SHYAM: a system for autonomic management of virtual clusters in hybrid clouds

Daniela Loreti and Anna Ciampolini

DISI - Department of Computer Science and Engineering
Università di Bologna,
Viale del Risorgimento 2 Bologna, Italy,
{daniela.loreti, anna.ciampolini}@unibo.it

Abstract. While the public cloud model has been vastly explored over the last few years to face the demand for large-scale distributed computing capabilities, many organizations are now focusing on the hybrid cloud model, where the classic scenario is enriched with a private (company owned) cloud – e.g., for the management of sensible data. In this work, we propose SHYAM, a software layer for the autonomic deployment and configuration of virtual clusters on a hybrid cloud. This system can be used to face the temporary (or permanent) lack of computational resources on the private cloud, allowing cloud bursting in the context of big data applications. We firstly provide an empirical evaluation of the overhead introduced by SHYAM provisioning mechanism. Then we show that, although the execution time is significantly influenced by the inter-cloud bandwidth, an autonomic off-premise provisioning mechanism can significantly improve the application performance.

Keywords: Autonomic, Hybrid Cloud, Big Data, MapReduce

1 Introduction

Offering "the illusion of infinite computing resources available on demand" [5], cloud computing is the ideal enabler for high computing power demanding applications. While the public cloud scenario had been well explored in the past, many organization are now focusing on the hybrid cloud model. Combining both on-premise (company owned) and off-premise (owned by a third party provider) cloud infrastructures, the hybrid scenario can indeed capture a broader use-case [19]. Recently, the exponential increase in the use of mobile devices and the wide-spread employment of sensors across various domains has created large volumes of data that need to be processed to extract knowledge. The pressing need for fast analysis of large amount of data calls the attention of the research community and fosters new challenges in the big data research area [10]. Since data-intensive applications are usually costly in terms of CPU and memory utilization, a lot of work has been done to simplify the distribution of computational load among several physical or virtual nodes and take advantage of parallelism [12]. Nevertheless, the execution of data-intensive applications requires a high degree of elasticity in resource provisioning. In this scenario, a widespread choice

is to relay on a cloud infrastructure to take advantage of its elasticity in virtual resource provisioning.

In this paper, we focus on the autonomic management of virtual machines (VMs) in the context of hybrid clouds. To this purpose, we present SHYAM (System for HYbrid clusters with Autonomic Management), a system for the autonomic management of VMs in hybrid clouds able to manage virtual clusters using both on-premise (i.e., computing nodes in a private internal cloud IC) and off-premise (i.e. in a public external cloud EC) hardware resources. The system is able to dynamically react to load peaks – due, for instance, to virtual machine (VM) contention on shared computing nodes – by redistributing the VMs on less loaded nodes (either migrating inside IC or crossing the cloud boundaries towards EC). As a case study, we consider the execution of data-intensive applications over clusters of VMs initially deployed on IC. If a physical node hosting a VM for data-processing becomes overloaded in terms of CPU, memory or disk utilization, the performance of the virtual cluster may dramatically decrease, thus slowing down the whole distributed application. In this case, if another less loaded physical machine is available on-premise, the best solution would be to migrate the VM on that physical node. However, the private cloud has a finite amount of resources and it may happen that all the physical machines in IC are too loaded to receive the VM: in this case, we can provide resources on EC and perform application-level load redistribution; in SHYAM we automated this mechanism. As this work tests the SHYAM system on data-intensive applications, it also explores the drawbacks and shortcomings of the hybrid scenario, primarily due to data movement crossing on-/off-premise boundaries. Although data-processing is significantly influenced by the limited inter-cloud bandwidth, our work shows that an autonomic off-premise provisioning mechanism could allow the user to significantly increase the application performance.

The paper is organized as follows. Section 2 presents the architecture of the proposed autonomic system, illustrating the data-processing scenario and management policy adopted, as well as practical details about the implementation. Section 3 discusses the experimental results obtained by testing our solution in the chosen data-intensive scenario. Related work and conclusion follow.

2 Framework Architecture

We focus on a hybrid scenario composed of two separated cloud installations: the on-premise IC, owned and managed by a private company, and the off-premise EC, a collection of resources owned by a cloud provider and rented to customers according to a predefined price plan. Having their own cloud management software and offering their virtualized resources to final users (e.g., customers, company employees, etc...), both IC and EC implement the cloud paradigm at the Infrastructure as a Service (IaaS) level.

As shown in Fig.1a, the key component of SHYAM is Hybrid Infrastructure as a Service (HyIaaS), a software layer that allows integration between IC and EC infrastructures. The layer interacts with both on- and off-premise cloud with the goal of providing hybrid clusters of VMs. Each cluster is dedicated to the execution of a particular distributed application (e.g., distributed data-processing).

If there are enough resources available, all the VMs of a cluster are allocated on-premise to minimize the costs introduced by the public cloud and the latency of data transferred between the virtual nodes. If on-premise resources are not sufficient to host all the VMs, a part is provisioned on IC and the others on EC. This partitioning should be transparent to the final user of the virtual cluster, allowing her to access all the VMs in the same way, regardless to the physical allocation. We call *hybrid cluster* the result of this operation.

HyIaaS is also responsible for autonomously handling to changes in the current utilization level of the on-premise physical machines hosting the VMs of the cluster. To avoid the application slowdown due to the poor performance of these VMs, HyIaaS layer is in charge of dynamically spawning new VMs on EC and providing them to the above Application layer (Fig.1a). This layer is responsible for installing and configuring a specific distributed application on the newly provided VMs. SHYAM's main goal is to unify on- and off-premise resources while keeping a strong separation between the infrastructure and application levels. It must be installed on IC, so that it can collect monitoring information about the utilization level of the on-premise machines. According to a specific user-defined policy, the HyIaaS layer can perform cloud bursting toward EC by translating generic spawning and scale-down requests into specific off-premise provisioning and de-provisioning commands. If both IC and EC have a centralized architecture, SHYAM makes them able to cooperate by communicating with their central controllers. In the following, we will use the term *compute nodes* to refer all the physical machines (of IC or EC) able to host VMs and not in charge of any cloud management task. HyIaaS layer consists of three components (Fig.1b): the Monitoring Collector (MC), the Logic and the Translation component. MC is in charge of fetching information about the current resource utilization level of the on-premise *compute node*. The Logic component uses the information read by MC and implements a custom-defined spawning policy. Given the current status of the on-premise cluster and additional constraints possibly introduced by the customer (e.g., deadline for the execution of a certain job), the output of the Logic component is a new allocation of the VMs over the physical nodes, possibly including new VMs spawned off-premise.

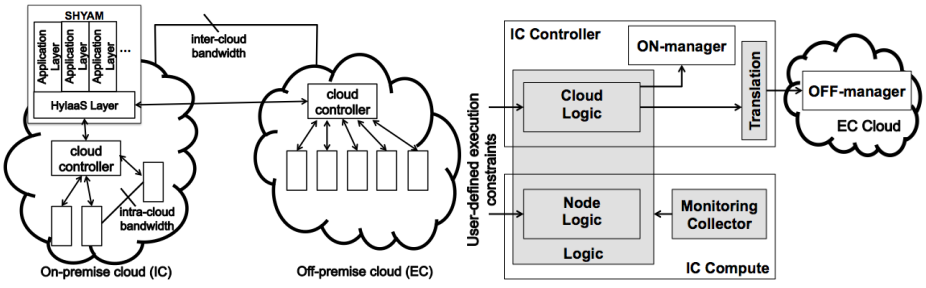


Fig. 1: Fig. 1a: Hybrid cloud scenario. SHYAM is an on-premise software component able to collect information about the current status of IC and dynamically add off-premise resources if needed. Fig. 1b: Hybrid Infrastructure as a Service layer in detail. Subcomponents are displayed in grey.

The Logic component has been split into two subcomponents: Node Logic and Cloud Logic. The Node Logic (one for each *compute*), responsible for analyzing the monitoring data from MC, detecting if a critical situation occurred on that physical machine (e.g., the *compute node* is too loaded) and sending notifications to the Cloud Logic. The Cloud Logic (installed on IC's controller node), in charge of autonomously taking spawning/migration decisions given the monitoring alerts received from Node Logic. The alerts from Node Logic and the policy of Cloud Logic can be defined by the IC system administrator. The rationale behind splitting the Logic component into two parts is to minimize the amount of information exchanged between the on-premise cloud controller and the physical nodes hosting VMs: the Node Logic sends notifications to the Cloud Logic only if a critical condition at node-level is detected. Having a wider vision of the state of the cloud, the Cloud Logic can combine the received information to implement a more elaborate policy. This should be taken into account by the IC system administrator, as she implements the spawning/migration policy. If the new VM allocation produced by the Logic involves EC, the Translation component is used to convert the directives into EC-specific APIs.

2.1 Applicative scenario

HyIaaS invokes the Application Layer functionalities via a standard interface. In particular, after HyIaaS has produced and deployed a new hybrid cluster structure, it calls the *configure* operation offered by the specific Application Layer involved, which is in charge of installing and configuring the applicative software on the newly provided virtual nodes.

As a case study, we focus on the data-intensive scenario, in which the application load can be distributed among several computing nodes. We adopt MapReduce [12], a widespread programming model to simplify the implementation of data intensive distributed applications. Following this approach, the input data-set is partitioned into an arbitrary number of parts, each exclusively processed by a different computing task, the *mapper*. Each *mapper* produces intermediate results (in the form of *key/value* pairs) that are collected and processed by other tasks, called *reducers*, in charge of calculating the final results by merging the *values* associated to the same *key*. The programs implemented according to this model can be automatically parallelized and easily executed on a distributed infrastructure. The MapReduce model is implemented by several platforms: one of the most popular is Apache Hadoop [1], an open source implementation consisting of two components: Hadoop Distributed File System (HDFS) and MapReduce runtime. The input files for MapReduce jobs are split into fixed size blocks (default is 64 MB) and stored in HDFS. MapReduce runtime follows a master-worker architecture. The master (Job-Tracker) assigns tasks to the worker nodes. Each worker node runs a Task-Tracker that manages the currently assigned tasks. Each worker node can have up to a predefined number of *mappers* and *reducers* simultaneously running. We execute the Hadoop workload over a virtual cluster that can be deployed on the hybrid cloud (partitioned between IC and EC) in case the on-premise resources are not enough. Therefore, the first Application Layer we implement for SHYAM is responsible

for installing and configuring Hadoop on the newly provided VMs and allows us to evaluate the performance of operating MapReduce in a hybrid cloud setup.

2.2 Logic component policy

Algorithm 1 SPAN policy

Input: $h, AL, ONmanager, OFFmanager, THR_U, \Delta t$.

```
1: while true do
2:   if h.getUtil() > THR_U then
3:     vmsToMove = selectToMove(h.getVMs())
4:     for each vm in vmsToMove do
5:       d = ONmanager.getAnotherAllocation(vm)
6:       if d! = null then
7:         ONmanager.migrate(vm, d)
8:       else
9:         vm_new = OFFmanager.provideLike(vm)
10:        AL.configure(vm, vm_new)
11:        vm_new = ONmanager.remove(vm)
12:      end if
13:    end for
14:  end if
15:  sleep(Δt)
16: end while
```

We implemented a first example of policy for the Logic component executed on every *compute* node of IC: the SPAN policy (Alg. 1). The algorithm aims to maintain the load of each *compute* node under a parametric threshold: THR_U. It periodically checks the resource utilization of the *compute* node h (line 2 in Alg. 1). If the load exceeds THR_U, the procedure selects to move a subset of the VMs currently on h (line 3 in Alg. 1). The $selectToMove$ function is implemented according to Minimization of Migrations algorithm from Beloglazov et al. [6]. This policy ensures to always move the minimum number of VMs that brings h utilization back under THR_U. For each vm selected, if there is another on-premise node that can host the VM, a migration is performed (line 7 in Alg. 1). Otherwise, if no IC's *compute* node can host the VM, a new one is spawned off-premise and the specific application level configuration is performed (line 10 in Alg. 1). As mentioned in section 2.1, we chose Hadoop as an example of distributed data-processing application. For this reason, the code of $AL.configure(vm, vm_{new})$ mainly consists of two operations, as show in Alg. 2. First of all vm_{new} is included in Hadoop virtual cluster (cl in line 1), then the old vm is decommissioned causing its data to be sent to other nodes of the cluster. Focusing on the first operation ($cl.include(vm_{new})$ in line 2), we must consider that Hadoop's Job-Tracker (running on the *master* node) assigns jobs to the workers according to the part of data currently allocated on the worker's portion of HDFS. Having no data initially allocated on the newly provided off-premise workers, they will be scarcely useful for the computation, because the Job-Tracker will not assign any task to them. Nevertheless, our solution relays on a well known Hadoop behavior: when the on-premise vm is decommissioned

and its data are replicated, Hadoop prefers the workers with low utilization of HDFS as destinations. Initially having 0% HDFS utilization, off-premise vm_{new} is likely to be preferred and no other data balancing is needed to give vm_{new} an effective role in computation. Therefore, $cl.exclude(vm)$ in line 3 of Alg. 2 is enough to trigger the data replication process and avoid the drawbacks of launching Hadoop Balancer process (which is high time-consuming mechanism [1] for equally redistribute data across the workers). It also produces the benefits of an inter-cloud VM migration (i.e., only vm's portion of HDFS is moved to EC) without performing the whole VM snapshot transfer.

Algorithm 2 $AL.configure$ procedure for Hadoop virtual cluster

Input: vm, vm_{new}.

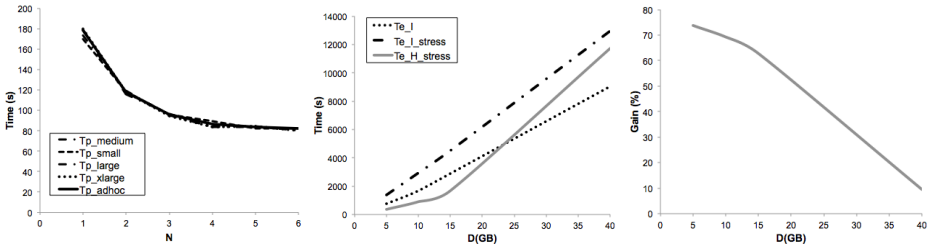
1: $cl = vm.getVirtualCluster()$
2: $cl.include(vm_{new})$
3: $cl.exclude(vm)$

2.3 Implementation

We implemented HyIaaS layer by extending OpenStack Sahara [3] component to allow cluster scaling operations in a hybrid scenario. OpenStack [2] is an open source platform for cloud computing with a modular architecture and Sahara is the OpenStack module specific to data processing. It allows the user to quickly deploy, configure and scale virtual clusters dedicated to data intensive applications like MapReduce. We modified the Sahara scaling mechanism to allow the spawning of new VMs on a remote cloud. The MC component is a simple daemon process running on each compute node. It checks the CPU, memory and disk utilization and compares them with THR_U. When the virtual cluster needs to be scaled by providing new off-premise VMs, the command is issued through the Translation component to EC. In our test scenario, the off-premise cloud runs another OpenStack installation, therefore the Translation component simply forwards the provisioning command to EC's Nova component (the central module for VM management in OpenStack infrastructure). Finally, the Application layer configures Hadoop and launches its daemons by connecting to the newly provided VMs.

3 Experimental Results

Our setup is composed of two OpenStack clouds to emulate IC and EC. The on-premise cloud has five physical machines, each one with a Intel Core Duo CPU (3.06 GHz), 4GB RAM and 225GB HDD. EC is composed of three physical machines, each one with 32 cores Opteron 6376 (1.4 GHz), 32GB RAM and 2.3TB HDD. On both IC and EC we provide *adhoc* VMs with two virtual CPUs, 4GB RAM and 20GB of disk. The intra-cloud bandwidth of IC and EC is 1000 Mbit/s, while the inter-cloud bandwidth (between the two) is 100 Mbit/s. Our initial scenario is composed of 4 VMs allocated on IC. In order to characterize the performance of the computation on the hybrid cluster, we first analyze the time to provide one or more new Hadoop workers on EC. Fig. 2a shows the average time to obtain a single Hadoop worker up and running as we vary the



(a) (b) (c)

Fig. 2: Fig. 2a shows the time to provide a certain number N of new off-premise VMs with different characteristics (small, medium, large and xlarge are the default VM configurations offered by OpenStack). Fig.2b compares the time to perform Hadoop word count on a fully on-premise cluster – with (Te_I) or without (Te_I_stress) a stressing condition on a physical node –, with the performance on a hybrid cluster created by the HyIaaS. Fig.2c shows the percentage gain obtained by our solution.

number of VMs spawned at a time on EC. As we expected, the trend of the curve suggests that there is a constant overhead caused by SHYAM provisioning mechanism, but the trend of total time is approximately linear with the number of VMs requested. Furthermore, we can easily verify from the graph in Fig. 2a that provisioning time is independent from the characteristics of the specific VM spawned. Given SHYAM's autonomic provisioning mechanism, we can evaluate the performance of operating MapReduce in a hybrid cloud setup. To this purpose, we assume to have four on-premise VMs already configured to run Hadoop jobs and provided with a certain amount of data D on HDFS, and we consider the time to execute a word count Hadoop job [12] over Wikipedia datasets of different size D [4]. Fig. 2b compares the execution time trends of three scenarios. The first one (Te_I in Fig. 2b) represents the ideal situation of having each Hadoop VM allocated on an on-premise dedicated physical machine. Since no other physical or virtual load is affecting the execution, we can obtain good performance (execution time is linear in D). The second scenario (Te_I_{stress} in Fig. 2b) shows the performance degradation when one of the four Hadoop workers is running on a overloaded physical machine and no VM redistribution mechanism is adopted. As we can see in Fig. 2b, the execution time is considerably higher when compared to Te_I because the VM on the stressed physical node sensibly slows down the whole distributed computation. The third scenario (Te_H_{stress} in Fig. 2b) repeats the second scenario and adopts SHYAM redistribution with SPAN policy. THR_U and THR_D are fixed at 90% and 10% respectively. In this case, a new VM is spawned off-premise and the on-premise worker running on the stressed machine is decommissioned (i.e., excluded from Hadoop cluster after its data have been copied on other worker nodes). This operation causes a part of data to cross on-/off-premise boundaries. As we can see in Fig. 2b, the adoption of SPAN policy can considerably improve the performance for low values of D. However, the trends show that in the third scenario the execution time is not linear in the volume of data involved and, for high values of D, we can have a lower execution time by avoiding the off-premise spawning. This is mainly due to data movement across the on-/off-premise boundaries, which is usually over a

higher latency medium when compared to a fully on-premise computation. Fig. 2c shows the gain in execution time obtained with SHYAM. Although for high volumes of data crossing on-/off-premise boundaries the gain is low, the graph suggests that the autonomic provisioning and configuration of VMs on EC can represent a good solution to face critical conditions of stress in private clouds. In the case of a word count application, the graphic in Fig. 2c suggests a pseudo linear correlation between the amount of data D and the gain obtained by providing a new off-premise VM and decommissioning the slow on-premise worker. This property can be useful to a priori estimate the advantage of SHYAM's cloud bursting given a certain volume of data D to be processed.

4 Related Work

Cloud computing is currently used for a wide and heterogeneous range of tasks. According to the classification introduced in [5], in this work we especially focus on the cloud from the *IaaS* perspective, intending it as an elastic provider of virtual resources, able to contribute to heavy computing tasks. Data-intensive applications are an example of resource demanding tasks. A widely adopted programming model for this scenario is MapReduce [12], whose execution can be supported by platforms such as Hadoop [1], possibly in a cloud computing infrastructure. We tested our system with MapReduce applications, choosing Hadoop as execution engine. Recently, a lot of work has focused on cloud computing for the execution of big data applications: as pointed out in [11], the relationship between big data and the cloud is very tight, because collecting and analyzing huge and variable volumes of data require infrastructures able to dynamically adapt their size and their computing power to the application needs. The work by Chen et al. [9] presents an accurate model for optimal resource provisioning useful to operate MapReduce applications in public clouds. Similarly, Palanisamy et al. [17] deal with optimizing the allocation of VMs executing MapReduce jobs in order to minimize the infrastructure cost in a cloud datacenter. In the same single-cloud scenario, Rizvandi et al. [18] focus on the automatic estimation of MapReduce configuration parameters, while Verma et al. [20] propose a resource allocation algorithm able to estimate the amount of resources required to meet MapReduce-specific performance goals. However, these models were not intended to address the challenges of the hybrid cloud scenario, which is the target environment of our work.

The choice of primarily relay on a small (e.g., private) cloud and then use the extra-capacity offered by a public cloud for opportunistic scale-out has been investigated by several authors [8, 7]. According to the classification in [19], our work mainly deals with the hybrid cloud approach for cloud interoperability, because the main motivation of our system is allowing cloud bursting to EC. However, our proposal could also be classified as a *Federation* mechanism for cloud aggregation because – as in federated clouds – the interoperation between clouds is completely transparent to end-users. The works in [16] and [13] focus on enabling cloud bursting through inter-cloud migration of VMs, which is generally a time and resource expensive mechanism. In particular, [13] optimizes the overhead of migration using an intelligent pre-copying mechanisms that proactively

replicates VMs before the migration. Our work doesn't take into consideration the VM migration, but only the dynamic instantiation of new compute nodes on EC, thus to avoid the unnecessary movement of the whole VM snapshot across the cloud boundaries. As we shown, this technique is particularly suitable for the MapReduce model because the Hadoop provisioning and decommissioning mechanism intrinsically contributes to simplify the cloud bursting process. The hybrid scenario is also investigated in the work by Zhang et al. [21] by focusing on the workload factoring and management across federated clouds. More similarly to our approach, cloud bursting techniques has been adopted for scaling MapReduce applications in the work by Mattess et al. [15], which presents an online provisioning policy to meet a deadline for the Map phase. Differently from our approach, [15] does not consider the time to balance data across the hybrid virtual cluster, which, as we showed, has a consistent role in determining the opportunity of cloud bursting towards the public cloud. Also the work presented by Kailasam et al. [14] deals with cloud bursting for big data applications. It proposes an extension of the MapReduce model to avoid the shortcomings of high latencies in inter-cloud data transfer: the computation inside IC follows the batch MapReduce model, while in EC a stream processing platform called Storm is used. The resulting system shows significant benefits. Differently from [14], we have chosen to keep complete transparency and uniformity in working node allocation and configuration. However, as in [14], our system allows the user to constrain the allocation of *mappers/reducers* in order to optimize the cost of data transfer between these tasks.

5 Conclusion

In this paper we presented SHYAM, a software component to allow VM autonomic management in a hybrid cloud scenario. We illustrated the architecture and the internal structure of the system and we evaluate its performance by executing a Hadoop data-intensive application on a virtual cluster and stressing one of IC's physical machines. In the given scenario, SHYAM autonomously spawns new VMs on EC and configures them as workers of the Hadoop cluster. Our results show that the time to provide a new off-premise worker is not influenced by the characteristics of the VM requested and the hybrid cluster obtained can sensibly improve the performance of a benchmark Hadoop word count application, although the performance of the hybrid cluster decreases as the inter-cloud bandwidth is saturated. Since this drawback could be also influenced by the kind of application executed (e.g., word count in our case study), we plan to further investigate SHYAM performance with different Hadoop workloads. Nevertheless, this work represents a first prototype of an autonomic infrastructure for hybrid clouds able to detect critical conditions on IC and autonomously request resources to EC.

As regards SPAN policy, we trigger the spawning/migration mechanism if the physical machine's CPU utilization exceeds THR_U. However, the policy could be easily modified to take into account the utilization of other resources (RAM, disk, etc.). Furthermore, in case of spawning new VMs towards EC our approach

lacks a mechanism for bringing back on IC the off-premise VMs once the critical condition is solved. Therefore, for the future, we plan to enrich the policy with a similar threshold mechanism to detect underloaded hosts in IC. This mechanism will have to be equipped with memory of the past actions taken in order to avoid the continuous provisioning and de-provisioning of VMs (moving data back and forth between IC and EC) due to small variations on the load of the physical machine.

References

1. Apache hadoop, https://hadoop.apache.org/
2. Openstack: Opensource cloud computing software, https://www.openstack.org/
3. Openstack sahara, https://wiki.openstack.org/wiki/Sahara
4. Puma datasets, https://engineering.purdue.edu/ puma/datasets.htm
5. Armbrust, M., Fox, O.: Above the clouds: A berkeley view of cloud computing. Tech. rep., Electrical Engineering and CS University of California (2009)
6. Beloglazov, A., Abawajy, J., Buyya, R.: Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. Future Generation Computer Systems 28(5), 755–768 (May 2012)
7. Bicer, T., Chiu, D., Agrawal, G.: A framework for data-intensive computing with cloud bursting. In: IEEE International Conference on Cluster Computing (2011)
8. Cardosa, M., Wang, C., Nangia, A., Chandra, A., Weissman, J.: Exploring mapreduce efficiency with highly-distributed data. In: Proceedings of the Second International Workshop on MapReduce and Its Applications. ACM (2011)
9. Chen, K., Powers, J., Guo, S., Tian, F.: Cresp: Towards optimal resource provisioning for mapreduce computing in public clouds. Parallel and Distributed Systems, IEEE Transactions on 25(6), 1403–1412 (June 2014)
10. Chen, M., Mao, S., Liu, Y.: Big data: A survey. Mobile Networks and Applications Volume 19(2), 171–209 (2014)
11. Collins, E.: Intersection of the cloud and big data. IEEE Cloud Computing (2014)
12. Dean, J., Ghemawat, S.: Mapreduce: Simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (January 2008)
13. Guo, T., Sharma, U., Shenoy, P., Wood, T., Sahu, S.: Cost-aware cloud bursting for enterprise applications. ACM Trans. Internet Technol. 13(3) (May 2014)
14. Kailasam, S., Dhawalia, P., Balaji, S.: Extending mapreduce across clouds with bstream. Cloud Computing, IEEE Transactions on (2014)
15. Mattess, M., Calheiros, R., Buyya, R.: Scaling mapreduce applications across hybrid clouds to meet soft deadlines. In: IEEE 27th International Conference on Advanced Information Networking and Applications. pp. 629–636 (2013)
16. Nagin, K., Hadas, D.: Inter-cloud mobility of virtual machines. In: Proceedings of the 4th Annual International Conference on Systems and Storage. ACM (2011)
17. Palanisamy, B., Singh, A., Liu, L.: Cost-effective resource provisioning for mapreduce in a cloud. Parallel and Distributed Systems, IEEE Transactions on (2015)
18. Rizvandi, N., Taheri, J.: A study on using uncertain time series matching algorithms for mapreduce applications. Concurrency and Computation (2013)
19. Toosi, A.N., Calheiros, R.N., Buyya, R.: Interconnected cloud computing environments: Challenges, taxonomy, and survey. ACM Comput. Surv. 47(1) (2014)
20. Verma, A., Cherkasova, L., Campbell, R.H.: Resource Provisioning Framework for MapReduce Jobs with Performance Goals. Springer (2011)
21. Zhang, H., Jiang, G., Yoshihira, K.: Proactive workload management in hybrid cloud computing. IEEE Transactions on Network and Service Management (2014)