus, NBS provides a sufficient outcome to the proposed problem as in the cloud system, the cloud provider's objective is to cooperatively minimize module's completion time and power consumption, and the preference is finding the Pareto optimality. A Cooperative game includes a set of M players who competes to achieve better performance. Each player, j, $(j \in \{1,...,M\})$ has an objective function and desired initial performance v_j^0 defined as the minimum performance required to be achieved by each player without any cooperation [48]. Player's objective function is on a subset of $\mathbf{R}^w$ describing P where P is nonempty, closed and convex set. As one of the objective's goals for each player is to achieve the minimum performance v^0 to be able to enter the game [48], our scheme considers that there is at least a vector $(f = \{f_1, f_2, ..., f_M\})$ performance for all players each component should be equal or superior than to v^0. This implies that there is a set of achievable performance, L, in the system, and if we assume that v^0 is part of $\mathbf{R}^w$ in case $V_0 = \{(v \in L), v \geq v^0\}$ [48], we can define v^0 as the initial agreement point in the game where each player should have by the system to be able to execute the assigned job modules. Let $Q = \{(L, v^0), L \subset \mathbf{R}^w\}$, we define the idea of Pareto optimality in the cooperative game as based on some previous work [25, 48]:

Definition 2: v is Pareto optimal if for each $z \in L, z \geq v$, then $z = v$. In large scale cloud systems with a set of data centers and computing machines, a set of Pareto optimal points exist with a set of infinite number of points [48]. It is our goal to find the point from those infinite points to operate the scheduler that guarantees the system's utilization rate.

To find the desired point: - first, we define fairness axioms because it is considered as the satisfactory method in game theory [48], then we introduce the concept of NBS which can satisfy the above requirement. Thus, the concept of NBS is defined according to the definition proposed by [25, 48]: A *mapping $S: \mathbb{Q} \rightarrow \mathbf{R}^w$ is said to be a NBS* under two conditions: a) $S(L, v^0) \in V_0$ and b) $S(L, v^0)$ is Pareto

Optimal and it should satisfy axiom (1), (2), and (3). The details of each axiom can be found in references [25, 48].

Definition 3: when v^* is given by $S(L, v^0)$, we can say that:

1) v^* represents the Nash Bargaining Point.
2) $f^{-1}(v^*)$ represents the set of Nash Bargaining Solutions.

After defining NBS, we need to define the bargaining point as explained in references [25, 48].

Theorem 2: according to [25, 48], if f_j is injective on X_0 where $j \in J$, and based on theorem 1 in [25], there are two problems that can be considered (p_{v_j}) and (p'_{v_j}) :-

$$(p_{v_j}) \qquad Max \prod_{j \in J} \left(f_j(x) - v_j^0 \right) \qquad x \in X_0 \qquad (17)$$

$$(p'_{v_j}) \qquad Max \sum_{j \in J} \ln \left(f_j(x) - v_j^0 \right) \qquad x \in X_0 \qquad (18)$$

Depending on the above considerations, we achieve:

a) (p_{v_j}) has a unique solution; the Nash Bargaining Solution set will be considered as a single point.
b) (p'_{v_j}) is a convex and has a unique solution.
c) It is understood that (p_{v_j}) and (p'_{v_j}) is equivalent with each other which makes the unique solution of (p'_{v_j}) as NBS.

There are two reasons behind the objective of (p'_{v_j}):

1) The low complexity of (p'_{v_j}).
2) (p'_{v_j}) always can guarantee the NBS

From this point, we need to optimize the cloud scheduling problem as (p'_{v_j}).

6 Optimality and Fairness Scheduling Scheme for Cloud Meta Modules

A few grid-scheduling schemes have been proposed based on the usage of the game theory. Some of them simulated the algorithm based on the idea of Nash equilibrium point [27, 42] while other applies the concept of the Pareto-optimal points [25]. In [25], the Nash bargaining point was proposed as a suitable solution for scheduling a set of tasks each with deadline constraint onto heterogeneous computational grids. Our mathematical models are different from [25] in two aspects: (1) The underlying Cloud infrastructure/Virtual Machine (VM) resource availability is time-dependent because of the dual operation modes namely on-demand and advance instances reservation supported by various cloud data centers. (2) Using game theory in cloud management to calculate the Pareto optimality at a point that guarantees the best utilization rate for cloud management. Similar to [25] we consider the Nash bargaining point as the desired point for the cloud scheduler to schedule the cloud meta-modules onto cloud infrastructure and execute over deployed VMs due to the Pareto optimality and fairness property related to NBS [48]. Achieving Nash bargaining point depends on the initial performance (v^0) required for each machine by the system. Machines with the least minimum performance can compete for assigned job modules in the system. To generalize our model: - first, we assume that there are *N* job modules, *i= {1, …, N}*, each with deadline constraint and *M* cloud computing nodes, *j= {1, …, M}*, each with vm_k^j virtual machines, *k= {1, …, M}*. Each node v_j aims to increase its performance better than its initial performance for assigned modules. All nodes in the cloud infrastructure have the same goal. In this case, the cloud scheduler has to schedule the cloud modules such that the scheduling should be fair for all machines. To address such an issue, we need to find the NBS. Because cloud architecture needs to meet the requirements of both the cloud users and cloud provider, NBS can be defined as solving the energy optimization problem for provider and also satisfying the deadline for each assigned module for users. Assuming that there are *M* nodes each with *VM* virtual machines compete for *N* job modules. Each computing node is characterized by: a) The Minimum Performance Rate (MPR) b) Peak Power Rate (PPR) c) Achieving performance higher than MPR with power consumption less than or equal to PPR d) the capacity for assigned job module u_i should be less than or equal to capacity of deployed VM. Based on this assumption and according to theorem 2, NBS can be the solution of the following optimization problem as stated in [48]:

$$Y = \begin{cases} \max\limits_{f_j} \prod\limits_{j=1}^{M} (f_j - MPR_j) & \\ f_j \geq MPR_j & j \in \{1 \dots M\} \\ f_j \leq PPR_j & j \in \{1 \dots M\} \\ C(u_i) \leq C(vm_k) & i \in \{1 \dots N\}, k \in \{1 \dots vm\} \end{cases} \qquad (19)$$

To construct our optimization problem defined by equation (19) and search the NBS for our cloud infrastructure, we need to firstly transform our problem into a cooperative game theory problem which considers each computing node as a player with the objective function of: a) achieving at least the minimum performance to be able to enter the game and compete for assigned job modules b) executing assigned modules with the minimum completion time (under deadline constraint) and consuming the minimum power as much as possible. Similar to what is described in [25], the cooperative game theory in the context of cloud computing scheduling system is defined by the following:

(1) $$\mathbf{min} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{Sj} \left(C_{p_{vm_{v_{j,k}}}}(u_i) \times C_{t_{vm_{v_{j,k}}}} \right) x_{i,j,k}$$

where $\left(C_{p_{vm_{v_{j,k}}}}(u_i) \right)$ is a power to execute module u_i on computing node v_j over virtual machine vm_k.

(2) $$\mathbf{min} \max_{1 \le j \le (m,k)} \sum_{i=1}^{N} et_{v_{j,k}}(u_i)\, x_{i,j,k}$$

Subject to:

(1) $[(et_{v_{j,k}}(u_i))] < (d_{u_i})$

(2) $f_{v_j}^{\min} < f_{i,v_{j,k}} < f_{v_j}^{\max}$

(3) $p_{vm_k} \ge 0$

(4) $\sum_{k=1}^{vm} p_{vm_k} \le PPR$

(5) $x_{i,j,k} \begin{cases} 1 & \textit{if module } u_i \textit{ schedules to node } v_i \textit{ over } vm_k \\ 0 & \textit{otherwise} \end{cases}$

We add constraint (3) because the power of each VM deployed on v_j should be a positive number to satisfy the cooperative game theory which states that the objective function, $f_j(x)$, for each player is closed, nonempty, and convex set [25] that makes the dimension of the set a positive number. Constraint (4) indicates that the total power consumption by VMs should be less than the total power of v_j. According to [25], to reduce the complexity of our problem and guarantee the bargaining point, we convert the min-min-max problem into the max-max-min problem which is equivalent to the above definition:

(1) $$\mathbf{max} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{Sj} -\left(C_{p_{vm_{v_{j,k}}}}(u_i) \times C_{t_{vm_{v_{j,k}}}} \right) x_{i,j,k}$$

(2) $$\mathbf{max} \min_{1 \le j \le (m,k)} \sum_{i=1}^{N} -et_{v_{j,k}}(u_i)\, x_{i,j,k}$$

Subject to: $\mathbf{(1'), (2'), (3'), (4'), and\ (5')}$.

Cloud providers are always interested in decreasing the power and increasing the resource utilization of cloud infrastructure which is also the same objective as players in the second definition. Based on equation (19), the NBS can be achieved by solving the optimization problem of total cost of cloud infrastructure as:

$$\mathbf{max} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{Sj} v_j^0 - \left(C_{p_{vm_{v_{j.k}}}}(u_i) \times C_{t_{vm_{v_{j,k}}}} \right) x_{i,j,k}$$

and $$\mathbf{max} \min_{1 \le j \le (m,k)} \sum_{i=1}^{N} -et_{v_{j.k}}(u_i)\, x_{i,j,k}$$

Subject to: $\mathbf{(1'), (2'), (3'), (4'), and\ (5')}$.

6.1 The Strategy for Each Player/Machine

The game starts with the condition that each player has to have initial performance v_j^0 to be able to execute the assigned modules. Players satisfying this condition can enter the game and each one has an objective of optimizing both the energy and makespan. Because the objective is to optimize cumulative performances, players collectively cooperate to find a decision that is both energy and makespan efficient. When the scheduler receives a new task, the players interact with each other and use their best strategies to determine some factors such as how long the execution time takes and how much power is needed to execute the task in a way to reduce the makespan while keeping the power consumption low. In our cloud system, each computer node/machine has a different capacity during a different time slot. Machines collectively search and find the best capacity from various nodes that guarantee both energy and makespan requirements. This cooperative action continues until overall system performance improves.

Theorem 3: The total cost for cloud infrastructure depends on two factors while executing the assigned cloud modules:

(a) Power consumption: Inspired by previous work [25], the power that consumed by VMs for executing job modules during different time slots:

$$p_c = \left(\frac{\sum_{i=1}^{N} \sum_{\forall s \in M, s \ne j} \sum_{k=1}^{Sj} C_{p_{vm_{v_{j,k}}}}(u_i) - PPR}{vm} \right)_{t_0, t_n} \tag{20}$$

To prove that for each machine v_k there is a unique solution f_j, we apply Lagrange method [45] for our optimization problem, which is defined as:

$$\ell(f,\alpha,\delta)=\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{S\,j}\mathbf{ln}\left(v_k(\alpha)-P_c+\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{S\,j}v_k-PPR\right)$$
$$+\sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{S\,j}\delta_j(v_k-P_c) \qquad (21)$$

Where $\alpha \le 0, \delta_k \le 0; i=1,2,\ldots\ldots vm$ denotes the Lagrange multipliers. It is observed that constraints are linear in v_k, and $f_{(x)}$ of each machine is to reduce $C_{p_{vm_{v_{j,k}}}}$ which implies that the first-order Kuhn–Tucker conditions are necessary and sufficient for optimality [48]. The proof can be found in [25].

(b) VM overhead: Each computer node has an overhead caused by deployed VMs to execute job modules defined as:

$$O_{v_j}=p(v_j,t_1,t_n)\times(t_{start}+Idle(VM_{v_{j,k}})+t_{shut}) \qquad (22)$$

Based on equation (20) and (22) we can define the NBS for total energy cost as:

$$E_c=\left(\left(\frac{\sum_{i=1}^{N}\sum_{\forall s\in M,s\neq j}\sum_{k=1}^{S\,j}C_{p_{vm_{v_{j,k}}}}(u_i)-PPR}{vm}\right)_{t_0,t_n}\times(t_{start}+Idle(VM_{v_{j,k}})+t_{shut})\right) \qquad (23)$$

7 Algorithm Design

The power cost in cloud consists of two parts, namely useful power for VMs to execute assigned modules and the overhead to setup and tear down VMs as well as idle VM time. By incorporating the equation (20) to mathematical model (12), we calculate the total energy and propose a heuristic Job scheduling approach referred to as Energy-Aware job Scheduling Algorithm (ESAD) within Deadline constraint for each assigned module. Our algorithm aims to maximize the Utilization Rate (UR) in equation (25) by balancing the following two factors: a) reducing the power consumption, b) reducing the makespan or execution time of assigned modules in meta-task structure under certain deadline constraints. The proposed algorithm starts with sorting the entire task modules in decreasing order of their deadlines and scheduling each module with a different deadline value. When ESAD starts to schedule the job modules onto cloud infrastructure, it takes into account two levels of optimization: a) Minimizing the overhead incurred by deploying and shutting down VMs including the VM idle time. Existing VMs are considered as candidate to be reused for new module execution. Reducing VM's overhead improves the resource utilization rate as fewer resources are wasted. b) Selecting appropriate nodes by cloud scheduler in cloud infrastructure to execute assigned module that satisfy the module's deadline. Controlling the frequencies that operate the processors of cloud infrastructure is done by DVS model as stated in section (4).

$$TEC=\sum_{i=1}^{N}C_{u_i}\left(\left(\frac{\sum_{i=1}^{N}\sum_{\forall s\in M,s\neq j}\sum_{k=1}^{vm}C_{p_{vm_{v_{j,k}}}}(u_i)-PPR}{vm}\right)_{t_0,t_n}\times(t_{start}+Idle(VM_{v_{j,k}})+t_{shut})\right) \qquad (24)$$

$$UR=\frac{\sum_{i=1}^{N}C_{u_i}}{TEC} \qquad (25)$$

Satisfying phases 1 and 2 can achieve the objective of maximizing the utilization rate of cloud system. The pseudo code of ESAD is presented in Algorithm 1.

Algorithm 1: Energy-aware job scheduling algorithm (ESAD) within Deadline constraint

Input: Meta-modules and set of DVS-enabled processors
Output: A task scheduling scheme with the minimum power consumption and minimum makespan

Sorted-Array1: Sort modules in decreasing order of their deadlines
for all $u_i \in$ Sorted-Array1 **do**
  compute power consumption for each node v_j where $j \in \{M\}$
  $p_c=\left(\frac{C_{p_{vm_{v_{j,k}}}}(u_i)}{vm}\right)_{t_0,t_n}$
  Sorted-Array2: sort computing nodes in decreasing order of their power consumption
  for all $v_j \in$ Sorted-Array2 **do**
    if the node v_j can satisfy the module u_i's deadline **then**
      if v_j has allocated VMs **then**
        if ($p_{vm} \le p_{v_j}$) and $(C(u_i) \le C(VM))$ **then**
          call *ReuseVM()* to see the chance of reusing a *VM* on v_j
          break
        end if
      end if
      call *AllocateNewVM()* to allocate a new *VM* on v_j
    end if
  end for

Update p_{v_j}

end for

We provide below a brief description of the functions and methods that applied in Algorithm 1. We categorize the functionality of the methods into two phases:

Phase 1) Sorting job modules and cloud nodes: Sorting both job modules and cloud nodes based on their deadlines and power consumption respectively. Modules with critical deadlines are mapped onto computing nodes that result in reduced acceptable makespan with power consumption as much as possible. Figure 4 illustrates an example of a mapping process for modules each with different deadline restriction onto cloud nodes.

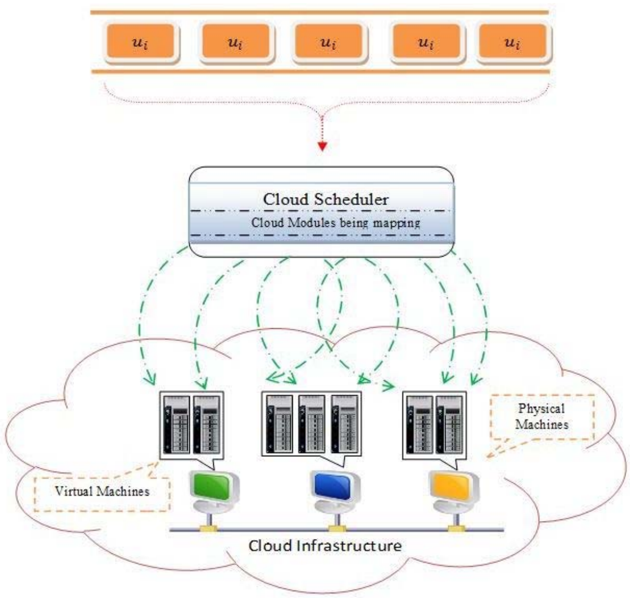


Figure 4: Cloud meta-modules mapped onto cloud infrastructure

Phase2) Mapping Process: To schedule module u_i onto computing node v_j, three considerations have to be taken into account: a) How long it takes by node v_j to deploy VMs and how much power consumed to execute assigned module u_i b) Whether or not deployed VMs on node v_j has enough capacity to handle the computation cost of module u_i c) The processing cost of module u_i on node v_j over virtual machine vm_k should be less than the power cost of node v_j due to the peak power provided. To address these considerations, first we compute the power consumption by each node in cloud infrastructure and then we sort all nodes in decreasing order of their power consumption. Because each node is equipped with DVS model, the frequencies that operate the processors for the system have been controlled by the proposed algorithm. ESAD always seeks for a frequency that operates the node v_j's processor to execute module u_i with the minimum acceptable makespan and the minimum power consumption as much as possible. Cloud nodes with the minimum power consumption execute the assigned job modules under two conditions: a) if and only if it guarantees the module's deadline and b) matching between modules' required capacity and VM's capacity should be satisfied. Two functions are called in this process:

1) **ReuseVM()**: ESAD calls this method when the computing node v_j has allocated VMs. ESAD starts checking whether or not we can reuse one of these VMs on node v_j. Two conditions must be satisfied if we reuse a particular VM: a) The available VM resource should be sufficient to run the module u_i. b) Any possible idle time between two assigned modules should be less than the time to shut down a VM and start up a new one.
2) **AllocateNewVM ()**: If the computing node v_j has no VMs or those VMs cannot be reused, ESAD calls AllocateNewVM () to allocate a new VM for module u_i. The AllocateNewVM () includes creating a new VM with the maximum allocable resource. With Figure 5 as an example, we can calculate the end time of the module u_i as ET_i. We have three different strategies to deploy a VM as *VM1, VM2 or VM3*. Let ve_x be the VM's end time and vs_x be its start time. We calculate the running time for module u_i to be mapped on each VM as: $\left(\dfrac{C_{u_i}}{p_{VM_{v_j,k}}}\right)$, and allocable resource cost on a VM is $p_{vm_{v_j,ve_x,vs_x}}(ve_x - vs_x)$

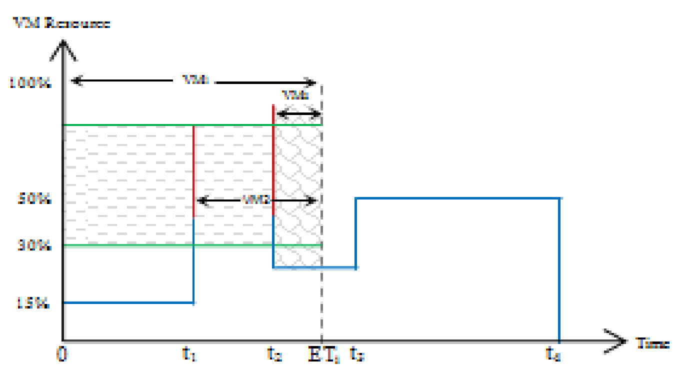


Figure 5: Three different VMs to execute module u_i with end running time of ET_i

The complexity of our heuristic, ESAD algorithm, is in 0(nm log(**m**)), where **n** represents the number of modules and

m denotes the number of computing nodes in the cloud system. Although finding NBS considers as NP-Hard problem [25], the heuristic polynomial time complexity is quite efficient due to convex objective functions in the game.

8 Results and Discussion

We implement the proposed ESAD in Visual C++ on windows 8 desktop PC equipped with Intel Centrino2 CPU of 2.27 GHz and 4.0 GB memory. In the experiments, we compared the system utilization rate, job makespan, and power consumption with that from the Greedy (FirstFit) and Rank Match algorithms in [31]. For the Rank algorithm, we used the cost of each possible scheduling result as the rank value. In the Greedy algorithm, the computing nodes were selected for VMs to be deployed without considering the maximum usage of the nodes. We ran four tests on a set of random modules and network each with different number of edge as illustrated in Table 2. The job scheduling results in term of utilization rate and makespan are presented in Table 3 and 4. Also the charts of utilization rate, makespan, and power consumption are explained in Figures 6, 7, and 8, respectively.

Table 2: Test cases used in the analytical models

Test Case	Number of Modules	Number of Nodes/Edge
1	5	6 / 29
2	10	6 / 29
3	10	10 / 66
4	15	10 / 70

Table 3: Mapping experimental in (%) for utilization rate

Algorithms	Test Case 1	Test Case 2	Test Case 3	Test Case 4
Greedy	0.37	0.41	0.41	0.53
Rank	0.48	0.51	0.52	0.67
ESAD	0.61	0.63	0.64	0.71

Table 4: Mapping experimental in (sec) for makespan

Algorithms	Test Case 1	Test Case 2	Test Case 3	Test Case 4
Greedy	21.84	61.6	58.47	59.51
Rank	19.06	50.43	45.71	41.1
ESAD	18.14	26.91	24.25	40.1

The results demonstrate that our algorithm achieves better mapping performance compared in terms of utilization rate, makespan, and power consumption. In each of the first two cases, we map cloud meta-modules in cloud infrastructure with six computing nodes. Because we define the rank algorithm based on the cost, the rank always achieved a better utilization rate compared with the greedy algorithm. Since neither of these two algorithms considers the module's makespan, this considerably increases the execution time for modules as the utilization rate increases. It implies that that there is no balance between these two performances. However, since our algorithm is based on a trade-off between power and execution

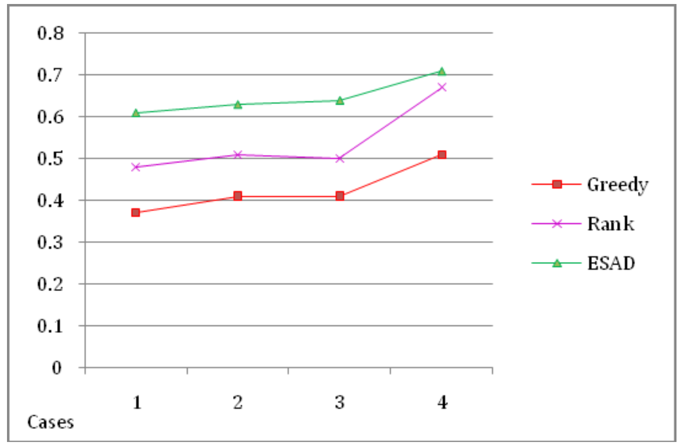


Figure 6: Comparison of UR among different algorithms

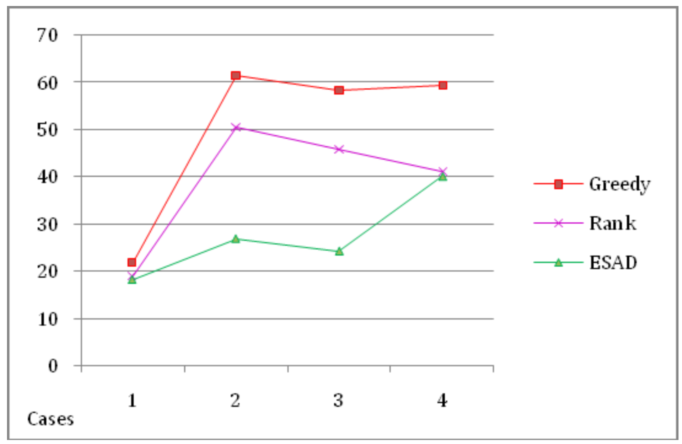


Figure 7: Comparison of makespan among different algorithms

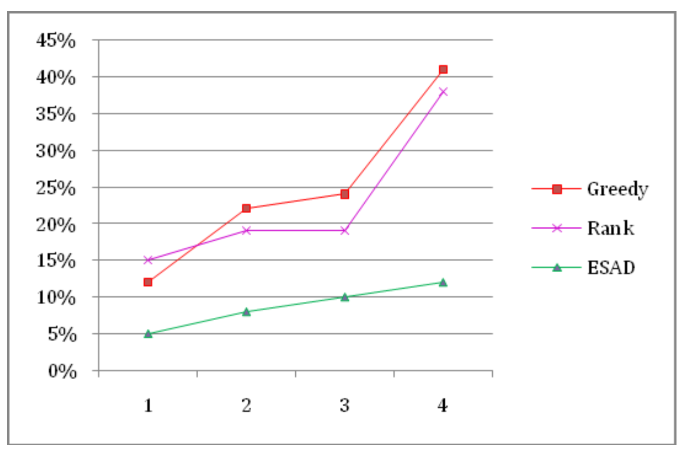


Figure 8: Comparison of power consumption among different algorithms

time under module's deadline constraint using NBS, it can produce high utilization rate. In each of the last two cases, we map job modules to a cloud infrastructure with 10 computing nodes. The rank algorithm achieves better results than the greedy algorithm in terms of utilization rate.

Because the matching between module's requirements and VM's capacity needs to be met at each level of mapping process, the common available cloud computing nodes may be different due to the deadline constraint. Figure 9 illustrates an example of 5 test jobs each with different deadline constraint mapped onto cloud infrastructure with a different number of computing nodes. Axis (x) represents the nodes that have the capacity to handle the module's requirements while axis (Y) represents the execution time in (sec) that each node needs to execute the assigned modules. For each job, the number of computing nodes is different due to: (1) resource capacity that each computing node has (b) matching between the module's and node's requirements. Because our time-dependent algorithm uses a cooperative game theory to seek and find Pareto-optimality at point that guarantees both the execution time and the power consumption without violate the deadline constraint, the results in Figure 9 show that our algorithm achieves smaller execution time than that of greedy and Rank due to the efficiency of mapping results. For instance in Mode l#1 for all computing nodes, the execution times for our algorithm are smaller than that of the greedy and Rank.

9 Conclusions

In this paper, we presented a cooperative game theory based approach for job scheduling in a cloud environment under some constraints. Apparently, it is of the cloud service provider's interest to improve the system throughout in order to satisfy more user requests with the limited hardware resources. The resource utilization rate is a very important performance metric. Furthermore, minimizing the job's execution time and power consumption are also very important. Our approach aims to achieve multiple goals, namely minimizing the energy consumption given certain maximum makespan bounds. Such trade-off between these two objectives is realized by using Nash Bargaining from cooperative game theory, which guarantees the Pareto optimality from bargaining points.

Our simulation experiment results have demonstrated that our algorithm significantly improved the utilization rate compared with two other scheduling algorithms of greedy and rank matching. It is of our future interest to incorporate the task consolidation and VM migration technique into our algorithm for better system performance.

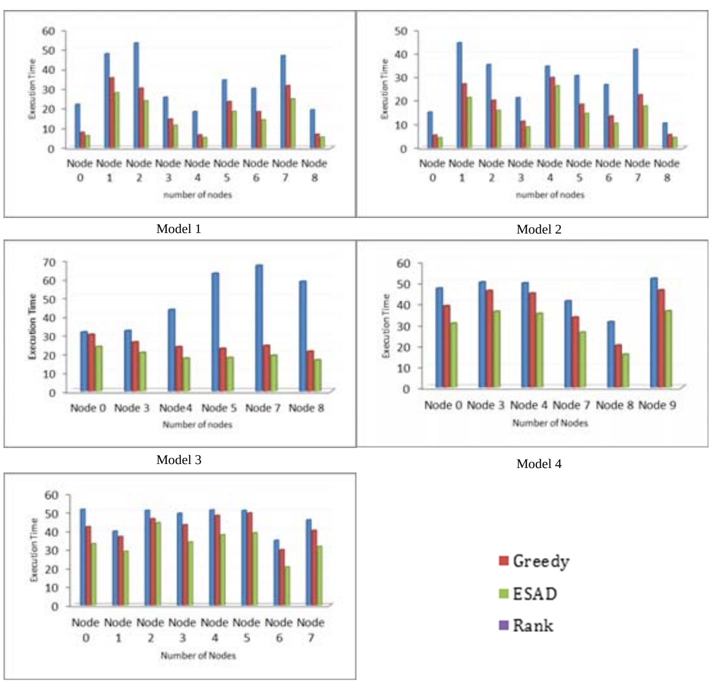


Figure 9: Execution time of scheduling cloud modules onto different cloud nodes using various algorithms

References

[1] H. Aydin, R. Melhem, D. Mosse, and P. Mejya-Alvarez, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," *Proc. of the 22nd IEEE on Real-Time Systems Symp.(RTSS '01)*, pp. 95-105, Dec 3-6, 2001.

[2] H. Aydin, R. Melhem, D. Mosse, and P. Mejya-Alvarez, "Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics," *Proc. Of the 13th Euromicro Conference on Real-Time Systems*, Delft, HollandX, pp. 225-232, 2003.

[3] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing," *Future Generation Computer Systems*, 28:755-768, 2012.

[4] P. Bertsekas, "Nonlinear Programming (Second ed.)," Cambridge, MA.,1999.

[5] R. Bianchini and R. Rajamony, "Power and Energy Management for Server Systems," *Computer*, 37(11):68-76, Nov. 2004.

[6] D. Borgetto, M. Maurer, Georges Da-Costa, Jean-Marc Pierson, and Ivona Brandic, "Energy-Efficient and SLA-Aware Management of IaaS Clouds," *Proc. of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy)*, Madrid, Spain, pp. 1-10, May 9-11, 2012.

[7] D. Bradley, R. Harper, and S. Hunter, "Workload-Based Power Management for Parallel Computer Systems," *IBM Journal of Research and Development*, 47(5):703-718, 2003.

[8] D. P. Bunde, "Power-Aware Scheduling for Makespan and Flow," *Proc. the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, Cambridge, USA, pp. 190-196, July 2006.

[9] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, 25(6):599-616, June, 2009.

[10] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing Server Energy and Operational Costs in Hosting Centers," *ACM SIGMETRICS Performance Evaluation Review*,

33(1):303-314, June, 2005.

[11] J. Cohen, "Cooperation and Self-Interest: Pareto-Inefficiency of Nash Equilibria in Finite Random Games," *Proc. of The National Academy of Sciences of the United States of America*, 95(17):9724-9731, August 18, 1998.

[12] S. Darbha and D. P. Agrawal, "Optimal Scheduling Algorithm for Distributed-Memory Machines," *IEEE Transactions on Parallel and Distributed Systems*, 9(1):87-95, Jan. 1998.

[13] T. Duy, Y. Sato, and Y. Inoguchi, "Performance Evaluation of a Green Scheduling Algorithm for Energy Savings in Cloud Computing," *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*, Atlanta, GA, USA, pp. 1-8, April 19-23, 2010.

[14] E. N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters," *Proc. of the 2nd International Conference on Power-Aware Computer Systems*, Cambridge, USA, pp. 179-197, 2003.

[15] D. Filani, S. G. J. He, M. Rajappa, A. Kumar, P. Shah, and R. Nagappan. "Dynamic Data Center Power Management: Trends, Issues, and Solutions," *Intel Technology Journal*, 12(1):59-68, 2008.

[16] J. Friedman, "A Non-Cooperative Equilibrium for Supergames," *The Review of Economic Studies*, 38(1):1-12, January 1971.

[17] S. Garg, C. Yeo, A. Anandasivam, and R. Buyya, "Energy-Efficient Scheduling of HPC Applications in Cloud Computing Environments," *arXiv preprint arXiv:0909.1146*, 2009

[18] Gartner Newsroom, Gartner says Worldwide Cloud Services Market to Surpass $68 Billion in 2010, http://www.gartner.com/it/page.jsp?id= 1389313," 2010.

[19] R. Ge, X. Feng, and K. W. Cameron, "Performance Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters," *Proc. of the ACM/IEEE Conference on Supercomputing*, p. 34, Nov. 12-18, 2005.

[20] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini, "Energy Conservation in Heterogeneous Server Clusters," *Proc. of the 10th ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, Chicago, IL, USA, pp. 186-195, 2005.

[21] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control," *IEEE Transactions on Computers*, 56(4):444-458, April, 2007.

[22] Tai-Yi Huang, Yu-Che Tsai, and Chu E.T.-H, "A Near-Optimal Solution for the Heterogeneous Multi-Processor Single-Level Voltage Setup Problem," *IEEE International Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, pp. 1-10, March 26-30, 2007.

[23] Chi-Hong Hwang and A. C.-H Wu, "A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation," *IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, USA, pp. 28-32, Nov. 9-13, 1997.

[24] R. Jejurikar and R. Gupta, "Dynamic Slack Reclamation with Procrastination Scheduling in Real-Time Embedded Systems," *Proc. of the 42nd Conference on Design Automation*, pp. 111-116, June 13-17, 2005.

[25] S. Khan and I. Ahmad, "A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids," *IEEE Transactions on Parallel and Distributed Systems*, 20(3):346-360, Oct. 2009.

[26] K. Kim, R. Buyya, and J. Kim, "Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-Enabled Clusters," *Proc. of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, Rio de Janeiro, Brazil, pp. 541-548, May 14-17, 2007.

[27] Z. Lan, and Z. Zhao-xia, "A Deadline and Cost Optimization Algorithm for Scheduling Task in Grids and Nash Equilibrium in Auction-Based Systems," *Chinese Control and Decision Conference (CCDC)*, Yantai, Shandong, China, pp. 2431-2436, July 2-4, 2008.

[28] B. Lawson and E. Smirni, "Power-Aware Resource Allocation in High-End Systems via Online Simulation," *Proc. of the 19th Annual International Conference on Supercomputing*, Cambridge, USA, pp. 229-238, 2005.

[29] Y. C. Lee and A. Y. Zomaya, "A Novel State Transition Method for Metaheuristic-Based Scheduling in Heterogeneous Computing Systems," *IEEE Transactions on Parallel and Dist. Systems*, 19(9):1215-1223, Sept. 2008.

[30] Y. C. Lee and A. Y. Zomaya, "Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling," *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, Shanghai, China, pp. 92-99, May 18-21, 2009.

[31] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson, Scheduling Strategies for Mapping Application WorMows onto the Grid, *Proc. of the IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pp. 125-134, 2005.

[32] J. Nash, "Non-Cooperative Games," *The Annals of Mathematics, Second Series*, 54(2):286-295, 1951.

[33] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting Platform Heterogeneity for Power Efficient Data Centers," *Fourth International Conference on Autonomic Computing (ICAC)*, Jacksonville, FL, USA, p. 5, June 11-15, 2007.

[34] Nimbus, http://nimbusproject.org.

[35] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," *Proc. of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'09)*, Shanghai, China, pp. 124-131, May 18-21, 2009.

[36] Open Nebular, http://www.opennebula.org.

[37] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath, "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," *Proc. of the Workshop Compilers and Operating Systems for Low Power (COLP)*, 180:182-195, 2001.

[38] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, and M. Schulz, "Bounding Energy Consumption in Large-Scale MPI Programs," *Proc. of the ACM/IEEE Conference on Supercomputing*, Reno, NV, USA, pp. 1-9, Nov. 10-16, 2007.

[39] C. Rusu, A. Ferreira, C. Scordino, and A. Watson, "Energy Efficient Real-Time Heterogeneous Server Clusters," *Proc. Real Time and Embedded Technology and Applications Symp. (RTAS '06)*, pp. 418-428, April 4-7, 2006.

[40] E. Seo, Y. Koo, and J. Lee. "Dynamic Repartitioning of Real-Time Scheduleon a Multicore Processor for Energy Efficiency," *Proc. Of the International Conference on Embedded and Ubiquitous Computing*, Seoul, Korea, pp. 69-78, 2006.

[41] T. Simunic, "Dynamic Management of Power Consumption," *Power Aware Computing*, pp. 101-125, 2002.

[42] M. O. Spata, "A Nash-Equilibrium Based Algorithm for Scheduling Jobs on a Grid Cluster," *16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Evry, France, pp. 251-252, June 18-20, 2007.

[43] M. Srivastava, A. Chandrakasan, and R. Brodersen. "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Transaction VLSI Systems*, 4(1):42-55, March, 1996.

[44] H. Topcuouglu, S. Hariri, and M.-Y. Wu, "Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260-274, March 2002.

[45] I. B. Vapnyarskii, "Lagrange Multipliers," Hazewinkel, Michiel, *Encyclopedia of Mathematics*, 2001.

[46] V. Venkatachalam and M. Franz, "Power Reduction Techniques for Microprocessor Systems," *ACM Computing Survey*, 37(3):195-237, September 2005.

[47] L. Wang and Y. Lu, "Efficient Power Management of Heterogeneous Soft Real-Time Clusters," *Proc. of the Real-Time Systems Symposium*, Barcelona, Spain, pp. 323-332, Nov. 30 -Dec. 3, 2008.

[48] H. Yaïche, R. Mazumdar, and C. Rosenberg, "A Game Theoretic Framework for Bandwidth Allocation and Pricing in Broadband Networks," *IEEE/ACM Transactions on Networking (TON)*, 8(5):667-678, Oct. 2000.

[49] Y. Yu and V. K. Prasanna, "Power-Aware Resource Allocation for Independent Tasks in Heterogeneous Real-Time Systems," *Proc. of the 9th IEEE International Conference on Parallel and Distributed Systems*, pp. 341-348, Dec. 17-20, 2002.

[50] D. Zhu, R. Melhem, and R. Bruce, "Scheduling with Dynamic Voltage/Speed Adjustment using Slack Reclamation in Multiprocessor Real-Time Systems," *IEEE Transaction on Parallel and Distributed Systems*, 14(7):686-700, July 2003.

[51] M. Zhu, Q. Wu and Y. Zhao, "A Cost-Effective Scheduling Algorithm for Scientific Workflows in Clouds," *IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, Austin, TX, USA, pp. 256-265, Dec. 1-3, 2012.

[52] A. Y. Zomaya, C. Ward, and B. S. Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues," *IEEE Transactions on Parallel and Distributed Systems*, 10(8):795-812, Aug. 1999.

Mustafa Khaleel received the B.S degree in Computer Science from Salahaddin University, Erbil, Iraq, in 2006, and Master degree from the University of Sulaimani, Sulaimani, Iraq, in 2008. He is currently a Ph.D student in the Department of Computer Science, Southern Illinois University, U.S.A. He is also a lecturer in the Department of Computer Science at Sulaimani University. His interests are in cloud computing system, wireless communication system, network security, game theory and robotics.

Saad Alqithami is currently a PhD. candidate in the Department of Computer Science, Southern Illinois University, where he had his Masters with honor in 2012. He is also a lecturer in the Department of Computer Science and Information Systems at Albaha University. His interests are in distributed AI, multiagent systems, game theory and robotics. He is also a Member of IEEE, ACM and others that can be found on his website (*http://www2.cs.siu.edu/~salqithami*).

Michelle M. Zhu received her Ph.D. degree in Computer Science from Louisiana State University in 2005. She worked as a Research Associate in the Computer Science and Mathematics Division at Oak Ridge National Laboratory on her Ph.D. dissertation from 2003 to 2005. Dr. Zhu is currently an Associate Professor in Computer Science Department at Southern Illinois University at Carbondale. Her research interests include grid and cloud computing, distributed and high performance computing, remote visualization and bioinformatics.

Dunren Che is currently a Professor in Computer Science at SIUC (Southern Illinois University Carbondale), USA. He had earned a PhD in Computer Science from the Beijing University of Aeronautics and Astronautics, Beijing, China in 1994, and had numerous years of post-doc research experience thereafter until he joined SIUC in 2001. His main research interests include database (especially XML database and query processing/optimization) and data mining, as well as cloud computing and big data as his most recent emphasis.

Wen-Chi Hou received the MS and Ph.D. degrees in Computer Science and Engineering from Case Western Reserve University, Cleveland Ohio, in 1985 and 1989, respectively. He is presently Professor of Computer Science at Southern Illinois University Carbondale. His interests are in statistical databases, mobile databases, XML databases, and data streams processing, and query optimization.

Moving Energy Consumption Control into the Cloud by Coordinating Services

Genoveva Vargas-Solar*
CNRS, LIG-LAFMIA labs, Grenoble, FRANCE

Catarina Ferreira da Silva†, ParisaGhodous†
Claude Bernard Lyon I University, Lyon, FRANCE

José-Luis Zechinelli-Martini‡
Universidad de lasAméricas Puebla, San Andrés Cholula, MEXICO

Abstract

In this paper we present a cloud-based service oriented approach for collecting, integrating, storing, and analyzing energy consumption data. Our approach models energy sensors as services that can be composed to provide value added information with various granularity levels that best suit users' needs and requirements: home-owners, energy providers, local and regional planning authorities, etc. The resulting system is a layer oriented service network where each layer provides information at different levels of aggregation based on a polyglot persistence approach.

Key Words: Cloud computing, data integration, service based querying, smart energy.

1 Introduction

Cloud computing has recently emerged as a new computing paradigm where unlimited computing and storage resources can be allocated for building and delivering applications and services over the Internet. Cloud infrastructures manage such resources transparently without requiring the application to have code to manage them or to reserve more resources than those it really requires. The difference with the conventional paradigms is that the application can have an ad hoc execution context and that the resources it consumes are not necessarily located in one machine. Cloud infrastructures provide data management functions as services that must be tuned and composed for efficiently and costly managing, querying and exploiting huge data sets (e.g., big data).

Consider a smart home scenario where intelligent control technology enables homeowners to monitor and reduce energy consumption of smart home electronics, conserve resources, and save money without sacrificing comfort or convenience. Smart energy monitors measure energy consumption and production in real-time, and exploit the histories of available energy management devices to provide consumers and managers with real-time information on electricity use and costs. A homeowner can monitor each and every aspect of electricity usage, from appliances to heating and lighting, and view her entire electricity usage or production at home or remotely. On a larger scale, energy providers, local and regional planning authorities can follow the behavior and the energy consumption trends of energy consumers to provide new energy provision plans, facilities and costing models more adapted to consumers needs and requirements. However, providing such monitoring and analysis capabilities involves handling considerable amounts of raw data that need to be processed, analyzed and stored. Moving data aggregation and analysis to the cloud can be interesting for many reasons. First, it allows process of huge amounts of produced data in an efficient way with the existence of unlimited and adaptable computation and storage resources. Second, it can provide an ad hoc personalized energy consumption analysis to different types of users.

In this paper we present a multi cloud-based service oriented approach for collecting, integrating, storing, and analyzing energy consumption data. Our approach models energy sensors as services that can be composed to provide value added information with various granularity levels that best suit users' needs and requirements: home-owners, energy providers, local and regional planning authorities, etc.

The remainder of the paper is organized as follows. Section 2 analyses related works and puts our work in context with respect to existing results regarding data integration, polyglot persistence and query rewriting. Section 3 describes our approach that is based on the notions of *view* for modeling data, and computations; and *strata* for describing the aggregation levels associated with data related to energy consumption. Section 4 presents our three-layer service architecture providing a polyglot data store [6] for storing energy consumption data histories and associated models. Section 5 concludes the paper and discusses future work.

* LIG-LAFMIA, 681 rue de la Passerelle, 38402 Saint Martind'Hères, France, genoveva.vargas@imag.fr.
† Department of Informatics, 43 bvd du 11 novembre 1918, 69622 Villeurbanne, France, catarina.ferreira@univ-lyon1.fr, parisa.ghodous@recherche.univ-lyon1.fr.
‡ Department of Computing, Electronics and Mechatronics, Ex-hacienda CatarinaMártir s/n, 72820 San Andrés Cholula, Mexico, joseluis.zechinelli@udlap.mx.

2 Related Work

Cloud storage represents a paradigm to store, retrieve and manage large amounts of data, using highly scalable distributed infrastructures. This area has received a great deal of attention in recent years, due to a growing interest in the challenges and opportunities associated to the NoSQL movement [2]. However, unlike traditional environments, where the use of the relational model is pervasive, there is a wide variety of data models that can be used in cloud applications. These data models include [2]: key-value, document, extensible record, graph and relational repositories. Each of these data models are designed for different use cases, and provide different support for functional and non-functional requirements of distributed systems [3], such as different degrees of consistency, scalability, replication and concurrency [2]. Moreover, there is also a wide variety of both public and private providers for the distributed infrastructure that is required for cloud data storage [9]. These providers offer different combinations of pricing, support, service levels, and usually have different APIs to store, retrieve and manage data. These differences make it difficult to design and deploy applications targeting different cloud environments [10]. In our polyglot system we use existing SpringRoo binding generation tools and we also developed bindings that were plugged in this environment. The idea is to couple our integration rewriting strategies with the spring code that implements the actual calls to the NoSQL stores participating in the polyglot solution.

Query rewriting using views (a.k.a. query answering using views) is the process of reformulating a query Q expressed over a mediated schema in terms of a set of views $V_1...V_n$ expressed over the same schema [7] (where a view is a named query). The obtained query is called a *rewriting*. The problem of query rewriting using views has been considered for two different purposes: (i) query optimization using materialized views, and (ii) data integration.

In the context of query optimization, the goal is to find an expression that uses the materialized views (which represent cached data) and is *equivalent* to the original query. The rationale behind query reformulation here is that using cached data (i.e., the materialized views) is much faster than accessing the actual database relation directly. In the context of data integration, the views describe a set of autonomous heterogeneous data sources. Users queries over the mediated schema need to be formulated to refer to the data sources the mediated schema itself does not contain any data.

In this context we usually cannot find a rewriting that is equivalent to the user query because of the data sources limited coverage (i.e., the data inside data sources are incomplete). Instead, we search for a ”maximally contained rewriting”, which provides the best answer possible, given the available data sources. When both the query and the views are conjunctive queries, the maximally contained rewriting is the union of all rewritings that are possible given the views. Different query rewriting algorithms were proposed in the literature including, the *MiniCon* [8], *Inverse Rules* [5] and *Bucket* algorithms for the relational model, [12] for XML queries and recently [3] for RDF queries.

An advantage of modeling services as views is that queries can be resolved "*on the fly*" by combining relevant services using a query rewriting algorithm (e.g., *Inverse Rules*, *Minicon*, *Buckets*, etc.) [7]. Similarly, value-added aggregated views (that could be needed in higher layers) can be constructed and populated on the fly. Application developers need only to express their data needs as queries over the global schema, the query rewriting algorithm can then select the relevant services and combine them to answer the queries; i.e., application developers are relieved from the painstaking task of selecting and combining services manually.

We believe that the challenges introduced by energy data integration must be supported both by cloud based polyglot persistence and query rewriting techniques as shown in the following sections.

3 Service-Oriented Approach for Collecting and Integrating Energy Data

Figure 1 shows an overview of our approach for energy data integration. In our approach, energy sensors organized as an observation network are represented as services (called *Sensing Services* see 1 in Figure 1). The semantics of sensing services are modeled as relational views. Our approach combines the data produced by sensing services to provide value added integrated views with different aggregation levels, called *Strata* (see 2 in Figure 1).

Continuing with the example scenario described in the introduction, in-house sensors and smart energy counters in a monitored area form a network of services that can be combined to construct *strata*. *Strata* provide useful information about, for example, the average energy consumption (per hour) at the scales of *room*, *house*, *blocks of houses*, *quarter*, *city*.

Since the construction of *strata* necessitates considerable processing and storage capabilities (as it is performed on huge data histories), our approach relies on computing services (e.g., data transformation services, indexation services, etc. see 3 in Figure 1) and on a polyglot [6] distributed data store to manage data histories and their associated analysis results. We define below the notions of view and *strata* that are fundamental in our approach.

3.1 Services and Views

This section introduces how data produced on demand and continuously by data services are modeled using the notion of view. Views are then used to compute Strata that provide aggregated views of data. Such aggregations are done by computing services defined in the following lines.

Sensing Services: are data services that represent the sensors in the monitored area. We model the semantics of a sensing service as a *relational view* over a mediated schema.

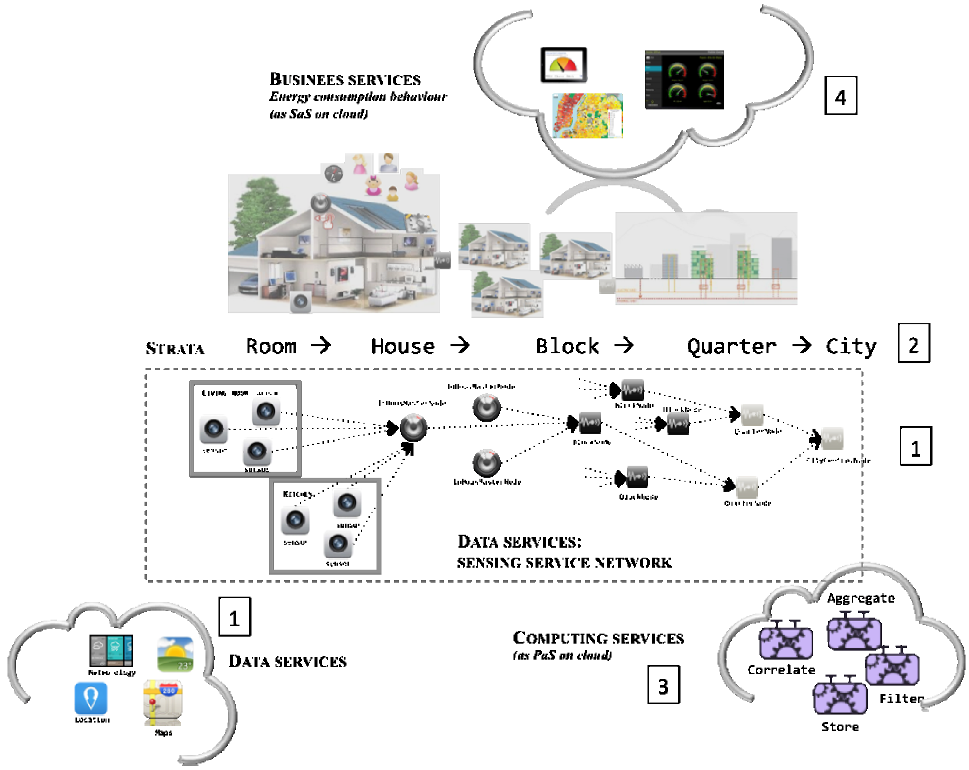


Figure 1: Stratified data flow consumption and delivering for analyzing energy consumption behavior

Formally, a sensing service is defined as:

$$WS(X^b, Y^f)\text{:- } R_1(X, Y, Z),\ldots, R_n(X, Y, Z)$$

Where $WS(X^b, Y^f)$ is the view head, it is a relational predicate containing the service inputs X^b and outputs Y^f. Inputs should be bound in order to invoke the service, therefore marked with the superscript b, outputs are free, therefore marked with the superscript f; R_i is a relational predicate and X, Y, and Z are attributes. For example a service monitoring the status of an air conditioner is represented as follows:

Air Conditioner WS (time^b, status^f, temp^f) : -
Apparatus(status, time, location),
Temperature(time, temp, location),
Location = home

A service monitoring the presence of people in a given location is represented as follows:

Presence WS(time^b, location^b, status*f*) : -
Person(status, time, location)

Concretely, the data returned by sensing services are stored in views (that correspond to the view heads in the previous definition) in a polyglot data store that we present in subsequent sections.

Sensing services can produce data on demand or as streams according to their exported interfaces. Data is gathered from on-demand data services by invoking their methods with the appropriate parameters, producing tuples as output. Stream services export subscription methods that after invocation, will produce a stream. For example, a **location** service is a streaming service that exports:

subscribe()→⌈**location**:⟨id, coor⟩⌉

which is a subscription method that after invocation, will produce a stream of **location** tuples with a nickname that identifies the object coordinates. Note that a stream is a continuous (and possibly infinite) sequence of tuples ordered in time.

Computing Service: performs data management and processing tasks (e.g., data analysis, indexation, storage, etc.) or particular calculations (e.g., mathematical functions), which can be useful for processing data. These operations are used for computing data aggregations, correlations and other processing operations necessary for providing an analytic view of energy consumption.

Computing services can be *simple* or *composite*. They are simple when they provide a basic functionality. For example, a **distance** computation service computes the geographical distance between two points, for instance, by using Vincenty's formula[2].

They are composite when they combine multiple simple or composite services to realize a complex functionality. They are specified as a workflow-based service coordination of basic computation services. This approach enables us to take advantage of existing services for programming more complex data processing operations. By developing data processing

[2] http://en.wikipedia.org/wiki/Vincenty%27s_formulae

operations by either simple or composite computation services, we can develop the core functionality required for observing energy consumption.

As we will describe in the following lines, data and computing services are coordinated to answer hybrid queries used for expressing data consumption requirements. Details on how these computation services are built and implemented are out of the scope of this paper. The interested reader can refer to [4] for details.

3.2 Stratified Data Integration

As we mentioned, a monitored area corresponds to a network of sensing services (refer to Figure 1). The data produced by these services are aggregated to form data (providing) services (or simply views) with different levels of granularities that we call *Strata*. *Strata*, simply, provide a logical organization of data provision represented by a hierarchy of aggregated granularities. A granularity denotes a set of (complex) types (its extension). In this work, we identified the following *strata*:

room → house → block → quarter → city

Sensing services that are located in the same room form a data service corresponding to the *Room stratum*. The different room-level data services in a given house form a data service called *inHouseMasterNode* that corresponds to the *house stratum*. Similarly, *inHouseMasterNodes* can be grouped into *block*, *quarter* and *city strata*. Data services of the types *house*, *block*, *quarter* and *city* can have attributes to characterize their geographic locations.

A granularity also has an associated aggregation function that applies to the aggregated data (i.e., the aggregation function computes the tuples set that is stored in the view based on the sensed data). Examples of aggregation functions include: *the average energy consumed during a day in the kitchen for all the days of the year, the pick of consumed energy during a day in the living room during winter*.

The hierarchy of *strata* also defines transformation functions among granularities. The functions for the *strata room* are classic aggregation functions like average or maximum and they are computed on data windows. Transformation functions among *room → house → block → quarter → city* are defined by statistical analysis that compute the behavior of energy consumption using the data of the lower level as input for computing the measure of a more general level. These computations are done by computing services.

Data consumers can access data by combining data from the same or more general granularities. As shown in Figure 1, in our work we rely on a logical network of data services that are devices represented logically grouped for defining the *stratum room*. For example, sensing services are connected to a device with more computing capacity that is connected with the external world called the *inHouseMasterNode* and that represents the *stratum house*. So, this *strata* provides an aggregated view of the energy consumption in a whole house during specific periods of time.

The nodes of type *inHouseMasterNode* are also services that form networks organized in layers called *block*, *quarter* and *city*. *inHouseMasterNodes* can be geographically located and they can be logically organized according to spatial geographic regions that denote either their location (lowest granularity), and then concentric regions grouped into quarters, and cities. The organization of the network and the computing capacities of the services are exploited to have different levels of aggregation and analysis views of such data.

3.3 Consuming Data

Energy consumption data can also be correlated with data stemming from other homes in order to determine the behavior on energy consumption of communities of homes, quarters, cities, regions and countries. More critical decision making for determining how to deliver energy to consumers can be done using such information.

"Software as a service" like solutions interact with these nodes for providing analysis and decision making support to different actors. For example *give me the a graphic representing the average energy consumption between 17:00 - 23:00 during summer of the private consumers living at rue Alembert in Grenoble*.

```
meteringDashboardService (nodeID^b, userRoleb, timeWindow^b,
      GraphicTypeb, GraphicFlowFunctinf): -
AggregatedConsumptionViewPerRole (nodeID, userRole,
      timeWindow),
ConsumptionOverTime(nodeID, timeWindow, GraphicFlowFunction)
```

This hybrid query [4] expresses data consumption requirements. A hybrid query is a query that can be mobile and continuous, and evaluated on top of on demand or streaming static or nomad data services [11].

An hybrid query combines data from the rooms of a house and a meteorology service providing information about the region where my house is located. In our approach the hybrid query is first expressed in Data log as shown in the above expression and rewritten according to available services.

The evaluation of such type of queries requires data services but also storage and computing services that can be used for logging continuous data. This can be useful for performing aggregations on data collected on given time windows.

For example data can be correlated with meteorological data histories to identify the time windows where:

$$-5 \leq \text{temperature or temperature} \geq 30.$$

The query is rewritten according to the available services exported views. In the case of our example, there are three services of type *BlockNode* and a *MapService* shown below.

Query:

```
Q1⟨Average, client⟩:- AveragePerUser (average, user, timeWindow,
                     zip), StreetZips (streetName, zipA, zipB),
      zipA <= zip <= zipb, timeWindow="17:23", StreetName –
                     'Alembert'.
```

Data services:

MapService(streetName^{b}zipA^{f}zipB^{f}) : StreetZips(streetName, zipA, zipB)

BlockNodes:

BN_1(average, user, timeWindow) : AverageaPerUser(average, user, timeWindow, sip), zip=69101
BN_2(average, user, timeWindow): AveragePerUser(averagae, user, timeWindow, sip), zip=69106
BN_3(average, user, timeWindow): AveragePerUser(average, user, timeWindow, zip), sip=20100

The query is rewritten into two sub-queries expressed in Data log below: the first one retrieves the region in which *rue d'Alembert*is located; the second one computes the average energy consumption per user (house) within a predefined time window and filters the result with respect to the geographic location.

$Q_1 = Q_2 \wedge Q_3$
Q_2(zipA, zipB) :- Streetzips(streetName, zipA, zipB, StreetName="Alembert".
Q_3(average, client) :- AveragePerUser(average, user, timeWindow, zip), zipA <= xip <= zipB.

A hybrid query is implemented by a query workflow that coordinates services for consuming and retrieving data in a one shot or a continuous manner. The query workflow (see Figure 2) is a program that runs continuously for executing the query and generating new results. In a query workflow, activities can call several services for computing the average consumption of users located within a specific geographic region and at a specific time interval (i.e., [17:00, 23:00]). The query workflow runs as a data processing service and is supported by a polyglot data store service for storing partial and final results (see the following section).

Home control and energy consumption observation need huge amounts of heterogeneous data flows produced by data services (sensors, temperature and meteorology services) that must be processed and stored by computing services. We addressed the storage problem by defining a polyglot data store solution based on NoSQL and relational models, as shown in the following section.

4 Description of our Cloud-Based Architecture

This section describes the implementation of our approach. The use of multiple and heterogeneous data stores within a single information system is a common practice in real-life application development. Modern applications very often rely on a polyglot approach [6] to data persistence, where conventional databases, non-relational data stores, and scalable systems associated to the emerging New SQL movement, are used simultaneously. We followed this approach for building our system and we adopted a service-oriented multi-cloud architecture for deploying our solution. We implemented a three layer system that integrates a data provision layer with a SaaS layer, thanks to a data integration layer implemented as a polyglot database system (see Figure 3).

As shown in the Figure 3, our system is comprised of three Spring Java web applications that are in charge of different data collections, and expose services through REST interfaces (i.e., the metering dashboard, the energy business intelligence analysis, the energy load control). These business services rely on data services, such as on the sensing services and aggregate this information. For instance, the energy load services (composite activity). In Figure 2 the activity *Get* control relies on the information of the sensing service and on business rules to act on actuators that can automatically reduce temperature of an air conditioner system. The applications are deployed in different Platform as a Service (PaaS) providers, and access data through Database as a Service (DaaS) vendors providing NoSQL data stores: relational, document and graph databases, deployed on multiple cloud providers (OpenShift, CloudFoundry, Xeround and MongoLab).

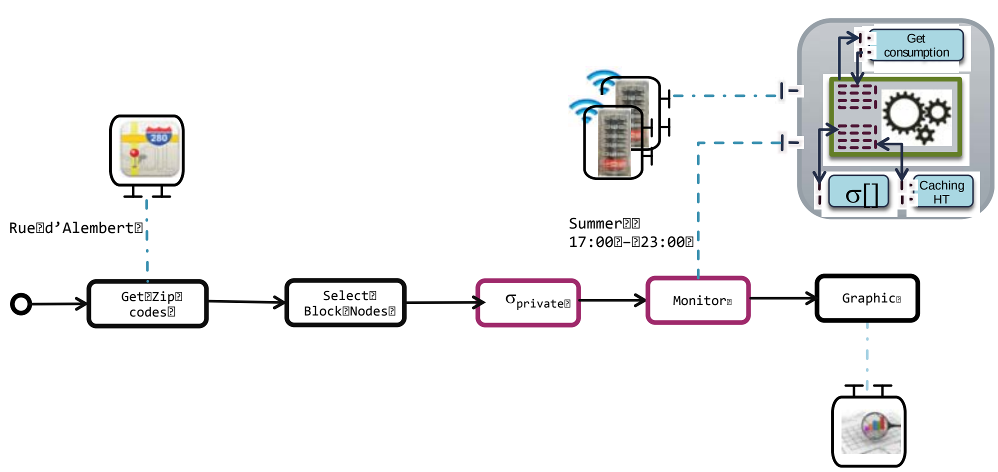


Figure 2: Query workflow example

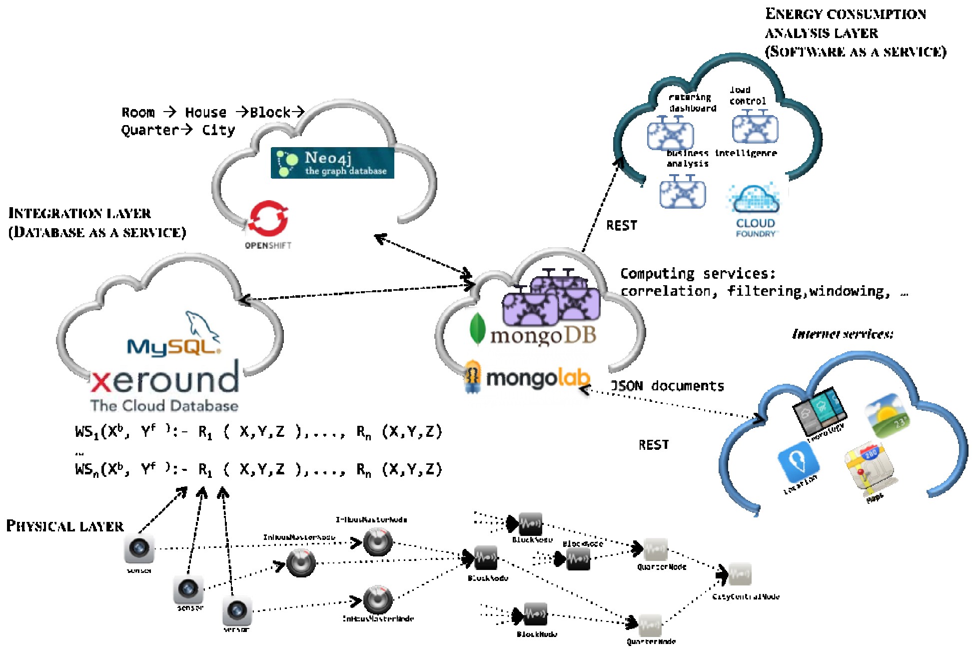


Figure 3: Energy data management multi-cloud polyglot architecture

4.1 Physical Layer

The physical layer is composed of services for collecting raw energy consumption data from sensors and electrical devices. Each sensing service collects data from one or more sensors and sends its measurements as messages via Internet protocols (i.e., XML messages in SOAP for SOAP-based service implementations). Services are clustered in the physical layer based on their functionalities and registered in a service registry. These services communicate with a MySQL server deployed on Xerund for periodically storing their data.

The services are proprietary devices of an energy company and because of confidentiality issues we cannot give technical details of their characteristics. A sensor is a monitoring device programmed for reading analogical data that can be transformed to a digital representation. A sensor has specific computing, storage, information transmission/reception capacities and limited energy. In the paper, it is enough to say that sensors that are wrapped as OSGi services (www.osgi.org) exporting an interface that enables the retrieval of measures from the sensor buffer.

We profit from the OSGi technology for building a sensor network that integrates the data they produce in the so-called InHouseMasterNode. The InHouseMasterNode is a sensor with more storage and computing capacity that communicates recurrently with the data integration layer deployed in the Xerund cloud provider for flushing data histories. The InHouseMasterNode serves as global controller for synchronizing sensors so that they can beat under the same global clock. The views associated to services (sensors and InHouseMasterNodes) are recurrently computed due to the arrival of new data flows. This is done by services that are continuously observing data consumption by interacting with the data integration for storing views given their reduced storage capacity. The arrival of new data triggers views computation.

4.2 Data Integration Layer

Data integration and processing requires alot of storage and computing capabilities as well as data processing functions that can vary according to the analysis requirement of different consumers. This layer implements a polyglot approach for integrating data from different services to provide value added information.

Data integration is based on a data pivot model [1] associated to a polyglot distributed database. The data model used in this layer relies on four main constructs (Structs, Sets, Attributes and Relationships), that can be used to represent data modeled using the key-value, document and column-family and graph data models. Data stemming from different NoSQL stores can be transformed into this model and made available to the application. In our scenario the representation of the logical nodes network is stored on the graph oriented Neo4J vendor deployed on Open shift cloud provider. The data produced by computation and data services that in general produce JSON documents, are stored on the document store MongoDB deployed on the MongoLab cloud provider.

Graph Database: the information about the devices networks at different levels is managed by the system Neo4J that supports graph oriented databases. The description of the networks organized by *strata* where each *stratum* is a graph managed by a service that stores it persistently for maintaining information about the network state: which nodes join or leave the network. At the creation of the system, the graph database was tuned using the Eclipse UML tool. Given the classes that implement the service functions for managing information about the networks of the different *Strata* (e.g., sensor networks), we use the Model2Roo plugin[3] for generating a SpringRoo[4] binding to the Neo4J data store.

For the time being the graph database serves for answering queries and guiding the aggregation of data. For example given a query asking for the average energy consumption of the rooms that are near the kitchen in my house, the graph database will help to determine which are the sensors that will participate as data providers for answering the query (i.e., the sensors that are installed in the room for solving the query presented in the previous section). The database is updated every time nodes adhere or leave the network. This is not very often for the time being because we consider that the network is rarely modified. In a future version of our system we will consider that the network is dynamic and that this database will have to be updated.

Document Database: Our system uses information stemming from Web services, for example the meteorology service, for correlating the data produced by the energy consumption physical layer. For example, for determining that the temperature sensed in a house corresponds in fact to the second week of summer 2013. The meteorology data are recurrently retrieved according to specific geographical locations and points in time (hours, weeks, seasons). These services produce data as JSON documents that are stored in the MongoDB document database. As for the graph data store we used tour Model2Roo plug in for configuring the database and generating the SpringRoo binding for storing the documents produced by the Web services.

Polyglot Database System: integrates these databases into a global view used for querying and exploiting them. As said above, these stores are populated as new data arrive from the networks. Views associated to services are computed recurrently and stored in Neo4J. The polyglot database system enables then the evaluation of queries on continuous data. Therefore, our system exploits query-rewriting techniques to automatically determine the data services that are needed to answer data requests. For instance, to determine the services for constructing a desired *stratum*, or for answering a given data analysis query. This is possible as the semantics of our services are modeled as relational views. *Strata* developers and data analysis applications need only to specify their data needs as queries over a mediated schema. Then, the system rewrites that query in terms of calls to relevant services. Our system uses the *MiniCon*query rewriting algorithm [8].

Running Example: assume we are interested in studying the energy extra consumption related to the use of cooling systems in summer. At the *InHouseMasterNode* level, we are interested in constructing a view (or a service) to observe the working of air conditioners, along with the house temperature and whether or not there are people in proximity of conditioners. Such data needs can be expressed using the global schema (in the Data log notation) as follows:

InHouseMasterNode_View1 (time,status,temp,presence,location):
Apparatus (status, time, location),
Temperature (temp, time, location),
Presence (presence, time, location),
location = 65266 Lyon

Assume the existence of the following services:

- ***Service observing an air conditioner:***

ACWS(time^{b}, status^{f}) : -
Apparatus (status, time, location),
location = 65266 Lyon

- ***Service observing the house temperature:***

TempWS (time^{b}, temp^{f}) : -
Temperature (temp, time, location),
location = 65266 Lyon

- ***Service observing the presence of people:***

PresenceWA (time^{b}, presence^{f}) : -
Presence (time, presence, location),
location = 65266 Lyon

Given these services, the query-rewriting algorithm rewrites the *InHouseMasterNodeView*1 as follows:

InHouseMasterNode_View1 (time,status,temp,presence,location):-
ACWS (time^{b}, status^{f}),
TempWS (time^{b}, temp^{f}),
PresenceWS (time^{b}, presence^{f}),

Similarly, *blockNode Strata* views can be constructed using *InHouseMasterNode Strata* views. For instance, assumes we are interested in constructing a block view observing the working of air conditioners in houses located between 65250 Lyon and 65260 Lyon. Such view can be expressed as follows over the global schema:

BlockNode_View (time, status,temp, location) : -
Apparatus (status, time, location),
Temperature (temp, time, location),
65250 Lyon <location< 65260 Lyon

Such view could be rewritten in terms of the *InHouseMasterNode*views as follows:

BlockNode_view(time, status, temp, location) : -
InHouseMasterNode_View1 (time,status,temp,-,65251 Lyon)
InHouseMasterNode_View2 (time,status,temp,-,65252 Lyon)
…

Once the query has been rewritten the system generates a workflow using another rewriting algorithm that transforms Data log expressions into a query workflow. This algorithm is out of the scope of this paper, but the interested reader can see details in [4]. The workflow can implement continuous or one

[3]http://code.google.com/p/model2roo/
[4]http://www.springsource.org/

shot queries, according to the arrival rate of new data to the stores. The data integration layer serves as mediator between the physical layer. The physical layer produces data and the energy consumption analysis layer that consumes data. The layer consists of services that implement analysis applications that deliver information to final users. This layer is described in the following section.

4.3 Energy Consumption Analysis Layer

The energy consumption analysis layer implements the business logic to offer decision maker assistance applications to homeowners and planning authorities. This layer is deployed on the CloudFoundry cloud provider. The business services made available by this layer compose external information (such as energy tariff) with the data provided by the sensing services of the data integration layer of our architecture. These business services are the following:

The Metering Dashboard provides graphical energy monitoring by exhibiting the analyzed energy consumption behavior. It provides aggregated information concerning energy consumption about specific zones such as rooms, or aggregated views of building and city zones. The metering dashboard also gives the ability to alert excessive energy consumption provided that the user has previously defined corresponding thresholds. Graphical functionalities are configured with Google Charts Visualization API.[5]

The Energy Business Intelligence Analysis supports managers in the decision making process. It offers energy benchmarks by combining the energy consumption information with energy tariff information, pricing and peak demand usage. This business intelligence service generates energy audit reports and provides energy consumption simulation forecasts based on past energy usage.

The Energy Load Control implements energy saving strategies for automating local load control e.g., automatically turning off room lights if enough daylight is available and if the preset energy threshold is exceeded. This service enables the definition of periodic schedules to automatically control actuators over facilities. Each schedule communicates with the corresponding actuators that can be programmed for scheduled on/off periods. For this, the energy load control combines the information provided by the sensing services and specific business rules to trigger actuators that will automatically take some action, such as reduce temperature of air conditioner systems. Consider the case of the energy manager who wants to automatically turn off air conditioner systems of office rooms if no person is inside after working hours. The following rule is executed: if the service observing air conditioner ACWS($time^b$, $status^f$) returns *On* for specific office room locations and the service observing the presence of people Presence WS($time^b$, $presence^f$) returns *False* then the corresponding actuators will turn these air conditioner systems off.

[5] https://developers.google.com/chart/interactive/docs/reference?hl=en

5 Conclusion and Future Work

This paper presented an approach for collecting and integrating data produced by networks of energy consumption for the purpose of providing aggregated data on energy consumption. This energy information can be further used to manage energy consumption and reduce energy waste.

Our approach relies on the notions of view and *strata* for describing on demand and continuous data producers where data can be relational, streams, documents and produced on-demand and continuously.

The main contribution of our work is the proposal of a three layer architecture that relies on a polyglot service based data management system that benefits from the flexibility of the cloud for deploying services for processing and analyzing of collected energy consumption data.

We provide a service-oriented approach for our cloud-based architecture that provides a transparent access to autonomous services with their own resources.

Beyond the application of energy consumption, we are currently addressing data management issues on the cloud. Particularly, concerning polyglot persistence, we have developed tools Model2Roo[6] and ExSchema[7] for supporting the definition of polyglot data stores and its maintenance.

We are also addressing the implementation of data processing operations using map-reduce models for better addressing the analysis and correlation of huge volumes of data given a certain “unlimited” availability of computing resources on the cloud. These current actions are being tested and tuned for dealing with energy data management.

Acknowledgements

We thank Mahmoud Barhamgi, University Claude Bernard, Lyon, France for his precious help in the preparation of this paper, for his helpful discussions and clever advises.

References

[1] J. Castrejon, G. Vargas-Solar, C. Collet, and R. Lozano, “Model-Driven Cloud Data Storage,” First International Workshop on Model-Driven Engineering on and for the Cloud, 2012.

[2] R. Cattell, “Scalable SQL and NoSQL Data Stores,” *SIGMOD Rec.*, 39(4):12-27, May 2011.

[3] H. Chen, Z. Wu, H. Wang, and Y. Mao, “Rdf/rdfs-Based Relational Database Integration,” *Proceedings of the 22nd International Conference on Data Engineering*, IEEE, pp. 94, 2006.

[4] V. Cuevas-Vicenttín, G. Vargas-Solar, and C. Collet, “Evaluating Hybrid Queries through Service Coordination in Hypatia,” *Proceedings of the 15th International Conference on Extending Database Technology*, AMC,

[6] http://code.google.com/p/model2roo/
[7] http://code.google.com/p/exschema/

pp. 602-605, 2012.

[5] O. M. Duschka and M. R. Genesereth, "Query Planning in Infomaster," *Proceedings of the Symposium on Applied Computing*, ACM pp. 109-111, 1997.

[6] M. Fowler and P. Sadalage, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, Addison-Wesley, 2012.

[7] Y. Halevy, "Answering Queries using Views: A Survey," *VLDB J.*, 10(4):270-294, 2001.

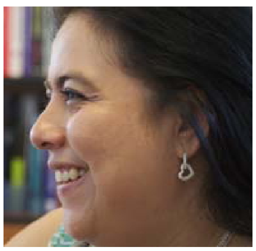
[8] R. Pottinger and A. Y. Halevy, "Minicon: A Scalable Algorithm for Answering Queries using Views," *VLDB J.*, 10(2-3):182-198, 2001.

[9] A. Ruiz-Alvarez and M. Humphrey, "An Automated Approach to Cloud Storage Service Selection," *Proceedings of the 2nd International Workshop on Scientific Cloud Computing*, Science Cloud '11, New York, NY, USA ACM, pp. 39-48, 2011.

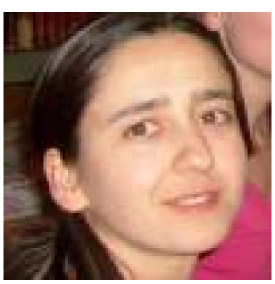
[10] A. Ruiz-Alvarez and M. Humphrey, "A Model and Decision Procedure for Data Storage in Cloud Computing," IEEE International Symposium on Cluster Computing and the Grid, pp. 572-579, 2012.

[11] G. Vargas-Solar, N. Ibrahim, C. Collet, M. Adiba, J.-M. Petit, and T. Delot, *Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications*, Chapter Querying Issues in Pervasive Environments, pp. 1-20, Nov. 2010.

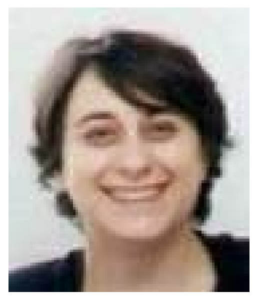
[12] C. Yu and L. Popa, "Constraint-Based XML Query Rewriting for Data Integration," SIGMOD Conference, pp. 371-382, 2004.



Genoveva Vargas Solar is a Senior Scientist of the French Council of Scientific Research (CNRS) and Deputy-Director the Franco-Mexican Laboratory of Informatics and Automatic Control (LAFMIA, UMI 3175). She is also member of the Informatics Laboratory of Grenoble (France) and invited research fellow of the Data and Knowledge Management Group at Universidad de lasAméricas Puebla. Her research contributes to the construction of service based database management systems. The objective is to design data management services guided by Service Level Agreements (SLA). She proposes methodologies, algorithms and tools for integrating, deploying and executing a service composition for programming data management functions. The results of her research are validated in the context of grids, embedded systems and clouds.



Catarina Ferreira da Silva is currently an Associate Professor at the University of Lyon in France. She has a Ph.D. in Computer Science from this same university (2007). Between 2009 and 2012 she worked as a researcher at the Information Systems research group of the Center for Informatics and Systems of the University of Coimbra, Portugal. Between2004 and 2007, she worked as a researcher at the Information Technology and Knowledge Dissemination Department (Division of Innovation and Services) of the Scientific and Technical Center for Building at Sophia Antipolis (France) and participated in several international and national research projects. She gave lectures at the Technical University of Nice Sophia-Antipolis (France) between 2004 and 2006. Her research interests are in the areas of Service Science, Cloud Computing Services and Semantic Web. She is currently working on the description and combination of complex cloud services.



Parisa Ghodous is currently a Professor at the University of Claude Bernard Lyon 1 in France. She has lead the collaborative information systems in LIRIS Laboratory. Her research interests are in the areas of cloud computing, interoperability, collaborative Information Systems, concurrent engineering, Ontologies and Web services. She has organized several conferences in the field of CE. She is a member of several experts groups (interop, Afnor, EPPPM, ISPE etc.). She is also on the board of several journals (CERA, ICAE, IJAM).

José Luis Zechinelli-Martini is an Associate Professor of the Department of Computing, Electronics and Mechatronics at the Universidad de lasAméricas Puebla (UDLA) since 2002 and he is currently Senior Scientist at LAFMIA. He is head of the Data and Knowledge Management Group (DBKM) of LAFMIA. Between 2003 and 2006 he was director of the CENTIA y coordinator of the PhD program on Computer Science. His fundamental research project addresses distributed databases on heterogeneous networks. The objective is to provide data access, querying and analysis adapted to the execution context by maximizing heterogeneous computing resources of continuous data production environments. He has developed his research through research projects financed by governmental agencies.

Data Warehouse Systems in the Cloud: Rise to the Benchmarking Challenge

Rim Moussa*
University of Tunis, Tunis, TUNISIA

Hassan Badir†
Abdelmalek Essaadi University, Tangier, MOROCCO

Abstract

The most common benchmarks for cloud computing are the Terasort benchmark and the YCSB benchmark. Although these benchmarks are quite useful, they were not designed for data warehouse systems and related OLAP technologies. The most prominent benchmarks for evaluating decision support systems are the various benchmarks issued by the Transaction Processing Council (TPC), namely TPC-H and its successor TPC-DS benchmarks. TPC benchmarks mismatch cloud rationale (scalability, elasticity, pay-per-use, fault-tolerance features) and Customer Relationship Management rationale (end-user satisfaction, Quality of Service features). In this paper, we present new requirements for implementing a benchmark for data warehouse systems in the cloud. The proposed requirements aim at allowing a fair comparison of different cloud systems providers' offerings

Key Words: Data warehouse, OLAP, cloud, TPC-H, TPC-DS, benchmark.

1 Introduction

Business Intelligence (BI) aims at supportting better decision-making, through building quantitative processes for a business to arrive at optimal decisions and to perform business knowledge discovery. Business intelligence often uses data provided by Data Warehouse Systems, in order to provide historical, current and predictive views of business operations. Nevertheless, data warehousing is very expensive, since it requires experts, advanced tools as well as costly hardware. Some organizations with limited means related to each of human, software and hardware resources for data analytics, are throwing terabytes of data away. Thus, the arrival of *pay-as-you-go Cloud Computing* presents new opportunities for decision support systems.

The cloud computing market is booming, and many research groups as Forrester [9] and Gartner [10], forecast a big invest in short-time on cloud technologies. Also, the Business Intelligence market continues growing and information analysts embrace OLAP concepts and related technologies (Microsoft Analysis Services, Oracle Business Intelligence, Pentaho BI suite, SAP NetWeaver, …). According to Gartner's latest enterprise software survey, the market for BI platforms will remain one of the fastest growing software markets in most regions (refer to [15] for details). However, there are hurdles around dealing with Big Data. Along Ralph Kimball, *Big data is a paradigm shift in how we think about data assets, where do we collect them, how do we analyze them, and how do we monetize the insights from the analysis*. Therefore, a major reason for the growth of big data is financial and Decision Support Systems have to deal with the *Big Data four V-dimensions*} namely (i) *Volume*-challenge of management of huge volumes of data, (ii) *Velocity*-challenge of how fast data is analyzed, (iii) *Variety*-challenge of dealing with unstructured, semi-structured, relational data, and finally (iv) *Veracity*-challenge of semantics and variability meaning in language.

Cloud computing has gained much popularity recently, and many companies now offer a variety of public cloud computing services, based on traditional relational DBMS, extended RDBMS and NoSQL technologies. Traditional software technologies tend to get quite expensive to manage, maintain and enhance. Two architectures have emerged to address big data analytics, which are extended RDBMS and NoSQL technologies (Apache Hadoop/MapReduce framework). Architectural developments for extended RDBMS are Massively Parallel Processing (MPP) and columnar storage systems. NoSQL has emerged as an increasingly important part of Big Data trends, and several NoSQL solutions are emerging with highly variable feature sets. Cloud services differ in service models and pricing schemes, making it challenging for customers to choose the best suited cloud provider for their applications. Data Warehouse Systems place new and different demands on cloud technologies, and vice-versa. In this paper, we propose new requirements for fair benchmarking of data warehouse systems in the cloud.

The outline of this paper is the following: first, in Section 2, we discuss related work in order to highlight our contribution. Then, we present preliminaries related to both cloud computing and data warehouse systems. In Section 3, we

* LaTICE Lab., Department of Computer Science. Email: rim.moussa@esti.rnu.tn.
† LabTIC Lab., Department of Computer Science. Email: hassan.badir@uae.ma.

recall the most important characteristics of cloud computing, and what a benchmark for data warehouse systems should feature; and in Section 4, we briefly overview data warehouse systems and well-known decision support systems benchmarks. We argue that TPC-H benchmark-the most prominent benchmark for decision support system, mismatches cloud rationale (scalability, elasticity, pay-per-use, fault-tolerance features) and Customer Relationship Management rationale (end-user satisfaction, Quality of Service features). In Section 5, we present new requirements for benchmarking data warehouse systems in the cloud. The proposed benchmark should allow a fair comparison of different cloud systems, as well as tuning of a cloud system for a given Cloud Service Provider (CSP) and selection of best optimizations and best cost-performance tradeoffs. Finally, we conclude the paper and present future work.

2 Related Work

In this section, we overview related work. The following research projects addressed specific issues when migrating data warehouse systems to the cloud,

- *Forrester* released a *Cost Analysis Tool*: *Cloud versus internal file storage Excel Workbook*, as a tool for comparison of storage on-premises and in the cloud [8],
- Nguyen et al. [20] propose cost models for Views Materialization in the cloud. Proposed cost models fit into the pay-as-you-go paradigm of cloud computing. These cost models help achieve a multi-criteria optimization of the view materialization under budget constraints.

There are few papers dealing with processing and evaluating by performance measurement OLAP workloads on cloud systems. Next, we overview research projects related to OLAP experiments in the cloud,

- Floratou et al. [7] conducted a series of experiments comparing cost of deployment in the cloud of different DBMSs, in order to make cloud customers aware of the high cost of using freeware software in the cloud. For instance, they ran Q21 of the Wisconsin Benchmark, and compared its response time using the open-source MySQL to the commercial MS SQL Server. For the SQL Server-based service, the user has to pay an hourly license cost, while he does not need to pay any license fee for MySQL usage. MS SQL server runs Q21 in 185sec, while MySQL runs the same query in 621sec. Obviously, the end-user bill will be affected by this 3.3X performance gap,
- In order to compare SQL technologies to NoSQL technologies, Pavlo et al. [22] compared the performance of Apache Hadoop/Hive to MS SQL Server database system using TPC-H benchmark,
- In [18], we proposed OLAP scenarios in the cloud. The proposed scenarios aim at allowing best performances, best availability and tradeoff between space, bandwidth and computing overheads. Evaluation is conducted using Apache Hadoop/Pig Latin with TPC-H benchmark, for various data volumes, workloads, and cluster sizes.

Many cloud computing benchmarks exist, but have different objectives than data warehouse systems. For instance,

- The *TeraSort* [12] benchmark measures the time to sort 1 TB (10 billion 100B records) of randomly generated data. It is used to benchmark NoSQL storage systems such as Hadoop and MapReduce performances.
- The *Yahoo Cloud Serving Benchmark -YCSB* [4] measures the scalability and performance of cloud storage systems such as HBase-the column-oriented database of Hadoop project, against a standard workload.
- The *CloudStone* Benchmark [24] is designed to support Web 2.0 type applications and measures the performance of social-computing applications on a cloud. For data analytics.
- The *MalStone* Benchmark [1] is specifically designed to measure the performance of cloud computing middleware that supports the type of data intensive computing common when building data mining models.

In [2], Binnig et al. presented initial ideas of requirements towards a web-shop benchmark (i.e., OLTP workload) in the cloud. They introduced new metrics for analyzing the scalability, the cost and the fault tolerance of cloud services. Later, in [14] they listed alternative architectures to effect cloud computing for web-shop database applications and reports on the results of a comprehensive evaluation of existing commercial cloud services. They used the database and workload of the TPC-W benchmark, with which they assessed Amazon, Google, and Microsoft's offerings.

The *CloudCMP* project [14] aims at comparing the performance and the cost of various cloud service providers. It models a cloud as a combination of four standard services, namely, (1) *Elastic Computer Cluster Service*: The cluster includes an elastic number of virtual instances for a workload processing; (2) *Persistent Storage Servic*}: The storage service stores application data. Different types of storage services may exist: *table* (SQL and NoSQL storage are considered), *blob* (binary files) and *queue messages* (as for Windows Azure); (3) *Intra-cloud Network Service*: The network inside a cloud that connects the virtual instances of an application (4) *WAN Service*: The wide-area delivery network of a cloud delivers an application's contents to the end hosts from multiple geographically distributed data centers of the cloud. The project scope is general, it does not address benchmarking data warehouses in the cloud specifities.

Most published research focused on benchmarking through exclusively performance measurements of high level languages and platforms of cloud systems, or investigation of a cost model for a particular topic in the cloud. In this paper, we show that TPC-H benchmark-the most prominent benchmark for decision support system, mismatches both (i) *cloud rationale* (scalability, elasticity, pay-per-use, fault-tolerance features) and (ii) *Customer Relationship Management* *rationale* (end-user satisfaction, Quality of Service features). Indeed, its metrics are not sufficient for assessing the novel cloud services. Moreover, we propose new metrics which fit to the characteristics of cloud computing and to characteristics of OLAP workloads. The proposed requirements and metrics are to make CSPs' offerings comparable from capabilities, and services perspectives.

3 Cloud Computing

The National Institute of Standards and Technology (NIST) [17] defines cloud computing as *a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*. Hereafter, we recall the five cloud characteristics, the three cloud service models, and we overview Cloud Service Providers (CSP) pricing models.

3.1 Cloud Characteristics

The cloud model is composed of three characteristics of virtualized systems, namely (1) *broad network access*-cloud computing is network based, and accessible from anywhere and from any standardized platform (i.e., desktop computers, mobile devices, …); (2) *resource pooling*-the multi-tenancy aspect of clouds requires multiple customers with disparate requirements to be served by a single hardware infrastructure, and therefore, virtualized resources (CPUs, memory, etc.) should be sized and resized with flexibility; (3) *rapid elasticity*-cloud computing gives the illusion of infinite computing resources available on demand. In particular, it is expected that the additional resources can be (a) provisioned, possibly automatically in mere minutes, when an application load increases (scale-up) and (b) released when load decreases (scale-down). In addition to the aforementioned characteristics, the cloud model is composed of two characteristics of on-demand computing services: (4) *on-demand self-service*-consumers of cloud computing services expect on-demand, nearly instant access to resources; (5) *measured service* (a.k.a. pay as you go) cloud services must be priced on a short term basis (e.g., by hour), allowing users to release resources as soon as they are not needed, and metering should be done accordingly for different types of service (e.g., storage, processing, and bandwidth).

3.2 Cloud Service Models

Based on user demand, cloud services include the delivery of software, infrastructure, and storage over the Internet, either as separate components or as a complete platform. Three primary cloud service models exist. The first being *Infrastructure as a Service* (IaaS) -An IaaS provider delivers computer hardware (servers, network, storage) as a service. It may also include the delivery of operating systems and virtualization technology to manage the resources. Examples of IaaS CSPs are: Amazon Elastic Computing Cloud (EC2), GoGRID. The second being *Platform as a Service* (PaaS) -a PaaS provider delivers infrastructure and an integrated set of software which provides everything a developer needs to build an application. Examples of PaaS CSPs are: Google AppEngine, Microsoft Azure Platform. The third being *Software as a Service* (SaaS) -a SaaS CSP access to software and its functions remotely as a Web-based service. Examples of SaaS providers for data analytics is: Google BigQuery, and for database as a service is: Amazon Relational Database Service.

4 Data Warehouse Systems

Business Intelligence aims at supportting better decision-making, through building quantitative processes for a business to arrive at optimal decisions and to perform business knowledge discovery. Business intelligence often uses data provided by Data Warehouse Systems. The concept of a *data warehouse* first appeared in articles published in the late 1988s by Bill Inmon. A *data warehouse* is defined as a *collection of subject-oriented, integrated, non-volatile, and time variant data to support management's decisions*. Data warehousing definition evolved to the process of collecting, cleansing, and integrating data from a variety of operational systems and making the resultant information available for the foundation of decision support and data analysis.

4.1 Typical DWS Architecture

Figure 1 illustrates a typical architecture of a data warehouse system. The latter is composed of three components: (1) Source integration system, (2) Data warehouse storage system and (3) Data analysis system. Next, we describe these components.

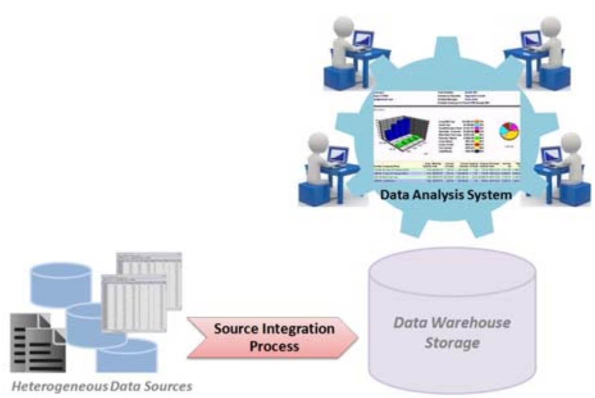


Figure 1: Typical data warehouse system architecture

4.1.1 Source Integration System. The source integration process deals first with acquiring data from a set of relevant data sources (e.g., legacy systems, relational databases, spreadsheets, …), then with integrating the schemas of the sources in order to obtain a global schema. For this purpose, it specifies the mapping between the global schema and the sources, and includes the specification of how to load and refresh data according to the global schema. Integration has to deal with the problem of cleaning and reconciling data coming from different sources, and consequently resolving naming, structural and data conflicts.

4.1.2 Data Warehouse Storage System. Two main approaches can be distinguished for storing data within a data warehouse, namely (i) *MOLAP*, where both the source data and the aggregation calculations are stored in a multidimensional data structures; and (ii) *ROLAP*, the data warehouse is physically stored using conventional Relational Database Management System and cubes are defined logically. There are also hybrid OLAP products (HOLAP), which allow both direct access to relational data for multidimensional processing, as well as having their own optimized multidimensional disk storage for aggregates and pre-calculated results. MOLAP is the fastest option for data retrieval, but it requires the most storage space and it is not very scalable.

4.1.3 Data Analysis System. The data analysis system embeds an OLAP server. The latter is a high-capacity, multi-user data manipulation engine specifically designed to process an OLAP workload. Multidimensional querying implemented by OLAP clients is an exploratory process, performed by navigating along the dimensions and measures, and allowing, (i) increase/decrease the level of detail (respectively drill-down and roll-up OLAP operations), (ii) focus on specific subparts of the cube for on-screen viewing (slice and dice OLAP operations), and (iii) rotation of dimensions to new on-screen viewing (rotate OLAP operation).

4.2 Common Optimization Strategies

Data warehouse solutions and appliances achieve better performances with the following technologies,

4.2.1 Hardware Technologies. Some data warehouse appliances provide special hardware products as storage solutions on-premises. Hardware solutions propose data storage devices allowing high I/O throughputs such as DRAM, Solid-State Drives (SSDs) and Parallel disks I/O. Notice that these hardware-based solutions are expensive and obsolete over time.

4.2.2 Columnar Storage Technology. A column-oriented storage system stores each record's column value (or familty of columns) in different data blocks. This technology allows higher compression ratio and higher scan throughput than ordinary row-based storage systems.

4.2.3 Derived Data. In order to get a fast response, data warehouses use derived data, such as OLAP indexes (e.g., bitmap, *n*-tree), derived attributes, aggregate tables (a.k.a. materialized views), and data synopsis. There are multiple techniques to perform approximate query processing using data synopsis. The most popular involve histograms, wavelets, sketches and sampling [5]. Nevertheless, derived data present disadvantages related to complexity of derived data calculus and refresh cost.

4.3 Decision Support Systems Benchmarks

There are few decision-support benchmarks out of the TPC benchmarks. Next, we overview most known benchmarks in the community.

4.3.1 APB-1 Benchmark. APB-1 [21] has been released in 1998 by the OLAP council, a now inactive organization. APB-1 warehouse dimensional schema is structured around five fixed size dimensions and its workload is composed of 10 queries. APB-1 is proved limited [8] to evaluate the specificities of various activities. It proposes a single performance metric termed AQM (Analytical Queries per Minute). The metric AQM denotes the number of analytical queries processed per minute including data loading and computation time.

4.3.2 TPC-H Benchmark. The most prominent benchmarks for evaluating decision support systems are the various benchmarks issued by the Transaction Processing Council (TPC). Since two decades, TPC-H benchmark is the most used benchmark in the research community. The TPC-H benchmark exploits a classical product-order-supplier model. It consists of a suite of business oriented adhoc queries and concurrent data modifications. The workload is composed of 22 parameterized decision-support SQL queries with a high degree of complexity and two refresh functions: RF-1 *new sales* (new inserts) and RF-2 *old sales* (deletes). Scale factors used for the test database are: 1, 10, …, 100,000; and resulting raw data volumes are respectively 1GB, 10GB, …, 100TB.

TPC-H benchmark, and its successor TPC-DS, report two main metrics (see details in Appendix A)

- *TPC-H Composite Query-per-Hour Performance Metric* (*Qph@Size*): The Qph@Size metric reflects multiple aspects of the capability of the system under test for query processing. These aspects include (i) the selected database size against which the queries are executed (i.e., *scale factor*), (ii) *power test* which is the query processing power when queries are submitted by a single stream, and (iii) the throughput test, which is the query throughput when queries are submitted by multiple concurrent users.
- *TPC-H Price-Performance Metric* ($/Qph): The $/Qph metric reflects the ratio of costs to performance. The calculation of the priced system consists of (i) the price of both hardware and software present in the system under test, (ii) the price of the communication interface supporting the required number of user interface devices, (iii) the price of on-line storage for the database and storage for all software, (iv) the price of additional products (software or hardware) required for customary operation, administration and maintenance for a period of three years, and finally (v) the price of all products

required to create, execute, administer, and maintain the executable query texts or necessary to create and populate the test database.

4.3.3 Mismatching of TPC-H Benchmark for Evaluation of DWS in the Cloud. The use of TPC-H for benchmarking Data Warehouse Systems in the cloud reveals the following problems,

First, considering the technical evolution of OLAP technologies in the last years, the TPC-H benchmark does not reflect modern implementations of data warehouse systems, and is not suitable for the benchmarking of commercial business intelligence suites, i.e., integration services (ETL performances), OLAP engines (OLAP hypercubes building) and reporting tools. Most business intelligence projects query the data warehouse system using Multi-Dimensional eXpressions language (MDX) [18], while the TPC-H and TPC-DS benchmarks feature an SQL workload.

Second, the primary metric used by TPC-H -*Qph@Size*, is the number of queries processed per hour, that the system under test can handle for a fixed load. The system under test is then considered static, and this metric does not show the system scalability, i.e., system performance under variable loads and for variable cluster size.

Third, the second metric used by TPC-H -*$/Qph*, is the ratio of costs to performance, such that the pricing is based on the total cost of ownership of the system under test on-premises. The ownership cost includes hardware pricing, software license costs, as well as administration and maintenance costs during three years. This is incompatible with the pay-as-you-go model of cloud computing, since the cloud customers are not directly exposed to the hardware, software maintenance, and administration costs of their deployment. For the cloud, different price-plans exist and the cost-performance ratio depends on data volume, workload, services, selected hardware, and consequently on the CSP pricing plan. Also, the demand for required hardware and software resources shall vary over time, and then is better formulated by the dynamic lot-size model.

Fourth, currently none of the TPC-benchmarks reports a *cost-effictiveness ratio metric*. Migration to the cloud should help the company determine the best hardware configuration for managing efficiently its data and running efficiently its workload. Indeed, it does not make sense to afford an Amazon EC2 Extra Large Instance (15GB of memory and 8 EC2 compute units for $0.480 per Hour), when an Amazon EC2 Large Instance (7.5GB of memory and 4 EC2 compute units for \$0.240 per Hour) satisfies the workload requirements. A second motivating example for cost-effectiveness ratio is the following: Oracle publishes a detailed DBaaS service catalog for DBaaS [1], where the main variables are: (i) DB service name-defined as combination of load estimate complexity and workload type, particularly {small, medium or large} and {OLTP or OLAP}, (ii) CPU Size -2,4,8 or 16 cores, (iii) Server Memory -6, 8, 16, 24 or 48GB, (iv) Storage Redundancy (2-way or 3-way), (v) Service Availability (Node, Server or Site). Hence, a given company may choose a not cost-effective DBaaS service. The cost-effectiveness ratio should help a company defining its needs.

Fifth, the CAP theorem [3], also known as Brewer's theorem, asserts that any networked shared-data system can have only two of three following properties, namely, (i) *Consistency* which guarantees that all nodes see the same data at the same time; (ii) *Availability* which guarantees that every request receives a response about whether it was successful or failed; and (iii) *Partition tolerance* which guarantees that the system continues to operate despite arbitrary message loss or failure of part of the system. Benchmarking data warehouse systems in the cloud on a networked data system should implement all different combinations of guarantees, namely CA, CP and AP when considering refresh functions and high-availability.

Finally, the TPC-H benchmark lacks adequate metrics for measuring the features of cloud systems like scalability, pay-per-use and fault-tolerance, and service level agreements. In the next section, we present requirements and new metrics for benchmarking data warehouse systems in the cloud.

5 Benchmarking Data Warehouse Systems in the Cloud

The data warehousing process is inherently complex and, as a result, is costly and time-consuming. The deployment of a data warehouse system in the cloud is very different than its deployment on-premises. Indeed, the relationship between the CSP and its customers is different than the relationship between a company and its BI department. Migration to the cloud should improve end-user satisfaction and induce greater business productivity. Thus, benchmarks designed for evaluation of data warehouse systems in the cloud should reflect end-user satisfaction, Quality of Service (QoS), as well as all inherent characteristics of cloud systems, namely high performance, elasticity, scalability, pay-per-use and fault-tolerance. Next, we first present use cases of benchmarking data warehouse systems in the cloud, then we present new requirements and corresponding metrics which aim at a fair comparison of different cloud systems providers of data warehouse systems.

5.1 Use Cases

Two main use cases are identified of benchmarking data warehouse systems in the cloud. First, the *comparison of different cloud systems*, which aims to select the best CSP for final deployment of a data warehouse system. Second, the *tuning of a system*: which aims to select, for a given CSP, the capacity planning (operating system, number of instances, instance hardware configuration, …), best optimizations, best cost-performance tradeoffs, best cost-effectiveness tradeoffs.

5.2 Proposed Requirements

Next, we detail new requirements and corresponding metrics for benchmarking data warehouse systems in the cloud.

5.2.1 High Performance Metering. Data warehousing is intended for decision support. The latter requires high performance for greater business productivity. Two main features of data warehousing in the cloud affect high performance, which are (i) data transfer to/from the CSP and (ii) workload processing.

- **Data Transfer to and from the Cloud Service Provider**: the source integration system and the data analysis system manipulate huge data sets. Practically, big data uploads on remote servers require a lot of bandwidth and perform better on local networks. The operational system of the company may be serviced at a different cloud service provider, at the same cloud service provider (CSP) selected for the data warehouse or on-premises. So, unless creation of an expensive private link between the operational system location and the data warehouse provider location, data warehousing in the cloud is constrained by low-speed connections and network congestion issues. The worst case presents an operational DB serviced at a different cloud service provider. Indeed, in this case, data transfer from a CSP to a different CSP should be considered. As usually data download from any CSP is charged, the cost of data migration from a CSP to a different CSP will be very expensive. If on-premises, companies are confronted to I/O-bound and CPU-bound applications, in the cloud they will confront to *network-bound applications*. Indeed, the bottleneck will be the network bandwidth available to perform huge data transfer to/from the CSP. Most CSPs provide data transfer to their data centers at no cost (e.g., Data Transfer IN To Amazon EC2 From Internet costs $0.00 per GB). Nevertheless, data download is priced (e.g., Data Transfer OUT To Amazon EC2 From Internet $0.12 per GB per month for data volumes comprised between 1GB and 10TB, and it costs cheaper for higher data volumes, and it is free for lower data volumes).
- **Workload Performance**: most OLAP engines implement *intra-query parallelism* to provide faster performance. *Intra-query parallelism* consists in breaking a complex single query into sub-queries, processing the workload over multiple processors, and finally performing post-processing for presenting the final query response. Three factors impact the final response time to a query, which processing implies intra-query parallelism. First, *Start-up costs*, which are related to starting up multiple processes for processing simultaneously sub-queries. The time to set-up these processes may dominate computation time if the degree of parallelism is high. Second, *Skew costs*, these costs show that in a distributed system the overall execution time is determined by the slowest of parallely executing tasks. Third, *Interference costs*, these costs relate to the time the processes are idle. Indeed, processes accessing shared resources (e.g., system bus, disks, or locks) compete with each other and spend time waiting on other processes. Most systems, relational DBMS or NoSQL technologies [18] feature a concave curve, with an optimum response time for a particular cluster size and where performance degrades from this optimum onward (see Figure 2). For cloud computing, the slope, showing performance gain (from N to N') should be also expressed in a cost metric. Indeed, to obtain an improvement in response time, the system scales-out horizontally, and more instances are provisioned.

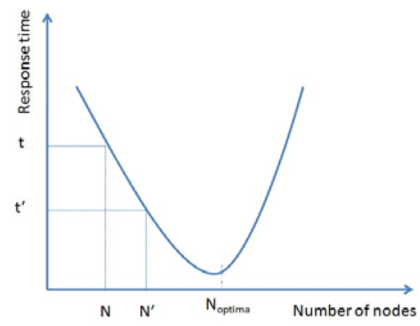


Figure 2: Response times of OLAP queries across cluster size

5.2.2 Cost Metering. Even though many services look similar from the outside, the services vary when it comes to system architectures, performance, scalability, and cost. Cloud Service Providers have different pricing models for storage, CPU, bandwidth and services. Next, we present the different charging plans adopted by cloud service providers,

- Compute Cost: There are two types of providers' charging for CPU cost,

  - Instance-based: the CSP charges the customer for the number of allocated instances and how long each instance is used. This is regardless of whether the instances are fully utilized or under utilized. Examples of CSPs which fall in this CPU pricing model are Amazon AWS and Windows Azure.
  - CPU cycles-based: the CSP charges the customer for the number of CPU cycles a customer's application consumes. Examples of CSPs which fall in this CPU pricing model are CloudSites and Google AppEngine.

- Storage Cost: Data Warehouse Systems are IO intensive applications. Thus, storage performance, throughput and bandwidth capacity planning become critical for data warehousing in the cloud. Storage devices have two limits (i) the amount of storage available and (ii) the amount of sustainable IOPs (Input/Output Operations per Second). Most CSPs implement a bundling-pricing for storage space charging (first 1 TB cost/ month, next *N* TB cost/ month and so on). Nevertheless, the real measure of storage performance is IOPS. Flash based storage whether it be DRAM or Solid State Drives (SSDs) maximize IOPS, but are expensive. Some CSPs charge for IO request. For instance, MS Azure charges $.01 per 10,000 IO requests, while Amazon S3 charges more per write operation: $.01 per 1,000 *put, copy, post, or list* *requests* and $.01 per 10,000 *get requests*.

- Software cost: the CSP may provide some software at no cost. Notice that most operating systems are charged to customers with the cost of instance. Applications are either charged on a *pay-as-you-go basis* or on *subscription basis*. For *pay-as-you-go*, the cost is aligned to usage.
- Intra-network cost: most providers offer intra-cloud network bandwidth consumption at no cost. Basically, no information is available about interconnectivity of nodes within a data center. Notice that, intra-network bandwidth is very important for distributed processing of OLAP workloads, for both SQL and NoSQL solutions.
- WAN cost: Charges for using the wide-area delivery network are based on the amount of data delivered through the cloud boundaries to the end-users. Currently, most providers have similar prices for this service, where data upload is free of charge and data download is priced.
- Services' cost: SaaS offers for analytics are different than IaaS and PaaS offers. Indeed, the cost of the service is included in the price model. For instance, BigQuery [24] pricing for storage resources depends on data volume, and the pricing of workload processing depends on the number of bytes retrieved for each business question. BigQuery is a columnar-storage system, which adopts an I/O-based pricing model. Other cloud service providers propose a subscription-based pricing model, for instance the Clustrix/GoGrid DBaaS cloud solution is available on a monthly subscription basis.

5.2.3 Scalability Metering. Scalability is the ability of a system to increase total throughput under an increased load when hardware resources are added. Ideally, cloud services should scale linearly with a fixed cost per processed business question. Current TPC-H implementation measures the capacity of a system for a static workload. We propose that the benchmark for data warehousing should assess the system under test with an ever increasing load, and measures the throughput consequently. Scalability can be measured with *speed-up metric* and *scale-up metric*. *Speed-up metric* refers to the workload processing time gained as a consequence of adding new nodes and keeping the workload constant, and *Scale-up metric* refers to the throughput processing capacity gained by adding new nodes and increasing the workload. In order to quantify this requirement, we can vary the workload on a time scale basis, every 1hour for instance, and measure the number of business questions processed during the time interval across a variable cluster size.

5.2.4 Elasticity Metering. Elasticity adjusts the system capacity at runtime by adding and removing resources without service interruption in order to handle the workload variation. First, the metric should assess the system capacity to autoprovision and release resources without service interruption, and in case it does, it reports first *scaling latency*, i.e., the time required for a system to scale-down or to scale-up horizontally, and second the *scale-up cost*, i.e., the cost of newly acquired resources or the *scale-down gain*, i.e., the cost of newly released resources. Finally, it reports the impact of the scale-up or scale-down operation on system performances.

5.2.5 High Availability Metering. Data distribution among multiple disks increases the distributed storage system failure likelihood. Many approaches to build highly available distributed data storage systems have been proposed. They generally use either (i) replication or (ii) parity calculus. The latter approach uses systematic erasure-codes (e.g., Reed Solomon (RS) codes, Low-Density Parity-Check (LDPC) codes, Tornado code). With replication, data management is straightforward. However, the storage overhead with replication is always higher than it is with systematic erasure codes. When a certain level of availability is targeted the erasure codes are able to provide service with a lower storage overhead than replication techniques. For data warehousing, high availability through erasure codes saves storage costs, particularly for big data of type *write-once* (i.e., not subject to delete refreshes). Nevertheless, data recovery is more complicated than replication. Indeed, *first* data recovery is not a simple *copy to operation* as for replication, it performs complex decoding calculus, and *second* data recovery involves different servers, which send their contents to a recovery manager and consequently it implies a high communication overhead. Erasure codes were investigated and proved efficient for highly available distributed storage systems [16] and grid systems [23]. Figure 3 illustrates the storage space requirements in different file high-availability schemes, namely replication and erasure codes. In our example, we show 4 blocks of a data file (m = 4) stored in such a way that any (n - m) = 2 missing blocks can be tolerated; values n = 6 and m = 4 are used as an example. With replication, k copies of the entire file are stored into separate places. The group of data blocks is 2-available through replication with a redundancy overhead of 200 percent versus the same group of data blocks 2-available through erasure-codes with a redundancy overhead of 50 percent.

Some CSPs implement replication for increasing the availability of stored data and preventing discontinuity of service. They also offer replicas management in data centers situated in different geographic locations. This allows disaster

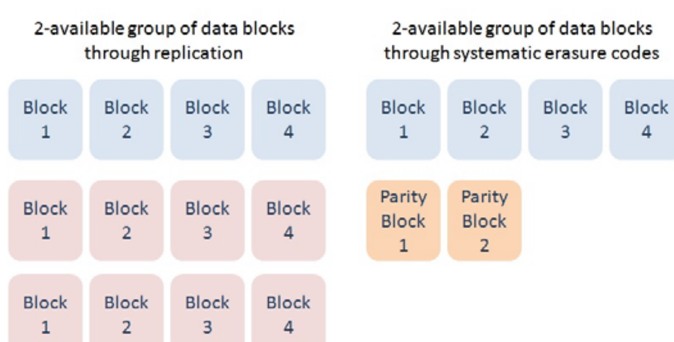


Figure 3: Replication vs. erasure codes for a group of 4 data blocks

recovery from a failure node within the data center as well as whole data center outage. Nevertheless, most CSPs do not customize high availability services to their customers. For data warehousing in the cloud, the end-user should be notified of the cost of rendering its data highly-available through different high availability strategies (i.e., for both synchronous and asynchronous refreshes), and different levels of availability should be offered which enables customization of the recovery capacity following disasters. Consequently, the benchmark should embed metrics measuring the cost of different targeted levels of availabilities (1-available, …, *k*-available, i.e., the number of failures the system can tolerate), as well as the recovery cost. We propose two metrics which denote the cost of maintaining of a *k*-available system $@*k*, with *k* is the targeted level of availability, and a metric denoting the cost of recovery expressed in time and decreased system productivity caused by the hardware failure from customer perspective. The latter should be charged to the CSP.

5.2.6 Cost-Effectiveness and Cost-Performance Metering. The cloud-based solutions should help companies, which look to optimize costs without compromising on efficiency and quality of service. Therefore, there is an emerging need to understand, manage and proactively control costs across the cloud from two perspectives, namely *performance perspective* and *effectiveness perspective*. Indeed, instead of searching for the minimal execution time, the user may want to run his application more cost effectively, which ensures a maximal computation at minimal costs. The cost management plan should include determination of the best hardware configuration versus performance and versus effectiveness; this assumes a systematic monitoring of resource utilization. For these purposes, we propose measuring the ratio of configuration cost to performance and to resource utilization. Resource utilization is the ratio of used resources to allocated resources. Notice that used resources and allocated resources vary over time.

5.2.7 Service Level Agreements and QoS Metering. A *Service Level Agreement* (SLA) is a contract between a service provider and its customers. SLAs capture the agreed upon guarantees between a service provider and its customer. They define the characteristics of the provided service including service level objectives, as maximum response times, minimum throughput rates and data consistency, and define penalties if these objectives are not met by the service provider. Penalty is an amount that the provider must pay to the customers if the SLA is not met. For example, in Google AppEngine, Microsoft Azure, or Amazon S3, if availability is lower than 99.9 percent, then the customers receive a service credit, according to SLA, and proportional to the revenue.

Sousa et al. [26] proposes QoSDBC framework, an approach to QoS for databases in the cloud. The SLAs categories for the data warehousing in the cloud are scalability, elasticity, performance (throughput and response time are both considered), high-availability and independency of the CSP. For the latter, the company should be able to easily migrate to another Cloud Service Provider (CSP), and get its data back in a standard format. This will limit losses in case the CSP requires the purchase of new software, imposes exorbitant prices, or goes bankrupt.

5.3 Summary of Proposed Metrics

In Table 1, we propose a summary of metrics for data warehouse systems' benchmarking in the cloud.

Table 1: Summary of metrics for data warehouse systems' benchmarking in the cloud

Requirement	Proposed Metrics
High Performance	*Data Transfer IN/OUT the CSP*, • time and cost for data upload IN the CSP, • time and cost for data download OUT the CSP *Workload Processing*, • Workload processing time
Cost	• depends on cloud service provider pricing scheme
Scalability	• *scale-up*: workload processing performances under an ever increasing workload across variable cluster size • *speed-up*: workload processing performances under a constant workload across variable cluster size
Elasticity	• capacity of scale-up/ scale-down, • scaling latency • scale-up/ scale-down impact on system under test performances • scale-up cost(+$) or scale-down gain (-$),
High-availability	• cost of a targeted k level-of-availability, • mean time to recovery • decreased productivity due to discontinouity of service
Cost-Performance	• ratio of cost to performance,
Cost-Effectiveness	• ratio of cost to aggregated resources' usage percent,
Service Level Agreements	• QoS assessment through tracking of unsatisfied service level agreements,

6 Conclusion

The rationale of migration of data warehouse systems to the cloud, are basically thrice, (i) reduction of capital expenditure through measured service, with infrastructure, platform, services are provided on a pay-per-use basis (ii) rapid elasticity for adaptive resource capacity to workload, and (iii) better cost-performance tradeoff. In this paper, we propose new requirements and metrics to be fulfilled by a benchmark for data warehouses in the cloud, such as high-performance, high-availability, cost-effectiveness, cost-performance, scalability, elasticity, as well as SLAs. In future work, we foresee to assess and compare most known CSPs for data warehousing in the cloud.

References

[1] Collin Bennett, Robert L. Grossman, David Locke, Jonathan Seidman, and SteveVejcik, “Malstone: Towards a Benchmark for Analytics on Large Data Clouds,” *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’10*, ACM, pp. 145-152, 2010.

[2] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing, “How is the Weather Tomorrow?: Towards a Benchmark for the Cloud,” *Proceedings of the Second International Workshop on Testing Database Systems, DBTest ’09*, ACM, pp. 91-96, 2009.

[3] Eric Brewer, “Pushing the CAP: Strategies for Consistency and Availability,” *Computer*, 45(2):23-29, February 2012

[4] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears, “Benchmarking Cloud Serving Systems with YCSB,” *Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC ’10*, pp. 143-154, 2010.

[5] Graham Cormode, Minos Garofalakis, Peter J. Haas, and Chris Jermaine, “Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches,” *Found, Trends Databases*, 4:1-294, 2012.

[6] Database as a Service: Reference Architecture, http://www.oracle.com/technetwork/topics/entarch/oes-refarch-dbaas-508111.pdf, 2011.

[7] Avrilia Floratou, Jignesh M. Patel, Willis Lang, and Alan Halverson, “When Free is not really Free: What does it Cost to Run a Database Workload in the Cloud?” 4th TPC Technology Conference on Performance Evaluation and Benchmarking (TPCTC), pp. 163-179, 2011.

[8] Forrester, “File Storage Costs Less in the Cloud than Inhouse,” http://www.forrester.com, 2011.

[9] Forrester, Sizing the Cloud, http://www.forrester.com, 2011.

[10] Gartner Group, Forecast: Public Cloud Services, Worldwide, 2011-2017, 1q13 Update, http://www.gartner.com/id=2391015, 2013.

[11] Jim Gray, Sort Benchmark Home Page. http://research.micro soft.com/barc/SortBenchmark/, 2008.

[12] Donald Kossmann, Tim Kraska, and Simon Loesing, “An Evaluation of Alternative Architectures for Transaction Processing in the Cloud,” *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, SIGMOD ’10*, ACM, pp. 579-590, 2010.

[13] Nicole Laskowski, “Business Intelligence Software Market Continues to Grow,” http://www.gartner.com/.

[14] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang, “Cloudcmp: Shopping for a Cloud made Easy,” *USENIX HotCloud*, pp. 5-5, 2010.

[15] Witold Litwin, Rim Moussa, and Thomas J. E. Schwarz, “With LH^*_{RS} - a Highly-Available Scalable Distributed Data Structure,” *ACM Trans. Database Syst.*, 30(3):769-811, 2005.

[16] Peter Mell and Timothy Grance, “The NIST Definition of Cloud Computing,” National Institute of Standards and Technology, csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, 2011.

[17] Rim Moussa, “Massive Data Analytics in the Cloud: Tpch Experience on Hadoop Clusters,” *Intl. Journal of Web Applications (IJWA)*, 4:113-133, 2012.

[18] Multi-Dimensional Expressions Language, msdn.micro soft.com/enus/library/aa216779(SQL.80).aspx.

[19] Thi-Van-Anh Nguyen, Sandro Bimonte, Laurent d’Orazio, and Jérôme Darmont, “Cost Models for View Materialization in the Cloud,” EDBT/ICDT Workshops, pp. 47-54, 2012.

[20] OLAP Council: APB-1, www.olapcouncil.org.

[21] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker, “A Comparison of Approaches to Large-Scale Data Analysis,” SIGMOD Conference, pp. 165-178, 2009.

[22] Mikko Pitkänen, Rim Moussa, D. Martin Swany, and Tapio Niemi, “Erasure Codes for Increasing the Availability of Grid Data Storage,” AICT/ICIW, pp.185-197, 2006.

[23] Kazunori Sato, “An Inside Look at Google Bigquery,” https://cloud.google.com/files/.

[24] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, A. Klepchukov, S. Patil, A. Fox, and D.Patterson, “Cloudstone: Multi-Platform, Multi-Language Benchmark and Measurement Tools for Web 2.0,” *Proceedings of Cloud Computing and Its Applications*, http://www.cca08.org/papers/Paper33-Armando-Fox.pdf, 2008.

[25] Flávio R. C. Sousa, Leonardo O. Moreira, Gustavo A. C. Santos, and Javam C. Machado, “Quality of Service for Database in the Cloud,” *CLOSER*, pp. 595-601, 2012.

[26] Erik Thomsen, “Comparing Different Approaches to Olap Calculations as Revealed in Benchmarks,” Intelligence Enterprises Database Programming & Design, 1998.

Appendix A: TPC-H Benchmark Metrics

- *Qph@SF Metric*

$$power\_test@SF = \frac{3600 \times SF}{\sqrt[24]{\prod_{i=1}^{i=22} QI(i,0) \times \prod_{j=1}^{j=2} RI(j,0)}}$$

$$throughput\_test@SF = \frac{S \times 22 \times 3600}{T \times SF}$$

$$with \begin{cases} 3600: & \text{is 1 hour duration in seconds} \\ SF: & \text{is the scale factor.} \\ QI(i,0): & \text{is the timing interval, in seconds, of query} \\ & Q_i \text{ within the single query stream of the} \\ & \text{power test,} \\ RI(j,0): & \text{is the timing interval, in seconds, of refresh} \\ & \text{finction } RF_j \text{ within the single query steam} \\ & \text{of the power test,} \\ S: & \text{is the number of query steams, such that a} \\ & \text{stream is composed of different 22 queries} \\ T: & \text{is the measurement interval,} \end{cases}$$

$$Qph@SF = \sqrt{power\_test@SF \times throughput\_test@SF}$$

$\$/Qph$ *Metric* $\$/Qph = \frac{\Pr iced\ System}{Qph@Size}$

Rim Moussa is an Associate Professor in the Department of Computer Science at the University of Carthage (Tunisia), where she is in charge of graduate lectures, related to distributed systems, information retrieval and business intelligence fundamentals and practices. She received her Ph.D. in Computer Science from Paris IX Dauphine University in 2004. Her research interests include data management systems and distributed systems.

Hassan Badir is an Associate Professor in the Department of Computer Science and Engineering and a member of LabTIC Laboratory at Abldemalek Essaadi National School of Applied Sciences at Tangier (ENSA-Tangier). He received a Ph.D. in Computer Science from the INSA-Lyon and Claude Bernard University (France) in 2005. His research interests are in the areas of big data management, sensor and scientific data management, cloud computing and security.

Index

Authors

A

B-C

D-F

G-J

K

L

M-P

Q-R

S-V

W

X-Z

Key Words

Instructions For Authors

The International Journal of Computers and Their Applications is published multiple times a year with the purpose of providing a forum for state-of-the-art developments and research in the theory and design of computers, as well as current innovative activities in the applications of computers. In contrast to other journals, this journal focuses on emerging computer technologies with emphasis on the applicability to real world problems. Current areas of particular interest include, but are not limited to: architecture, networks, intelligent systems, parallel and distributed computing, software and information engineering, and computer applications (e.g., engineering, medicine, business, education, etc.). All papers are subject to peer review before selection.

A. Procedure for Submission of a Technical Paper for Consideration

1. Email your manuscript to the Editor-in-Chief, Dr. Frederick C. Harris, Jr., Fred.Harris@cse.unr.edu.

2. Illustrations should be high quality (originals unnecessary).

3. Enclose a separate page (or include in the email message) the preferred author and address for correspondence. Also, please include email, telephone, and fax information should further contact be needed.

B. Manuscript Style:

1. The text should be **double-spaced** (12 point or larger), **single column** and **single-sided** on 8.5 X 11 inch pages.
2. An informative abstract of 100-250 words should be provided.
3. At least 5 keywords following the abstract describing the paper topics.
4. References (alphabetized by first author) should appear at the end of the paper, as follows: author(s), first initials followed by last name, title in quotation marks, periodical, volume, inclusive page numbers, month and year.
5. Figures should be captioned and referenced.

C. Submission of Accepted Manuscripts

1. The final complete paper (with abstract, figures, tables, and keywords) satisfying Section B above in **MS Word format** should be submitted to the Editor-in-Chief.
2. The submission may be on a CD/DVD or as an email attachment(s) . **The following electronic files should be included:**

   - Paper text (required).
   - Bios (required for each author).
   - Author Photos (jpeg files are required by the printer).
   - Figures, Tables, Illustrations. These may be integrated into the paper text file or provided separately (jpeg, MS Word, PowerPoint, eps).

3. Specify on the CD/DVD label or in the email the word processor and version used, along with the title of the paper.

4. Authors are asked to sign an ISCA copyright form (http://www.isca-hq.org/j-copyright.htm), indicating that they are transferring the copyright to ISCA or declaring the work to be government-sponsored work in the public domain. Also, letters of permission for inclusion of non-original materials are required.

Publication Charges

After a manuscript has been accepted for publication, the contact author will be invoiced for publication charges of **$35.00 USD** per page (in the final IJCA two-column format) to cover part of the cost of publication. For ISCA members, publication charges will be waived if requested; for non-members, publication charges are required.

May 2012