Enabling Big Data Analytics in the Hybrid Cloud using Iterative MapReduce

Francisco J. Clemente-Castelló*, Bogdan Nicolae†, Kostas Katrinis†, M. Mustafa Rafique†, Rafael Mayo*, Juan Carlos Fernández*, Daniela Loreti‡

*Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I 12.071–Castellón, Spain.
Email: {fclement, mayo, jfernand}@uji.es

†IBM Research - Damastown, Mulhaddart, Dublin, Ireland.
Email: {bogdan.nicolae, katrinisk, mustafa.rafique}@ie.ibm.com

‡Department of Computer Science and Engineering University of Bologna, Bologna, Italy
Email: daniela.loreti@unibo.it

***Abstract*—The cloud computing model has seen tremendous commercial success through its materialization via two prominent models to date, namely public and private cloud. Recently, a third model combining the former two service models as on-/off-premise resources has been receiving significant market traction: hybrid cloud. While state of art techniques that address workload performance prediction and efficient workload execution over hybrid cloud setups exist, how to address data-intensive workloads - including Big Data Analytics - in similar environments is nascent. This paper addresses this gap by taking on the challenge of bursting over hybrid clouds for the benefit of accelerating iterative MapReduce applications. We first specify the challenges associated with data locality and data movement in such setups. Subsequently, we propose a novel technique to address the locality issue, without requiring changes to the MapReduce framework or the underlying storage layer. In addition, we contribute with a performance prediction methodology that combines modeling with micro-benchmarks to estimate completion time for iterative MapReduce applications, which enables users to estimate cost-to-solution before committing extra resources from public clouds. We show through experimentation in a dual-Openstack hybrid cloud setup that our solutions manage to bring substantial improvement at predictable cost-control for two real-life iterative MapReduce applications: large-scale machine learning and text analysis.**

***Index Terms*—Hybrid Cloud; Big Data Analytics; Iterative Applications; MapReduce; Data locality; Performance Prediction**

I. Introduction

The *"public cloud"* service model has transformed IT over the years thanks to pay-as-you-go [1] (e.g. Amazon EC2) and the ability to quickly serve a large number of tenants from a common pool of virtualized and/or bare-metal, physically co-located (i.e. single datacenter) IT resources. Various technical (e.g. performance isolation, custom technology needs) and business (e.g. security, outsourcing of compliance burden) requirements have sparked an alternative cloud delivery model, termed *"private cloud"* [2] (e.g. IBM Softlayer Private Cloud). Despite its many commonalities with public cloud in terms of underlying technology and software stacks, the primary differentiator of the private model lies in the delivery of cloud services over a physical infrastructure that is exclusively dedicated to the tenant. Furthermore, this is usually coupled with additional foundational infrastructure services managed by the provider, such as provisioning, security, compliance and connectors.

Recently, a new cloud service model has been emerging, namely *hybrid cloud*. As the term implies, the hybrid model encompasses a combination of two or more instances of the previously outlined models, typically in the form of private/public: a set of VMs (virtual machines) hosted on the private cloud (*on-premise*) are temporarily complemented with VMs hosted on a public cloud (*off-premise*). The business reasons driving such an evolution are multi-fold, some of which are: cloud-bursting [3] from private to public clouds to overcome service hotspots without extending the private cloud with additional physical infrastructure, disaster recovery [4] (i.e. move/replicate to a public cloud to survive extensive private cloud failures), combining on-premise sensitive data processing with open data hosted on public clouds, etc. Market studies [5] forecast a hybrid cloud market size worth of $80 Billions by 2018, anticipated due to an aggressive average cumulative annual growth rate of 30% until 2018.

Hybrid cloud services have already started to be commercially available, e.g. VMWare vAir (infrastructure service) and Rackspace hybrid cloud hosting (platform service). Among the various widely used and trending platform and/or application software services that leverage the hybrid cloud delivery model [6], [7], data-intensive analytics at large scale is one of the most challenging cases, especially when large volumes of data are involved (often called "Big Data Analytics"). Conventional big data analytics frameworks assume physical co-location of IT resources with high-speed interconnection among servers. This assumption is lifted in hybrid cloud environments comprising physically separated datacenters that are interconnected via a network that is at least an order of magnitude slower (either dedicated connectivity or the open Internet). Furthermore, due to the sheer size of the data being analyzed, most big data analytics techniques make heavy use of data locality by shipping the computation close to the data. This aspect is particularly challenging when boosting a private cloud with extra temporary VMs from a public IaaS cloud to finish a big data analytics job faster: the newly provisioned VMs do not hold any data and as such it is necessary to send large amounts of data over the slow link in order to be able to leverage the extra computational capability. As such, data movement is elevated to a major challenge in "hybrid cloud big data analytics", which has various runtime and data management implications at the level of the storage layer.

This paper tackles the major challenge outlined above using MapReduce [8] as a reference analytics framework embodiment due to its significant deployment base and traction. We focus on one specific class of big data analytics applications that is particularly suitable for "hybrid cloud big data analytics": iterative applications that reuse invariant input data. For this class of applications, data locality can be leveraged over and over again once the input data was replicated off-premise. However, given the large initial overhead of the data movement, an efficient solution that facilitates data locality is non-trivial. Furthermore, since the extra off-premise resources incur pay-as-you-go costs, it is crucial to estimate the performance gains in advance, in order to be able to decide whether it is worthwhile to commit any extra resources at all, and, if so, how many of them in order to achieve the desired cost-performance trade-off. This paper aims to address both directions. We summarize our contributions as follows:

- We propose a novel technique that minimizes data movement over the inter-cloud network and thus guarantees elevated levels of data locality, while preserving cross-cloud data replication. We achieve this in a completely transparent fashion, without invasive changes to the MapReduce framework or the underlying storage layer by adapting existing features to the hybrid setup (Section IV).
- We propose a performance prediction methodology that combines analytical modeling with micro-benchmarking to estimate time-to-solution in a hybrid setup, including any data movement and computation (Section V).
- We evaluate our approach in a series of experiments that involve two representative real-life iterative MapReduce applications exhibiting a highly intensive map phase that processes large input datasets. Our experiments demonstrate both the ability to achieve strong scalability using our data movement technique, as well as small prediction errors (Section VI).

II. Related work

Recent years have seen extensive enhancements in the Hadoop framework. Several efforts to improve the MapReduce performance leverage cloud technologies [9]–[12], with the assumption of having unlimited computing resources. These efforts are complemented by works on storage elasticity [13], [14], which explore how to deal with the explosion of the costs related to the storage capacity and the I/O bandwidth, and are highly relevant for iterative HPC and MapReduce applications.

A few recent efforts are focused on improving the performance of MapReduce frameworks for hybrid environments. HybridMR [15] proposes a solution for executing MapReduce in hybrid desktop grids and external voluntary nodes, but does not consider expanding and shrinking a MapReduce setup dynamically. HadoopDB [16] proposes a hybrid system comprising Hadoop and parallel database systems to yield the resilience, and scalability of Hadoop and the performance and efficiency of parallel databases. Similarly, hybrid scheduling techniques [17]–[19] uses GPUs to improve the performance of MapReduce applications in accelerator-enabled clusters. These techniques uses hybrid computing to improve the performance of MapReduce applications, but the hybrid aspect is on the computational side rather than the networking side.

Specifically to workload performance prediction and optimization in hybrid clouds, Bossche et al. [20] have proposed a linear/integer programming model for relevant workloads, showing substantial improvement over naive executions. Albeit valuable from a bounds perspective, such models are typically hard to scale and also fail to capture framework-specific intricacies, such as data-movement and data locality. Imai et al. [21] explore hybrid cloud prediction patterns from the perspective of selection of resource units (virtual machine sizing types), with prediction techniques that could couple the present work in terms selecting matching VM sizes for hosting MapReduce daemons. The hybrid cloud extensions to MapReduce for the Aneka Cloud [22] constitutes another closely related work to this paper. It relies on the assumption that the distributed filesystem is persistently deployed and loaded with data throughout the execution of MapReduce workloads. As such, it does not capture the initial phase of cross-cloud data distribution and data-balancing, a vital phase in high value hybrid cloud use-cases, such as cloud-bursting.

A solution is proposed [23] for running data analysis applications when data is distributed across local and remote cloud systems. This work assumes an adapted MapReduce API that uses a local and global reduction for running Map and Reduce tasks to avoid the overheads of data transfers across local and remote clouds. In this work, a MapReduce job is executed either the local cluster or the cloud cluster using a *head* component, whereas, our approach executes a job on local as well as on the remove cloud concurrently using the resources available.

A hybrid cloud computing [24] model with an intelligent workload factoring service for proactive workload management proposes a fast frequent data item detection algorithm. The proposed hybrid architecture consists of a base workload that is executed on the local cloud and a flash crowd workload that is executed on the remote cloud. Instead, our approach focuses on single MapReduce workload where the processes exhibit tighter coupling and data dependencies.

III. Challenges of data locality in hybrid IaaS clouds

MapReduce applications typically exhibit a high degree of data parallelism: massive amounts of data are transformed in an embarrassingly parallel fashion in a *map* phase, after which they are aggregated in a *reduce* phase. This approach puts a high burden on the storage layer: it needs to serve a large number of concurrent read requests corresponding to the input data of the map phase, as well as a large number of concurrent write requests corresponding to the output of the reduce phase.

In this context, using a conventional distributed file system that is decoupled from the MapReduce runtime is not enough to deal with such highly concurrent I/O access patterns: this would incur a massive amount of network traffic, overwhelming the networking infrastructure and offsetting the benefits of storing the data in a distributed fashion. For this reason, a key design choice of MapReduce is the ability to take advantage of *data locality*: the storage layer is co-located with the MapReduce runtime on the same nodes and is specifically designed to expose the location of the data blocks, effectively enabling the scheduler to bring the computation close to the data and avoid a majority of the storage-related network traffic. By replacing the nodes with virtual machines, a similar configuration that can efficiently exploit data locality can be obtained in an IaaS cloud as well.

However, in a hybrid cloud setup, there are two major challenges. First, the storage layer and all data is deployed initially only on-premise. Thus, when additional off-premise VMs are provisioned from the external cloud provider to boost the initial setup, they cannot benefit out-of-the-box from data locality and need to fetch/write their data to/from the on-premise VMs. Second, the link between the on-premise infrastructure and the external cloud provider is typically of limited capacity. Thus, off-premise VMs that need to communicate with on-premise VMs create a network bottleneck much faster than the case when all VMs are located within the same cloud.

These two challenges are even more exacerbated in the context of iterative applications: in many cases, a majority of the input data needed for the first iteration will be needed for the subsequent iterations (such data is called the *invariant*). Thus, adopting a naive solution where the off-premise VMs read the input data from the on-premise VMs over and over again over a weak link is not feasible. Furthermore, for the data that changes from iteration to iteration, off-premise VMs need to constantly write their output remotely, then read it back in the subsequent iteration, again over the weak link. Given these circumstances, exploring a better solution that improves the ability to take advantage of data locality in a hybrid setup is critical.

IV. Asynchronous Data Rebalancing Technique

This section describes our proposal to enable efficient execution of iterative MapReduce jobs in a hybrid IaaS cloud setup. It focuses on defining a strategy to address the technical challenges mentioned in the previous section.

At first sight, the problem of avoiding remote data transfers over the weak link seems to be easily addressable by using a conventional caching solution: the invariant data and the newly written data can simply be stored locally on the off-premise VMs for faster subsequent access. However, adopting such a caching strategy is non-trivial, because it needs to integrate well into the whole MapReduce framework.

More specifically, since the scheduling of tasks is deeply linked with the data locality, the MapReduce scheduler will prefer on-premise VMs over off-premise VMs, which leads to a scenario where the off-premise VMs are underutilized. Furthermore, even if the scheduler would not exhibit such preference and would rather aim for load balancing, it is not enough to simply cache the data blocks off-premise and expose their location, because the storage layers fills other roles as well: replication support for resilience and high availability, load balancing of the data distribution, etc. Thus, in order to scale and properly take advantage of all these features, it is important to extend the storage layer beyond the on-premise VMs and re-balance the data blocks so that they are spread both over the on-premise VMs as well as the off-premise VMs. Also important are other non-functional aspects: users prefer to use a standard MapReduce distribution (e.g. *Hadoop*) that was tested and tuned in their on-premise cloud, rather than switch to a dedicated solution specifically written for a hybrid setup. Furthermore, switching to a custom storage layer may not always be feasible: for example, if a huge amount of data is already stored in a regular on-premise MapReduce deployment, the overhead of migrating to a custom storage layer might offset the benefits of enabling the hybrid support altogether.

For these reasons, we propose a non-invasive solution that solves the aforementioned issues without deviating from the standard storage layer. Our key idea is to leverage *rack awareness*, a feature typically implemented in production-ready MapReduce storage layers, such as *HDFS* [25]. Originally intended as a mechanism to enhance fault tolerance, rack awareness enables the user to specify for each HDFS node that is part of the deployment a logical group, typically corresponding to a physical rack of the cluster where Hadoop is deployed. Using this information, HDFS replicates each data block at least once outside of the group where it was written, under the assumption that such a behavior improves the ability to resist catastrophic failures where a whole rack would fail at once.

In our context, we leverage rack awareness from a novel perspective. Specifically, we create two logical groups: one for the on-premise VMs and another for the off-premise VMs. Thus, when new off-premise VMs are provisioned to boost the capability of the already running on-premise VMs, we extend the HDFS deployment in a rack-aware fashion on the off-premise VMs. Using this approach, whenever an off-premise VM writes a new data block, it actually writes both local copies (solving the locality issue) as well as at least a remote copy, which enables efficient storage elasticity: off-premise VMs can be simply killed as desired without having to worry about transferring the data back to the on-premise side. The only remaining issue is that the HDFS data nodes running on the off-premise VMs are initially empty, which prompts the need to re-balance the initial data blocks in order achieve load balancing and enable the scheduler to fully take advantage of the off-premise VMs. However, re-balancing has its own overhead and as such is subject to a trade-off: at one extreme one can wait until all invariant data is balanced, which enables a maximum acceleration of the iterations from the beginning; at the other extreme one can run the re-balancing asynchronously, which eliminates the initial overhead at the cost of gradual acceleration of the iterations as the data balancing progresses. We opted for the second option, since the initial overhead of re-balancing is significant and the ability to overlap the computation with the data transfers is crucial. While more elaborate balancing strategies (e.g. wait until a certain number of blocks was transferred off-premise then switch to the asynchronous strategy) are possible to explore, this is outside the scope of this work.

V. Performance Prediction Model

In some cases, using a hybrid cloud is a functional requirement: there are simply not enough resources on-premise to run the application with the desired level of complexity. However, most of the time, users are interested in a hybrid solution because they intend to accelerate their application by renting extra off-premise VMs. Since a hybrid solution incurs additional costs, it is important to understand how much the hybrid solution can accelerate the application, given a number of extra off-premise VMs. Ideally users would like to have the answer in advance, in order be able to decide apriori whether it is worthwhile to commit the extra off-premise VMs or not. In this section, we propose a performance prediction methodology that addresses this issue.

A. Assumptions

We assume a MapReduce deployment that initially spans N on-premise VMs where all initial invariant data is distributed. These N on-premise VMs are complemented by M extra off-premise VMs, where we extend the MapReduce deployment using the asynchronous rack-aware re-balancing strategy mentioned in the previous section and then run the iterative application on the resulting hybrid setup. For simplicity, we assume the on-premise VMs are identical in capabilities to the off-premise VMs. Furthermore, we assume that the user has access to the historical traces of the application or can estimate important MapReduce metrics: total number of map/reduce tasks (p_M and p_R); total number of map/reduce slots (k_M and k_R); average map/reduce/shuffle duration (A_M, A_R, A_S), average data/shuffle sizes per map/reduce/shuffle task (D_M, D_R, D_S). Also, we assume that the iterative applications exhibit a well-defined behavior: the number of iterations (I) is known in advance and the map/reduce tasks do not change in terms of number, amount of input data and computational complexity from one iteration to another.

B. Performance model for on-premise jobs

Using these metrics, techniques to estimate the runtime of MapReduce jobs on a single cluster have been proposed before and can be used in our case for the N on-premise VMs. In particular, Verma et. al. propose a model based on the make-span theorem [26], which states that for a greedy assignment of p tasks on k workers, the lower and upper bound for the execution time is $p \cdot A/k$ and, respectively, $(p-1) \cdot A/k + \lambda$, with A the average execution time of the tasks and λ the execution time of the slowest task. Intuitively, the lower bound corresponds to an ideal scenario where there is perfect load balancing, while the upper bound corresponds to a worst case scenario where the slowest task is scheduled last, after all other $p-1$ tasks finished in at most $(p-1) \cdot A/k$ time.

Since MapReduce does not overlap the map phase with the reduce phase, both can be treated separately using the make-span theorem. For simplicity, we focus in this paper on the lower bound only. The upper bound can be estimated in a similar fashion.

$$T_M^{low} = A_M \cdot \frac{p_M}{k_M} \qquad T_R^{low} = A_R \cdot \frac{p_R}{k_R}$$

More complexity is introduced by the shuffle phase, for which the first wave overlaps with the map phase and thus the resulting overhead needs to be considered separately (denoted A_{S1}). For the rest of the shuffle waves ($p_R/k_R - 1$), the make-span theorem can be applied as usual:

$$T_S^{low} = A_S \cdot (\frac{p_R}{k_R} - 1) + A_{S1}$$

Thus, the estimated completion time for a single iteration is:

$$T^{low} = T_M^{low} + T_R^{low} + T_S^{low}$$

Considering all iterations, the total estimated completion time is:

$$T^{low} = \sum_{i=1}^{I} (T_{M_i}^{low} + T_{R_i}^{low} + T_{S_i}^{low})$$

C. Performance model for hybrid jobs

In a hybrid setup, two important aspects affect the estimations discussed above: (1) while the asynchronous rebalancing progresses in the background, it generates extra overhead, which will slow down the map/shuffle/reduce (2) due to the weak link between the on-premise and off-premise VMs, the data transfer during the shuffle may experience a slowdown.

Due to the fact that (2) is highly complex and dependent on the nature of the application, we focus in this work on (1), leaving (2) as future work. Thus, we propose to amend the equations above such that they reflect the rebalancing aspect. Specifically, two important factors characterize this aspect. First, while the rebalancing is in progress, the mappers become slower due to the additional background activity. We denote this slowdown as α. It remains greater than 1 while the rebalance is in progress and equals 1 after the rebalance is complete. Second, as more data is transferred off-premise during the rebalancing, more locality can be exploited by the scheduler, which effectively translates to more mappers that are scheduled off-premise. For simplicity, we adopt a simple heuristic to account for this effect that assumes only rack-local mappers will be executed the scheduler. This roughly corresponds to real life: only a negligible fraction of mappers are not scheduled rack-local. Furthermore, we make another simplifying assumption: all mappers are scheduled at the beginning of the iteration. Under these circumstances, the total number of parallel mappers during iteration i (denoted k_{M_i}) depends on the progress of the rebalancing at the beginning of the iteration, ranging from the map slots available on-premise only (k_{M_1}) to the map slots available both on-premise and off-premise after the rebalancing is complete.

For the rest of this paper, we refer to α and k_{M_i} as the *hybrid rebalance factors*. Thus, for the hybrid case, the estimated completion time for the map phase is:

$$T_M^{low} = \sum_{i=1}^{I} \alpha \cdot A_M \cdot \frac{p_M}{k_{M_i}}$$

$$T_M^{up} = \sum_{i=1}^{I} \alpha \cdot A_M \cdot \frac{p_M - 1}{k_{M_i}} + \lambda$$

D. Methodology to leverage the hybrid performance model

In order to make use of the hybrid performance model introduced above for actual predictions, we need to estimate the hybrid rebalance factors. However, due to the complex inter-play between the system, the virtualization layer and the MapReduce framework that depends on a variety of parameters (i.e., point-to-point bandwidth between VMs, aggregated bandwidth of the weak link between on-premise and off-premise VMs, I/O pressure on the local storage, etc.), it is not easy to determine them analytically.

Thus, we propose to establish them experimentally, by using a series of micro-benchmarks that are executed on the hybrid setup independently of the application. More specifically, given N on-premise VMs and a desired number of M off-premise VMs, we create a similar setup as if running the actual application (i.e. same data size, number of mappers, etc.). However, instead of running the application, we run an I/O intensive benchmark that approximates the application behavior for the duration of the re-balancing.

To obtain α, we simply divide the result of the hybrid benchmark by the baseline (i.e. same I/O intensive benchmark running on-premise only). To obtain k_{M_i}, we correlate the rebalance progress observed during the I/O benchmark with the moment when each iteration starts for the real application as follows:

$$k_{M_i} = min(k_M^{max}, B_M(T_{M_{i-1}})/D_M)$$

In the equation above, $B_M(t)$ denotes the amount of data transferred off-premise at moment t, k_M^{max} denotes the total number of mapper slots available from the $N + M$ VMs both on-premise and off-premise, while D_M denotes the data size processed by each mapper. By convention, $T_{M_0} = 0$.

Note that it is not necessary to run the I/O intensive micro-benchmarks for before running every real application: these results can be cached and reused later if the off-premise setup is unchanged (e.g. same type of VMs, same aggregated throughput between the off-premise and on-premise nodes). Since in many cases it is possible to use historical micro-benchmark results to calculate the hybrid rebalance factors, we differentiate the running of the I/O micro-benchmarks from the actual calculation of the factors, which we henceforth refer to as *micro-calibration*.

Once the micro-calibration is done, T^{low} and T^{up} can be estimated as described in Section V-C. To find an optimal configuration, one can simply take a set of representative values for M and calculate T^{low} for each M. Armed with the knowledge of how long the execution time is likely to be for a variable M, it is easy to estimate whether a speed-up is possible in the first place, and, if so, how much extra cost would be necessary to achieve it. Furthermore, since we target iterative applications, we can estimate the completion times for an arbitrary number of iterations, which aids in choosing the right trade-off between cost and precision of results.

VI. Evaluation

A. Experimental setup

The experiments for this work were performed on the *Kinton* testbed of the HPC&A group based at Universitat Jaume I. It consists of 8 nodes, all of which are interconnected with 1 Gbps network links and split into two

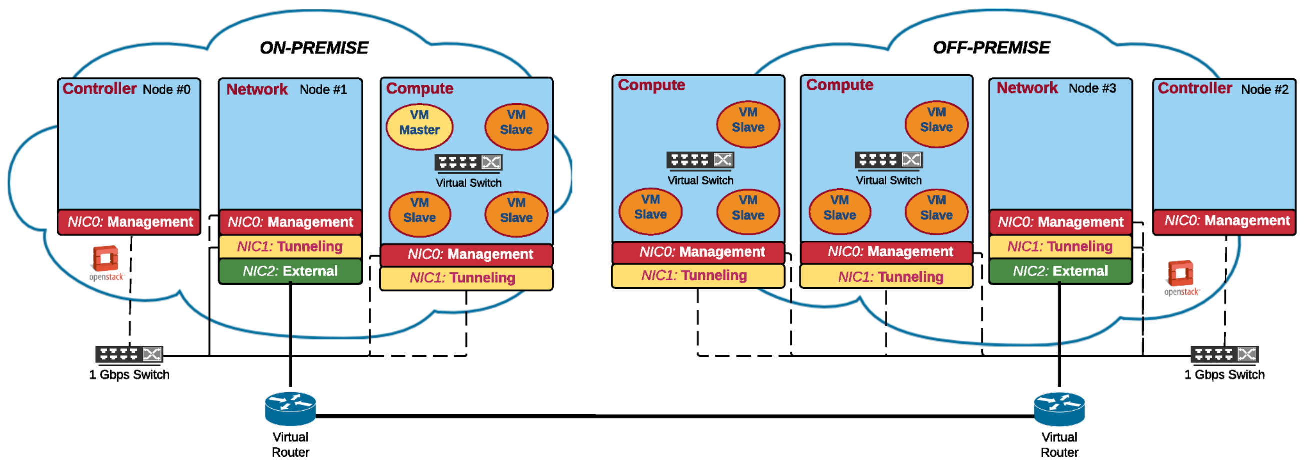


Fig. 1. Hybrid IaaS OpenStack cloud example: one *fat* node on-premise and two *fat* nodes off-premise

groups: four nodes feature an Intel Xeon X3430 CPU (4 Cores), HDD local storage of 500 GB, and 4 GB of RAM. These less powerful nodes (henceforth called *thin*) are used for management tasks. The other four nodes feature two Intel Xeon E5-2630v3 (2 x 8 Cores), HDD local storage of 1 TB, and 64 GB of RAM. These more powerful nodes (henceforth called *fat*) are used to host the VMs.

In order to get as close as possible to a real-life hybrid cloud, we configure two separate IaaS clouds based on OpenStack Icehouse and QEMU/KVM 0.12.1 as the hypervisor. One of the OpenStack deployments acts as the on-premise cloud, while the other one acts as the off-premise cloud. A fully-featured OpenStack deployment requires two management nodes: one *controller node* that manages the compute nodes where the VMs are hosted and one *network node* that manages the cloud networking, which is managed separately due to the complexity of the networking technologies involved.

More specifically, in a typical configuration based on *neutron* (the standard OpenStack network management service), there are three conceptually separated communication domains: the management network (i.e., used for control messages and administrative traffic), the internal network (i.e., traffic between the VM instances using private IP addresses) and the external network (i.e., traffic between the VM instances and the outside of the cloud). In this configuration, the VM instances are configured to directly communicate with each other via the links of their compute node hosts. However, all communication with the outside of the cloud is routed through the network node, which is equipped with three NICs, each dedicated to a communication domain. Thus, in a real-life hybrid cloud setup that involves two OpenStack deployments, any communication between on-premise and off-premise VMs will pass through the network nodes, which become the weak link (i.e., total aggregated throughput between all on-premise and off-premise VMs is 1 Gbps).

For our experiments, we created a new VM flavor: *i2.xlarge*. This flavor features 4 vCPU, HDD local storage of 100 GB and 16 GB of RAM. Thus, each compute node has the capacity to host 4 VMs simultaneously. Since two VMs that are co-located on the same compute node can communicate at much higher rate that two VMs that are hosted on different compute nodes, we limit the network capacity of this flavor to 1 Gbps to obtain a close-to-uniform environment where all VMs can communicate with each other at the same rate, regardless where they are located. This setup is illustrated in Figure 1, using one fat on-premise node and two fat off-premise nodes. On the on-premise part we provision 4 VMs in which Hadoop version 2.6.0 was deployed. One of these VMs is used as Hadoop master and the others as Hadoop slaves. Specifically, each Hadoop slave is configured both as a HDFS DataNode and as a YARN slave, with enough capacity to run 4 mappers and reducers simultaneously. On the off-premise part, we provision 4 VMs on each fat node, with a variable number of fat nodes ranging from one to three. In order to extend the Hadoop deployment over the off-premise VMs, we start the relevant services (i.e., HDFS DataNodes and YARN runtime) on the off-premise VMs. These services will report to the master, which integrates them into the Hadoop deployment. Rack-awareness is achieved by creating two groups corresponding to the on-premise an off-premise VMs and assigning each HDFS DataNode to the appropriate group.

B. Overview

We run extensive experiments with two real-life MapReduce iterative applications, each described in greater detail in Section VI-D and Section VI-E. Both applications exhibit a reduction phase that involves a negligible amount of data compared with the map phase, which is a frequent real-life scenario that emphasizes the map phase. The goal of these experiments is two-fold: (1) to demonstrate the feasibility of our re-balancing proposal; (2) to validate the hybrid performance prediction model introduced in Section V against the results observed in real life.

First, we run a series of I/O intensive benchmarks that correspond to the *micro-calibration* mentioned in Section V-D. To this end, we rely on the *TestDFSIO* micro-benchmark, which is a standard Hadoop tool that measures the HDFS read and write throughput using a predefined number of concurrent readers and writers.

Second, we run a series of experiments that study the strong scalability of the application on a single OpenStack cloud. Since there is no weak link in this setup, these experiments reveal the maximum theoretical potential for speed-up in a hybrid setup. We refer to this series of experiments as *Baseline*. Then, we run the same experiments in a hybrid setup, where we fix the number of on-premise VMs and vary the number of off-premise VMs. We refer to these experiments as *Hybrid-real*. We discuss these results in comparison with *Baseline* to address goal (1).

Forth, based on the results from *Baseline* experiments and the *micro-calibration*, we extract the relevant application metrics and compute the hybrid rebalancing factors to estimate T_M^{low} and T_M^{up} using the equations described in Section V-C. Since the reduce phase is negligible compared with the map phase, $T^{low} \approx T_M^{low}$ and $T^{up} \approx T_M^{up}$. We then discuss these results in relationship with *Hybrid-real* to address goal (2).

C. Micro-calibration

In this section we illustrate how to perform the micro-calibration. To demonstrate how to reuse the results of the micro-benchmarking for multiple applications, we fix the application input data at 20 GB and the HDFS chunk size (corresponding to the size of data per mapper D_M) at 64 MB, which means a total of 300 map tasks (p_M) are needed.

First, the data is generated by running *TestDFSIO* in write mode in an HDFS deployment spanning 3 on-premise VM instances. After the initial data was written, the HDFS deployment is extended by a variable number of additional off-premise VM instances. Then, the hybrid rebalancing is started at the same time with another *TestDFSIO* that runs this time in read mode using a number of concurrent readers equal to the number of VM instances. Meanwhile, the rebalancing moves at least one replica for each chunk of data from the on-premise to the off-premise nodes. While the experiment is running, we monitor the amount of data that accumulates off-premise during the rebalancing. We record both this progress and the metrics reported by *TestDFSIO*, which is run repeatedly in an iterative fashion until the rebalancing is complete.

The results of the rebalancing progress are depicted in Figure 2. As can be observed, the off-premise HDFS data accumulates steadily in all configurations. Furthermore, there is little difference between the various hybrid configurations, which enables an estimation of $B_M(t)$ (introduced in Section VI-C) even when micro-benchmarking results are not available for a particular configuration.

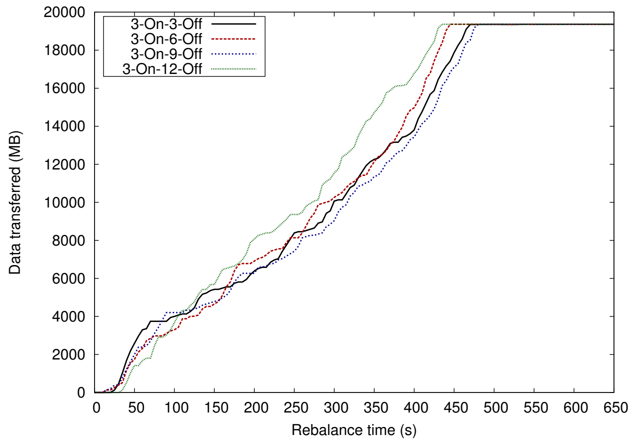


Fig. 2. TestDFIO micro-calibration: rebalance progress for 20 GB total data

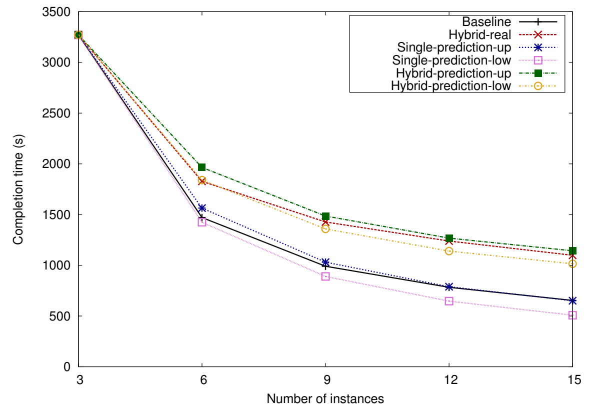
TABLE I
TestDFSIO AVERAGE COMPLETION TIME PER ITERATION

Configuration	Time / iteration	Alpha
3-on-0-off	276s	N/A
3-on-3-off	472s	1.70
3-on-6-off	471s	1.70
3-on-9-off	485s	1.75
3-on-12-off	416s	1.5

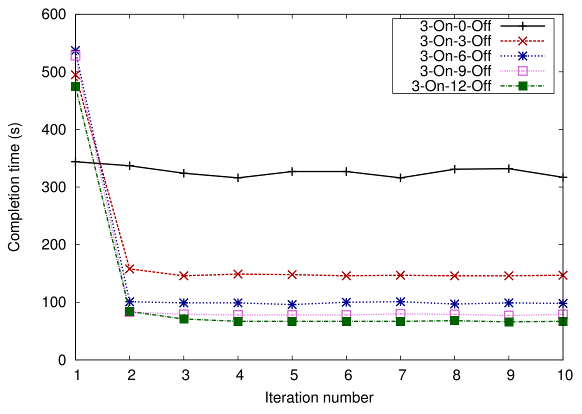
The average completion time per concurrent read iteration for *TestDFSIO* is illustrated in Table I. By convention, we denote a configuration with N on-premise VMs and M off-premise VMs as $\mathsf{N{-}on{-}M{-}off}$. $\mathsf{3{-}on{-}0{-}off}$ is the baseline for which no re-balancing is present. Both the baseline and the hybrid *TestDFSIO* experiments are repeated 10 times. These results are then used to calculate α, included in the table. As can be observed, the re-balancing introduces significant background overhead that reduces the concurrent read throughput and lowers the overall completion time per iteration by up to 75%. Also, interesting to observe is that α remains very close for all hybrid configuration except $\mathsf{3{-}on{-}12{-}off}$. Thus, the previous observation about a rough estimation being possible even when no micro-benchmarks are available for a particular configuration holds for α too.

D. KMeans

Our next series of experiments focus on *K-Means* [27], a widely used application in a multitude of contexts: vector quantization in signal processing, cluster analysis in data mining, pattern classification and feature extraction for machine learning, etc. K-Means partitions a set of multi-dimensional vectors into k sets, such that the sum of squares of distances between all vectors from the same set and their mean is minimized. This is typically done by using iterative refinement: at each step the new means are calculated based on the results from the previous iteration, until they remain unchanged (with respect to a small epsilon). K-Means was shown to be efficiently parallelizable and scales well using



(a) Strong scalability: predicted vs. real total completion time for 10 iterations for a single cloud and a hybrid cloud setup. The measured completion time observed on the single cloud is the *Baseline*. Lower is better.



(b) Iteration analysis: completion time per iteration for an increasing number of off-premise VM instances. Lower is better.

Fig. 3. K-Means: iterative clustering (10 iterations) of a 20 GB dataset.

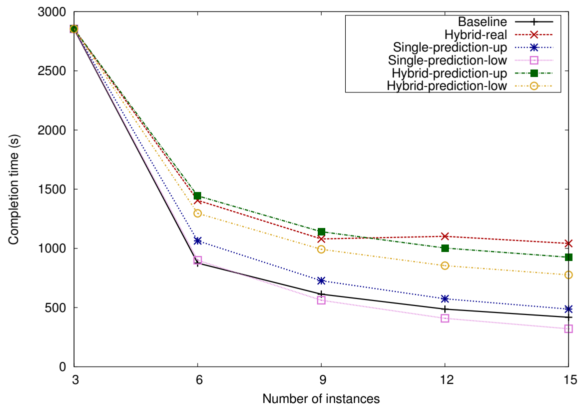
MapReduce [28], which makes it a popular tool to analyze large quantities of data at large scale.

For the purpose of this work, we use the MapReduce K-Means implementation that is part of the Mahout 0.10 collection of machine learning algorithms. This implementation generates only a minimal amount of intermediate data at each iteration (i.e., the mean for each of the k sets), however it typically analyses a large amount of input data that remains unchanged between the iterations. Thus, it is classified as a map-intensive job. We generate 20 GB worth of input that is processed in 10 iterations. The data is generated using the data generator included in Mahout and is uploaded to HDFS before starting each experiment. For comparison, the shuffle data for each iteration is in the order of several MB, which is why we can consider the reduction phase negligible (i.e., $T^{low} \approx T_M^{low}$ and $T^{up} \approx T_M^{up}$).

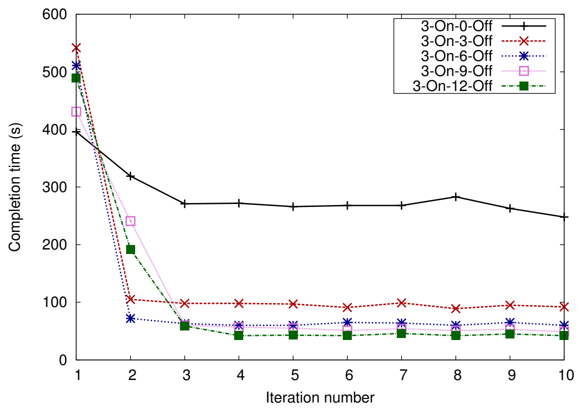
First, we run the *Baseline* experiment by deploying a single OpenStack cloud where we vary the number of VMs allocated to the Hadoop deployment. As can be observed in Figure 3(a), with an increasing size of the Hadoop deployment, K-Means experiences a steady drop in the total completion time, confirming its potential to achieve strong scalability. Furthermore, applying the performance model for on-premise jobs introduced in Section reveals a good estimation of the total completion time: the *Baseline* stays within the lower (*Single-prediction-low*) and upper (*Single-prediction-up*) prediction bounds at all times. Furthermore, there is almost a perfect overlap between *Single-prediction-up* and *Baseline*, while *Single-prediction-low* provides an over-optimistic estimation that deviates by at most 20%.

Next, we run the *Hybrid-real* experiment, where we deploy a hybrid setup consisting of 3 on-premise VMs and a variable number of off-premise VMs (X axis depicts total number of VMs). Initially, Hadoop is deployed only on the on-premise VMs and is extended as described in the Section VI-A, with the asynchronous rebalancing and the application being started simultaneously. The total completion time can be observed in Figure 3(a). Interesting to note is the drop in completion time with increasing number of off-premise VMs. As expected, the rebalancing overhead in the hybrid case has a negative impact on the strong scalability when compared with *Baseline* (up to of 40% increase in execution time), however the scalability trend is clearly visible, confirming the viability of adopting our proposal to extend iterative MapReduce jobs using additional VMs leased from an off-premise cloud. Furthermore, by using the relevant application metrics extracted from the *Baseline* experiments (i.e., A_M, λ) and the micro-calibration results from Section VI-C in the equations described in Section V-C, we obtain the lower (*Hybrid-prediction-low*) and upper (*Hybrid-prediction-up*) total estimated hybrid completion time. As can be observed, we can see again a good prediction: the real result stays within the lower and upper bound, while the error is at most 8% for the lower bound and 4% for the upper bound.

To understand these results better, we zoom in Figure 3(b) on the completion time per iteration. We use the same N−on−M−off notation for each configuration as explained in Section VI-C. As expected, for the 3−on−0−off case, the completion time per iteration remains constant. However, in the 3−on−M−off cases, a large gap between the first and the rest of the iterations is visible. This is explained by the fact that the re-balancing finishes during the first iteration, such that beginning with the second iteration, the data locality can be fully exploited. Since the invariant input data is reused at each iteration, most of the increase in the total completion time is due to the first iteration. As a consequence, the more iterations are needed during the computation, the better this initial overhead will be amortized.



(a) Strong scalability: predicted vs. real total completion time for 10 iterations for a single cloud and a hybrid cloud setup. The measured completion time observed on the single cloud is the *Baseline*. Lower is better.



(b) Iteration analysis: completion time per iteration for an increasing number of off-premise VM instances. Lower is better.

Fig. 4. IGrep: iterative text analysis (10 iterations) of a 20 GB dataset.

E. Iterative Grep

The second application we evaluate is *iterative grep (IGrep)*, which is a popular analytics tool for large unstructured text. Iterative grep consists of a set of independent grep jobs that find all string matches of a given regular expression and sorts them according to the number of matches. The iterative nature is exhibited in the fact that the input data remains the same, but the regular expression changes as a refinement of the previous iteration. For example, one may want to count how many times a certain concept is present in the Wikipedia articles, and, depending on the result, prepare the next regular expression in order to find correlations with another concept. Since the regular expression is typically an exact pattern, the output of the mappers is very simple and consists of a small number of key-value pairs that are reduced to a single key-value pair. Thus, it can be classified as a typical map-intensive MapReduce job.

For the purpose of this work, we use the standard *grep* implementation that comes with the Hadoop distribution. We use 20 GB worth of Wikipedia articles as input data and 10 keywords to run 10 iterations over this input data, which is uploaded to HDFS before each experiment. The shuffle data for each iteration is less than one MB, which is why we can consider the reduction phase negligible (i.e., $T^{low} \approx T_M^{low}$ and $T^{up} \approx T_M^{up}$).

As can be observed in Figure 4(a), for the *Baseline* experiment (measured total completion time for a single cloud), there is again evidence of strong scalability. This is understandable, since grep is almost embarrassingly parallel. However, there is a slight degradation of scalability for an increasing number of VMs, due to the increasing overhead of parallelization. Applying the performance model for on-premise jobs (Section VI-D), we observe the following estimations for the total completion time: the lower bound (*Single-prediction-low*) under-estimates by up to 24% and the upper bound over-estimates by up to 15%, which places the measured total completion time within the lower and upper bound.

For the *Hybrid-real* experiment, we deploy a hybrid setup that keeps 3 on-premise VMs and adds a variable number of off-premise VMs. The total completion time can be observed in Figure 4(a) (total number of VMs on X axis). Again, we observe a drop in completion time with increasing number of off-premise VMs, which confirms the viability of adopting our re-balancing proposal. Furthermore, the lower (*Hybrid-prediction-low*) and upper (*Hybrid-prediction-up*) estimated hybrid completion time keep the measured result within their limits up until 6 off-premise VMs. However, when increasing the number of off-premise VMs beyond 6, both the lower and upper bound under-estimate the measured completion time: by up to 25% and 12% respectively.

These results are better understood by analyzing the per iteration completion times, which are depicted in Figure 4(b). Surprisingly, for the 3−on−0−off case, the completion time per iteration remains constant only after a few iterations, which hints at possible OS caching effects for the input data read from HDFS at each iteration. In the hybrid configurations, it can be observed that the rebalancing does not finish during the first iteration for the case when a large number of off-premise VMs is used (i.e. 9 and 12). While the hybrid estimations exhibit larger errors than in the case of K-Means, so do the single cloud estimations, which means that the hybrid aspect was accurately captured but the application is inherently more unpredictable due to its increased complexity.

VII. Conclusions

Hybrid clouds open an entire new horizon in the big data analytics landscape, effectively enabling on-premise resource owners to extend complex workloads beyond the capacity of their infrastructure by leasing off-premise resources. However, the need to transfer large data sizes off-premise poses a difficult challenge to the ability to exploit data locality efficiently.

This paper contributed with a novel proposal that addresses this challenge for iterative MapReduce applications. It transparently manages data movements asynchronously in an efficient fashion without invasive changes to the MapReduce framework or the underlying storage layer. At the same time, it is able to predict the runtime of the application for a variety of hybrid configurations, by combining analytical modeling with micro-calibration. Results using two real-life iterative MapReduce applications show excellent hybrid scalability potential that follows a similar trend as the single-site scalability except for an initial overhead during the first few iterations, whose impact on the overall execution time is diminished with increasing number of iterations. Furthermore, our prediction of the execution time for a hybrid setup matches the accuracy of the techniques used in single-site setups, with maximum upper/lower bound errors of 4%/8% and, respectively, 12%/25%.

Encouraged by these results, we plan to broaden the scope of our work in future efforts. In particular, we focused on map-intensive applications where the reduce phase is negligible in comparison. Thus, one interesting direction is to complement the current work with an analysis of reduce-intensive jobs in a hybrid setup: study of the weak link and interferences with the rebalancing, refined prediction equations, etc.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010.

[2] P. Hofmann and D. Woods, "Cloud computing: The limits of public clouds for business applications," *Internet Computing, IEEE*, vol. 14, no. 6, pp. 90–93, Nov 2010.

[3] T. Guo, U. Sharma, T. Wood, S. Sahu, and P. Shenoy, "Seagull: Intelligent cloud bursting for enterprise applications," in *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, ser. USENIX ATC'12, Berkeley, CA, USA, 2012, pp. 33–33.

[4] B. Javadi, J. Abawajy, and R. Buyya, "Failure-aware resource provisioning for hybrid cloud infrastructure," *J. Parallel Distrib. Comput.*, vol. 72, no. 10, pp. 1318–1331, Oct. 2012.

[5] "Hybrid cloud market: Forecasts and analysis (2013 2018)," Market Research Insight, Markets and Markets, 2013.

[6] H. Zhang, G. Jiang, K. Yoshihira, H. Chen, and A. Saxena, "Intelligent workload factoring for a hybrid cloud computing model," in *Services - I, 2009 World Conference on*, July 2009, pp. 701–708.

[7] C.-H. Suen, M. Kirchberg, and B. S. Lee, "Efficient migration of virtual machines between public and private cloud," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, Nov 2011, pp. 549–553.

[8] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[9] W.-T. Tsai, P. Zhong, J. Elston, X. Bai, and Y. Chen, "Service replication strategies with mapreduce in clouds," in *ISADS '11: 10th International Symposium on Autonomous Decentralized Systems*, Kobe, Japan, 2011, pp. 381–388.

[10] F. Tian and K. Chen, "Towards optimal resource provisioning for running mapreduce programs in public clouds," in *CLOUD '11: 2011 IEEE International Conference on Cloud Computing*, Washington DC, USA, 2011, pp. 155–162.

[11] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "Mapreduce in the clouds for science," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 565–572.

[12] X. Zhang, L. T. Yang, C. Liu, and J. Chen, "A scalable two-phase top-down specialization approach for data anonymization using mapreduce on cloud," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 25, no. 2, pp. 363–373, 2014.

[13] B. Nicolae, P. Riteau, and K. Keahey, "Bursting the Cloud Data Bubble: Towards Transparent Storage Elasticity in IaaS Clouds," in *IPDPS '14: Proc. 28th IEEE International Parallel and Distributed Processing Symposium*, Phoenix, USA, 2014, pp. 135–144.

[14] ——, "Transparent Throughput Elasticity for IaaS Cloud Storage Using Guest-Side Block-Level Caching," in *UCC'14: 7th IEEE/ACM International Conference on Utility and Cloud Computing*, London, UK, 2014.

[15] B. Sharma, T. Wood, and C. R. Das, "Hybridmr: A hierarchical mapreduce scheduler for hybrid data centers," in *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*. IEEE, 2013, pp. 102–111.

[16] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin, "Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 922–933, 2009.

[17] K. Shirahata, H. Sato, and S. Matsuoka, "Hybrid map task scheduling for gpu-based heterogeneous clusters," in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*. IEEE, 2010, pp. 733–740.

[18] M. M. Rafique, A. R. Butt, and D. S. Nikolopoulos, "A capabilities-aware framework for using computational accelerators in data-intensive computing," *J. Parallel Distrib. Comput.*, vol. 71, no. 2, pp. 185–197, 2011.

[19] M. M. Rafique, B. Rose, A. R. Butt, and D. S. Nikolopoulos, "Cellmr: A framework for supporting mapreduce on asymmetric cell-based clusters," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE, 2009, pp. 1–12.

[20] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads," in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, July 2010, pp. 228–235.

[21] S. Imai, T. Chestna, and C. A. Varela, "Accurate resource prediction for hybrid iaas clouds using workload-tailored elastic compute units," in *IEEE/ACM 6th International Conference on Utility and Cloud Computing, UCC 2013, Dresden, Germany, December 9-12, 2013*, 2013, pp. 171–178.

[22] M. Mattess, R. Calheiros, and R. Buyya, "Scaling mapreduce applications across hybrid clouds to meet soft deadlines," in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, March 2013, pp. 629–636.

[23] T. Bicer, D. Chiu, and G. Agrawal, "A framework for data-intensive computing with cloud bursting," in *CLUSTER '11: The 2011 IEEE International Conference on Cluster Computing*, 2011, pp. 169–177.

[24] H. Zhang, G. Jiang, K. Yoshihira, and H. Chen, "Proactive workload management in hybrid cloud computing," *Network and Service Management, IEEE Transactions on*, vol. 11, no. 1, pp. 90–100, 2014.

[25] K. Shvachko, H. Huang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *26th IEEE (MSST2010) Symposium on Massive Storage Systems and Technologies*, May 2010.

[26] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: Automatic Resource Inference and Allocation for Mapreduce Environments," in *ICAC '11: The 8th ACM International Conference on Autonomic Computing*, Karlsruhe, Germany, 2011, pp. 235–244.

[27] H.-H. Bock, "Clustering methods: A history of K-Means algorithms," in *Selected Contributions in Data Analysis and Classification*, ser. Studies in Classification, Data Analysis, and Knowledge Organization. Springer Berlin Heidelberg, 2007, pp. 161–172.

[28] W. Zhao, H. Ma, and Q. He, "Parallel K-Means clustering based on MapReduce," in *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, Beijing, China, 2009, pp. 674–679.