Performance Model of MapReduce Iterative Applications for Hybrid Cloud Bursting

Francisco J. Clemente-Castelló*, Bogdan Nicolae†,
Rafael Mayo*, Juan Carlos Fernández*

*Universitat Jaume I, Spain
Email: {fclement, mayo, jfernand}@uji.es
†Huawei Research Germany
Email: bogdan.nicolae@acm.org

Abstract—Hybrid cloud bursting (i.e., leasing temporary off-premise cloud resources to boost the overall capacity during peak utilization) can be a cost-effective way to deal with the increasing complexity of big data analytics, especially for iterative applications. However, the low throughput, high latency network link between the on-premise and off-premise resources ("weak link") makes it difficult to maintain scalability. While there are several data locality techniques dedicated for big data bursting on hybrid clouds, their effectiveness is difficult to estimate in advance. On the other hand, such estimations are critical for users, because they aid in the decision of whether the extra pay-as-you-go cost incurred by using the off-premise resources justifies the runtime speed-up. To this end, the current paper contributes with a performance model and methodology to estimate the runtime of iterative MapReduce applications in a hybrid cloud bursting scenario. A key idea of the proposal is to focus on the overhead incurred by the weak link at fine granularity, both for the map and reduce phase. This enables high estimation accuracy, as demonstrated by extensive experiments at scale using a mix of real-life iterative MapReduce applications from standard big data benchmarking suites that cover a broad spectrum of data patterns. Not only are the produced estimations accurate in absolute terms compared with the actual experimental results, but they are also up to an order of magnitude more accurate than applying state-of-art estimation approaches originally designed for single-site MapReduce deployments.

Index Terms—Hybrid Cloud; Big Data Analytics; Iterative Applications; MapReduce; Performance Prediction; Runtime Estimation

1 Introduction

An important class of problems running on private clouds is *big data analytics*. However, with data sizes exploding (Zettabytes predicted by 2020 [1]) and applications becoming increasingly complex, private clouds struggle to accommodate the required scale and scope: there is often simply not enough capacity to run the desired analytics or it is difficult to obtain the desired results within a given deadline. In addition, the rich, shared big data ecosystem facilitated by public cloud computing (large amounts of data exploitable from multiple data sources and users) opens many new opportunities for combined analytics that potentially enables new insight beyond what is possible within the scope of a private cloud alone. In this context, *cloud bursting* [2] has seen a rapid increase in popularity among big data analytics users. It is a form of *hybrid cloud computing* that enables temporary boosting of on-premise resources managed by a private cloud with additional off-premise resources from a public cloud provider, for the purpose of overcoming the limitations of private data centers only when necessary (e.g. during peak utilization) in a flexible, cost-efficient pay-as-you-go fashion.

However, enabling cloud bursting for big data analytics at large scale poses a major challenge: unlike conventional datacenters where big data analytics applications and middleware run on top of physically co-located IT resources with high-speed interconnections, the use of both on-premise and off-premise resources creates a "weak link" between them that is at least an order of magnitude slower. This weak link becomes a major bottleneck in the context of big data analytics, because massive data sizes need to be shipped back and forth between the on-premise and the off-premise part as a result of complex concurrent data access patterns that are not easy to predict. This effect has multiple implications at the level of the runtime and storage layer, prompting the need for new "hybrid cloud big data analytics" approaches.

One particular class of big data analytics applications are particularly well suited for cloud bursting: iterative applications that reuse invariant input data. In this case, a large part of the data needs to be shipped over the weak link only once and can be reused for subsequent iterations, which potentially reduces the I/O pressure on the weak link and thus diminishes its negative effect on the overall performance. Furthermore, iterative applications refine the result progressively with each iteration, which means intermediate results are accessible while the computation is still on-going. This has important consequences in the context of cloud bursting, because it facilitates early decisions, e.g. stop when the result is good enough and does not justify extra cost for refinement or guide other computations in a

complex workflow that depends on the results.

Accelerating iterative applications using hybrid cloud bursting is non-trivial, because it raises several issues: how to ship the initial data, how to overlap the computations with the data transfers and how to exploit data locality efficiently.

In our previous work [3], [4], we introduced two complementary techniques to address these issues: (1) extended off-premise HDFS storage layer using asynchronous rack-aware replica rebalancing, (2) locality-enforced scheduling that avoids redundant data transfers over the weak link. Thanks to these techniques, we have shown that iterative MapReduce applications running in a cloud bursting scenario can experience significant speed-up compared with the default Hadoop implementation designed for a single data-center. In addition, we have also shown that such techniques can perform close to the lower bound, i.e. the performance is close to the case when more on-premise resources are added to match the capacity of the hybrid setup.

However, despite potential to achieve significant speed-up, an equally important challenge is to *estimate the runtime* in advance: the extra off-premise resources provisioned through hybrid cloud bursting incur pay-as-you-go costs, thus advance knowledge about potential speed-up aid the user to decide *before* committing any money whether it is worthwhile to use hybrid cloud bursting at all, and, if so, how many off-premise resources are optimally needed to achieve a desired performance-cost trade-off.

In this paper we extend our previous work with a generic performance model that applies to the complementary techniques introduced above in order to estimate the runtime of iterative MapReduce applications for hybrid cloud bursting. We summarize our contributions as follows:

- We elaborate on the fundamental issues in operating iterative MapReduce over hybrid cloud setups comprising both on- and off-premise virtual machines (VMs). In particular, we further develop the main issue discussed in our previous work (lacking data locality on the off-premise part and associated consequences) from multiple angles: I/O interactions with the underlying storage layer, task scheduling and data shuffling (Section 3 and Section 4).
- We propose a methodology that combines analytical modeling with synthetic benchmarking to estimate the time-to-solution in a hybrid setup, including all fine-grain overhead associated with the map phase and the reduce phase (shuffle, sort, reduce). This model extends our preliminary efforts in this direction [4] that address the map phase only (Section 5).
- We evaluate our approach in a series of experiments that involve four representative real-life iterative MapReduce applications from standardized big data benchmarks that cover a broad range of use cases. Our experiments demonstrate small errors between the runtime estimations and the actual measured values, which are up to an order of magnitude smaller than using state-of-art MapReduce runtime estimation approaches designed for single site setups (Section 6).

2 Related work

MapReduce applications have been studied extensively on single cloud computing platforms [5], [6]. Storage elasticity [7], [8] is a particularly interesting aspect for iterative applications, as it is an important component of the overall pay-as-you-go cost. Data shuffling is another difficult problem even in a single data center [9].

The topic of performance and cost prediction in a single data center was studied from multiple angles. A storage performance and cost model for iterative applications was introduced in [10]. Given the data-intensive nature of MapReduce applications and the need to persist data between jobs, such a direction is an important complement to our own work. MapReduce performance modeling in particular has focused on various aspects: scheduling, resource provisioning, performance and cost estimation.

Tian et al. [11] proposed a cost model that estimates the performance of a job from a set of test runs on a small input datasets and small number of nodes. The authors provision the resources for the job using a simple regression technique. Chen et al. [12] further improved the cost model and proposed CRESP which employs the brute-force search technique for provisioning the optimal cluster resources in term of map slots and reduce slots for Hadoop jobs. However, in the two models, the number of reduce tasks have to be equal to the number of reduce slots which means that these two models only consider a single wave of the reduce phase.

Lama et al. [13] proposed AROMA, a system that automatically obtains the optimal resources and optimizes the configuration parameters of Hadoop for a job to achieve the objectives. However, AROMA does not provide a comprehensive mathematical model to estimate the job execution time as well as optimal configuration parameter values of Hadoop. There are a few other models those they use the previous executed job profiles for performance prediction. Starfish [14] applies dynamic Java instrumentation to collect the past executed jobs profile information at a fine granularity for job estimation and automatic optimization. However, collecting a large set of metrics generate an extra overhead, especially for CPU-intensive applications.

Verma et al. [15] presented the ARIA, a MapReduce analytic performance model that computes the lower and upper bounds on the job execution time. The HP model [16] extends the ARIA mode by adding scaling factors to estimate the job execution time on larger datasets using a simple linear regression. The work presented in [17] divides the map phase and reduce phase into six generic sub-phases (i.e. read, collect, spill, merge, shuffle and write), and uses a regression technique to estimate the duration of these sub-phases. The estimated values are then used in the analytical model presented in [15] to estimate the overall job execution time. The same bound-based approach [15] is applied by Zhang et al. [18] to heterogeneous Hadoop cluster environments. Several other works show interest [19]–[22] in heterogeneous MapReduce environments.

To our best knowledge, we are the first to introduce a performance model that specializes both on *hybrid cloud bursting* and *iterative MapReduce applications*.

3 Data Locality Challenges under Hybrid Cloud Bursting

The MapReduce paradigm is specifically designed to facilitate a high degree of data parallelism: in the first stage (*map* phase), massive amounts of input data are read from a storage layer (typically a distributed file system like HDFS [23]) and transformed in an embarrassingly parallel fashion by mapper processes such that an intermediate output (consisting of key-value pairs) is obtained that is sorted by key. Then, in a second stage (*shuffle phase*), a separate set of reducer processes are responsible to fetch (in parallel) the data corresponding to individual keys from all mappers and to merge it. Finally, in a third stage (*reduce phase*), the reducers apply an aggregation over the values merged under the same key to obtain the final result (one value per key), which is typically persisted in the storage layer.

Both the mappers and reducers are distributed processes that exhibit highly concurrent I/O intensive data access patterns, which can overwhelm the networking infrastructure with inter-node data transfers. To address this issue, *data locality* awareness is a key feature of MapReduce: the storage layer is co-located with the runtime on the same nodes and is designed to expose the location of the data blocks, effectively enabling the scheduler to bring the computation close to the data and to avoid a majority of the storage-related network traffic.

In Figure 1a we illustrate a hybrid cloud bursting scenario. In this case, the premises for leveraging data locality are different: the input data is present only on the on-premise VMs initially, so it has to be shipped to the off-premise VMs before they can contribute to the computation. Furthermore, the link between the on-premise infrastructure and the external cloud provider is typically of limited capacity (i.e., a *weak link*). Thus, off-premise VMs that need to communicate with on-premise VMs create a network bottleneck much faster than in the case where all VMs are located within the same datacenter. Specifically, the weak link affects MapReduce applications in the following fashion:

Map phase: since the input data is present initially only on the on-premise VMs, any map task that is scheduled off-premise needs to access the on-premise data, which involves a data transfer over the weak link. Furthermore, all off-premise mappers are running in parallel and thus compete for the weak link, which introduces high I/O pressure on it.

Shuffle phase: each reduce task needs to collect the intermediate data generated by the map tasks: if r reduce tasks collect the intermediate data from m map tasks, an *m-to-r* concurrent communication is required during this phase. Therefore, the weak link will be stressed by all communication required between the on-premise maps and the off-premises reduces, and between the off-premise maps and the on-premise reduces, as can be seen in Figure 1b.

Reduce phase: once the reduce tasks have finished pulling the intermediate data and have performed the aggregation, they typically need to persist the results on-premise. Again, this involves data transfers from the off-premise VMs to the on-premise VMs over the weak link, which puts I/O pressure on it.

In the context of iterative MapReduce applications, the impact of the weak link accumulates as each iteration needs to go through all three phases. However, the iterations are not independent MapReduce jobs: they share a large part of the initial input data. Thus, it is important to leverage this particular aspect in order to reduce the pressure on the weak link.

4 Techniques to Leverage Data Locality for Iterative MapReduce

In this section we present two complementary techniques to improve data locality for hybrid cloud bursting. These techniques were introduced by our previous work [3], [4].

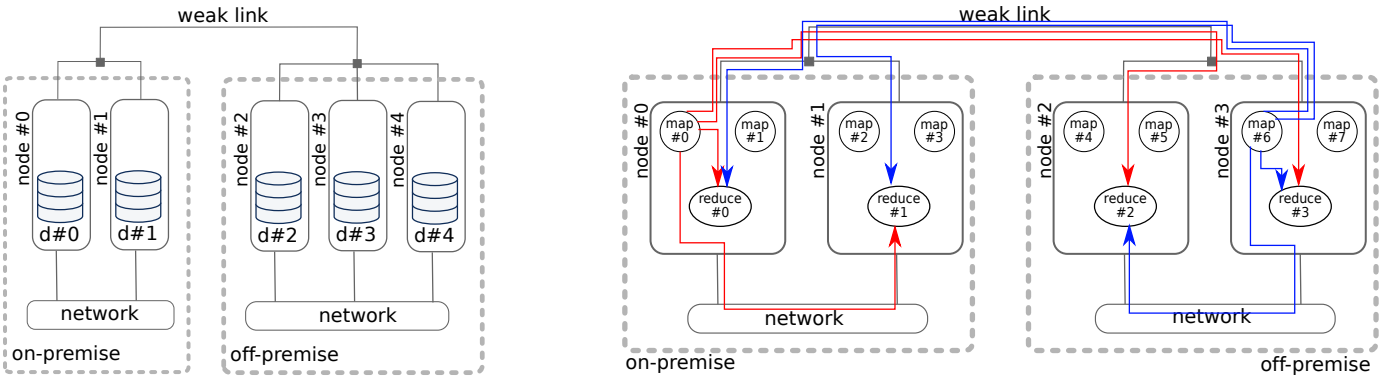
4.1 Off-Premise Replication using Rack-Aware Rebalancing

An obvious choice is to simply leave the input data on-premise and pull it on-demand from the off-premise map tasks. This approach has two advantages: (1) it works out of the box with no modification necessary to the default runtime; (2) it overlaps the I/O with the computation, as map tasks can start off-premise right away without any need to wait for data transfers. On the other hand, there is also a major disadvantage: if the iterative application reuses the input data blocks, they will be transferred over the weak link multiple times unnecessarily, which leads to performance degradation.

Another choice is to replicate the input data off-premise before running the MapReduce application. Using this approach, the MapReduce runtime can fully exploit locality as if it were running on a single cluster, which also avoids the problem of sending the same input block over the weak link repeatedly. However, creating the replicas off-premise before running the application (i.e. *synchronous* replication) is a time-consuming process that adds to the overall completion time. Furthermore, it also leads to an extra off-premise storage space utilization.

To avoid waiting for the replication process to finish, a third option is possible: to use *asynchronous* replication. Using this approach, the input data is shipped off-premise at the same time as the map tasks are running, under the assumption that a majority map tasks will benefit from the data locality. However, the background data transfers create additional overhead that interferes with the computation and may create enough slowdown to offset the benefit of starting the computation right away.

We have shown that asynchronous replication can be efficiently achieved using *rack awareness*, a core feature of *HDFS* [23], the default storage layer of Hadoop. Specifically, data blocks are replicated in HDFS (three times by default) for resilience purposes, with at least one replica in a different rack than the one where the write originated. Furthermore, HDFS also can rebalance the replicas across the storage elements to distribute the load evenly while preserving rack awareness. Thus, by deploying new HDFS storage elements on the off-premise VMs as a separate rack, a rebalancing operation will migrate one replica for each data block to the off-premise VMs asynchronously.



(a) Infrastructure: The weak link interconnects the on-premise VMs with the off-premise VMs

(b) MapReduce shuffle phase: Concurrent data transfers put I/O pressure on the weak link. For clarity, only data transfers from one on-premise map and one off-premise map are depicted.

Fig. 1: Schematic representation of a hybrid cloud bursting architecture and its implications on MapReduce applications

The main advantage of this approach is that it minimizes the amount of data transferred off-premise (a single replica) to achieve full potential of exploiting the data locality, while maintaining the resilience constraints. Moreover, it is a non-invasive solution that works out-of-the box without deviating from the standard HDFS, which is a major concern for many users.

4.2 Scheduling Based on Enforced Rack-Locality

The default Hadoop scheduler uses data locality only as a preferential matching mechanism between map tasks and free slots. However, if asynchronous HDFS rebalancing is employed, then a map task may be scheduled off-premise before a replica of its corresponding data block was migrated, which triggers a pull and leads to a double data transfer of the same block. In this case, it is beneficial to delay the scheduling of such off-premise map tasks, under the assumption that avoiding stress on the weak link leads to a a smaller overall overhead. To this end, we propose an enforced rack-locality scheduling policy: a map task will never be scheduled off-rack if no replica of the data block is present in the other rack. This effectively leads to the desired behavior in our case: a map task will never be scheduled off-premise if a replica was not already migrated there.

We implemented this policy in Hadoop by modifying the Resource Manager to make use of the *relaxLocality* flag. Thus, unlike the HDFS replica rebalancing, this is an intrusive modification to Hadoop that requires the user to deploy a custom version.

5 Performance Modeling Proposal

In this section we introduce the broad principles and methodology behind our performance modeling proposal. This model is specifically targeted at a MapReduce runtime that makes use of the two techniques introduced in Section 4 to accelerate iterative applications in a hybrid cloud bursting scenario. Our goal is to estimate the completion time of a given iterative MapReduce application as a mathematical expression based on a series of system-level and application-level parameters that are extracted in advance.

For simplification, we assume the setup consists of a fixed set of on-premise VMs and off-premise VMs of similar capability, which gives us a fixed set of system-level parameters. Users interested in estimating the runtime for various setups (e.g., find out the optimal number of off-premise VMs to achieve the desired runtime) can apply our approach for each configuration individually.

Our approach consists of three steps. First, we run a synthetic benchmark to extract the fixed system-level parameters corresponding to the on-premise and off-premise VMs. These parameters are *independent of the application* and can be reused for a different application or user (e.g., they can be cached on-premise). We refer to this step as *calibration*.

Second, we extract the application-level parameters. These parameters are *independent of the hybrid setup* and can be either known in advance or obtained by running the application at smaller scale on-premise only. This way, users can estimate the benefits of hybrid cloud bursting without actually ever trying it, as long as the calibration step was already performed for the desired configuration. We refer to this step as *characterization*.

Finally, once both the calibration and the characterization is complete, we apply a mathematical expression to estimate the completion time. Note that there is an inherent variability in the approximations introduced above due to the complexity of MapReduce applications in general and the additional complexity introduced by the weak link. Therefore, it is important to be able to present both the optimistic (*lower bound*) and pessimistic (*upper bound*) runtime estimation to the user. To derive the mathematical expressions for both cases, we make use of the makespan theory as applied in the context of MapReduce.

To aid the extraction of both system-level and application-level parameters, we have developed a tool that can analyze a job in terms of map/shuffle/reduce times, HDFS data distribution, task distribution between on-premise and off-premise and even node statistics such as CPU, I/O network, I/O disk and memory utilization. It extracts information from a combination of Hadoop counters, Hadoop logs, Hadoop Rumen tool [24] and Systat [25] to generate a profiling information.

For the rest of this section, we detail all aspects summarized above.

5.1 Theoretical Makespan Bounds for MapReduce

For completeness, we briefly introduce in this section the theoretical makespan bounds as applied in the context of MapReduce. This is needed as a preliminary background in order to understand the reasoning behind the parameters extracted from the calibration and characterization steps.

Each of the map and reduce phases can be abstracted as a series of n tasks of duration t_i that need to be processed by k slots. The assignment of tasks to slots is done dynamically by the MapReduce runtime according to a simple greedy policy: assign each task to the slot with the earliest finishing time.

The best case is obtained when all slots are evenly loaded, in which case each slot is busy for at least $(\sum_{i=1}^{n} t_i)/k = (t_{avg} \times n)/k$. Therefore, this is the lower bound of the makespan.

The worst case is obtained when the longest task t_{max} is scheduled last. This means the k slots are busy with the other tasks and take at most $(t_{avg} \times (n-1))/k$ to process them. Once they finish with these tasks, one of them finally needs to process t_{max}. Therefore, the upper bound of the makespan is $(t_{avg} \times (n-1))/k + t_{max}$.

It is important to note that the lower bound can be expressed more precisely as a function of the average task duration (rather than the minimum). As a consequence, several of the parameters we extract from the calibration and characterization steps are averages and maximum values.

5.2 Calibration Using Synthetic Benchmarking

We develop a synthetic benchmark that focuses on the extraction of the I/O and communication overheads in a hybrid setup. Specifically, the goal is to extract these overheads for each phase (map, shuffle, reduce) based on the quantity of data involved: (1) the amount of data read from HDFS; (2) the amount of data written to HDFS and (3) the amount of network traffic between map and reduce tasks. Using this approach, all possible combinations of data sizes in each phase can be covered.

To achieve this goal, we implement a collection of map, combiner and reduce functions that generate a synthetic workload based on a series of configurable input parameters used to specify the amounts of data. Both the map phase and the reduce phase of the synthetic workload deliberately avoid computational overhead (minimal load on the CPU) in order to isolate the I/O and communication overheads. The map phase is structured in two parts: it reads the input chunk in the map function and writes a specified amount of intermediate data in the combiner function. The output of the combiner is grouped by key, with each group collected by the corresponding reducer. The reducers simply writes a predefined amount of data as output in HDFS.

This synthetic benchmark is then executed for a variable quantity of data in a hybrid setup comprising a fixed number of on-premise and off-premise VMs. We call this process *calibration*. We illustrate how this works using an experimental example that is based on a typical hybrid cloud setup (described in more detail in Section 6.1), where we use two representative configurations for the weak link: 100 Mbps and 1 Gbps.

Then, based on the profiling information, we define a series of system-level parameters that quantify the hybrid-specific overheads.

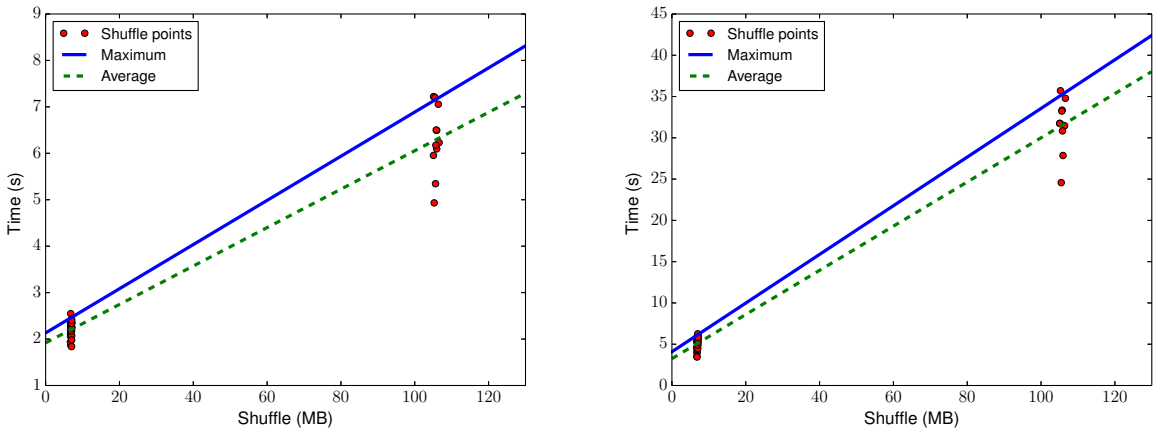
Hybrid map overhead stretch coefficient: Although the scheduler forces the execution of mappers on the nodes where the input data from HDFS is present, there are some extra reads when the logical end of data does not exactly match the HDFS end of a split (i.e. the input of the mapper is covered by two different HDFS chunks). Therefore, mappers will sometimes read remote data over the weak link, which creates an extra overhead compared with the on-premise only case. We measure this overhead using the synthetic benchmark and express it as a stretch coefficient denoted α.

Hybrid approximation of shuffle overhead per reducer: Once a sufficient number of mappers have finished producing the intermediate data, reduce tasks are launched and begin collecting it. However, a reduce task cannot start the aggregation at the same time as the intermediate data is collected, because it needs to sort it first (which cannot happen before all mappers have finished and their intermediate data was collected). Therefore, each reducer experiences a shuffle overhead that is proportional to the amount shuffle data it needs to pull. Since the amount of shuffle data per reducer (denoted d_{Sh}) is application dependent, we express the shuffle overhead as an approximation function (denoted f_{Sh}). To this end, we choose a set of representative shuffle sizes, measure the shuffle overhead per reducer using the synthetic benchmark, and then apply linear regression to obtain two approximation functions: one for the average (using all reducers) and one for the maximum (using only the slowest reducer for each shuffle size).

Figure 2 illustrates this for a hybrid scenario with three on-premise VMs and three off-premise VMs using 24 mappers and 12 reducer slots. The actual runtimes of the reducer tasks are illustrated as clusters of points, while the approximation function (average and maximum) is illustrated as a line.

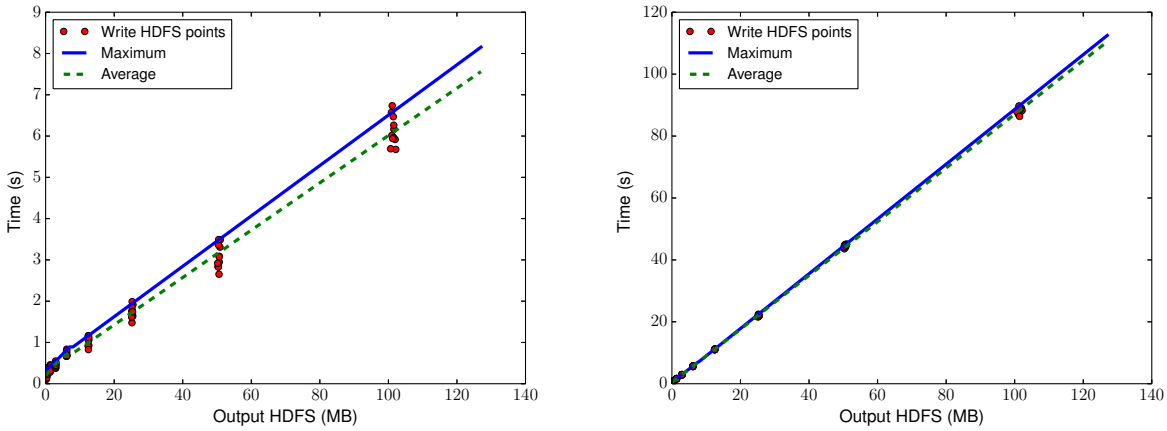
Hybrid approximation of write overhead per reducer: In the case of the reduce phase, the computational overhead of each reducer does not depend on the weak link and represents a significant part of the runtime of the reduce phase. However, once the reducer has finished the computation, it needs to write the output to the storage layer, which stresses the weak link. Again, the amount of output per reducer (denoted d_R) is application dependent. Thus, we need to express it as an approximation function (denoted f_{Rd}) in a manner similar to the shuffle overhead. Using linear regression, we obtain the corresponding approximation function. Figure 3 illustrates this for the same hybrid scenario used above (both average and maximum).

Rebalancing bandwidth: In addition to the overheads related strictly to the MapReduce runtime, it is also important to estimate how long the off-premise rebalancing of the input data will last. This is important because it runs asynchronously in the background while the iterative MapReduce application is progressing, therefore creating interference and potentially visible slowdown. To this end, we run a HDFS rebalancing using a large HDFS input data size (e.g. 10 GB) and measure its completion time. Then,



(a) 1 Gbps inter-cloud network link (b) 100 Mbps inter-cloud network link

Fig. 2: Shuffle approximation in a 3-ON-3-OFF premise hybrid architecture



(a) 1 Gbps inter-cloud network link (b) 100 Mbps inter-cloud network link

Fig. 3: Reduce approximation in a 3-ON-3-OFF premise hybrid architecture

we compute an average bandwidth denoted β that is used in our performance model to account for the effect of the rebalancing.

We summarize all system-level parameters extracted using the calibration step in Table 1.

5.3 Application Characterization

In this section we show how to extract the necessary information to characterize the iterative MapReduce application. Note that this step is only necessary when the application-level parameters introduced below (needed by the mathematical expression) are not known in advance or cannot be directly computed based on some existing knowledge about the application.

Specifically, the user needs to run the application on-premise only at reduced scale (both number of nodes and number of iterations) and extract the following information:

- The total number of mappers: M
- The total number of reducers: R
- The average and maximum time to run a mapper: t_{Mp} and, respectively, t_{Mp}^{max}

TABLE 1: Parameters obtained from the calibration and characterization. To simplify the notation, we use a superindex only for the maximum.

System-level parameters	
Name	Description
S_{on}^{M}	On-premise map slots
S_{off}^{M}	Off-premise map slots
S^{R}	Reduce slots
α	Hybrid map phase stretch factor
β	Rebalancing bandwidth
$f_{Sh}(d_{Sh})$ $f_{Sh}^{max}(d_{Sh})$	Shuffle time per reducer (avg and max)
$f_{Rd}(d_{Rd})$ $f_{Rd}^{max}(d_{Rd})$	Write time per reducer (avg and max)
Application-level parameters	
Name	Description
M	Total number of mappers
R	Total number of reducers
t_{Mp} and t_{Mp}^{max}	Total runtime for mappers (avg and max)
t_{Rc} and t_{Rc}^{max}	Sort and aggr. for reducers (avg and max)
d_{Sh}	Amount shuffle data per reducer
d_{Rd}	Amount of output data per reducer

- The average and maximum time to run a reducer computation (includes the time to sort and compute the aggregation, but not the time to write the result to HDFS): t_{Rc} and, respectively, t_{Rc}^{max}.
- The amount of input data in the shuffle phase, d_{Sh}.
- The amount of output data of the reduce phase, d_{Rd}.

Note that the MapReduce framework incurs a scheduling overhead, which is observable as a gaps between the tasks assigned to the mapper and reducer slots. For simplification, we assume t_{Mp} and t_{Rc} already include the average gap duration, while t_{Mp}^{max} and t_{Rc}^{max} already include the maximum gap duration.

We summarize these application-level parameters in Table 1.

5.4 Performance Model

In this section, we introduce a performance model that enables users to estimate the runtime of iterative MapReduce applications in hybrid cloud bursting scenarios. The performance model is a mathematical expression that uses the application-level and system-level parameters described in Table 1 as input and produces an estimation of the total runtime as output.

The lower bound of the completion time $Total$ of an iterative map-reduce job with n iterations, can be expressed as:

$$Total = \sum_{i=1}^{n} T(i) \tag{1}$$

$T(i)$ is the lower bound of the i-th iteration and can be decomposed as:

$$T(i) = T_{Mp}(i) + T_{Sh}(i) + T_{Rd}(i) \tag{2}$$

where T_{Mp} is the lower bound for the map phase, T_{Sh} for the shuffle phase and T_{Rd} for the reduce phase.

Similarly, for the upper bound we obtain:

$$Total^{max} = \sum_{i=1}^{n} T^{max}(i) \tag{3}$$

$$T^{max}(i) = T_{Mp}^{max}(i) + T_{Sh}^{max}(i) + T_{Rd}^{max}(i) \tag{4}$$

For the rest of this section, we detail how to obtain each of T_{Mp}, T_{Sh} and T_{Rd}.

5.4.1 Completion time of map phase

In order to obtain a mathematical expression to estimate the completion time of the map phase, it is important to understand how this phase evolves during the successive iterations of the MapReduce job. We force the execution of any map task on a node where there is a copy of its input data, so for the first iteration all the map tasks will be scheduled on-premise only. In parallel with the execution of the first iteration, the rebalancing of the input data to the off-premise infrastructure proceeds in the background. This means that at the beginning of the second iteration, some replicas of the input chunks have already been migrated on the off-premise nodes and the scheduler will launch off-premise mappers to handle them. As the rebalancing progresses, the number of map tasks that will be executed off-premise will increase at each iteration until it stabilizes (which is not necessarily the moment when the rebalancing has finished, because it can happen that the off-premise map slots are saturated even before a replica of each chunk was transferred off-premise).

We express this intuition mathematically as follows: for the first iteration, all map tasks (M) will be executed on-premise. So, considering that there are S_{on}^{M} map slots on the on-premise nodes, the lower bound of the map phase for the first iteration is:

$$T_{Mp}(1) = \left\lceil \frac{M}{S_{on}^{M}} \right\rceil \times t_{Mp} \tag{5}$$

For the second iteration, there will be a set of input chunks already transferred off-premise (M_1^{off}). This number can be approximated using the replication bandwidth (β), the size of the HDFS chunk s and the runtime of the first iteration $T(1)$ as follows:

$$M_1^{off} = \frac{\beta \times T(1)}{s} \tag{6}$$

These M_1^{off} off-premise chunks enable the scheduler to assign up to $M - M_1^{off}$ map slots off-premise. Thus, the lower bound of the map phase of the second iteration is:

$$T_{mp}(2) = \left\lceil \frac{M - M_1^{off}}{S_{on}^{M}} \right\rceil \times t_{Mp} \tag{7}$$

Using the previous reasoning for the third iteration, the lower bound of the map phase is:

$$T_{Mp}(3) = \left\lceil \frac{M - ((\beta \times T(2))/s)}{S_{on}^{M}} \right\rceil \times t_{Mp} \tag{8}$$

By generalization, for the i-iteration we obtain the following:

$$T_{Mp}(i) = \left\lceil \frac{M - ((\beta \times T(i-1))/s)}{S_{on}^{M}} \right\rceil \times t_{Mp} \tag{9}$$

This formula is true as long as all on-premise map slots are filled and there are off-premise idle map slots that cannot be used because the rebalancing did not ship enough chunk replicas off-premise. The moment when stabilization happens can be expressed mathematically as follows:

$$M^{off} \geq \frac{S_{off}^{M}}{(S_{on}^{M} + S_{off}^{M})} \times M \tag{10}$$

From this moment onward, the time to process on-premise scheduled maps is almost the same that the time to process off-premise maps. In this situation, the number of map tasks scheduled off-premise will be M^{off} and the number of map tasks scheduled on-premise $M - M^{off}$. These numbers will remain constant for the rest of the iterations, leading to the following expressions for the remaining runtime:

$$\begin{aligned} T_{Mp}(i) &= \left\lceil \frac{M - M^{off}}{S_{on}^{M}} \right\rceil \times t_{Mp} \text{ where} \\ M^{off} &= \frac{S_{off}^{M}}{(S_{on}^{M} + S_{off}^{M})} \times M \end{aligned} \tag{11}$$

It is possible to join the expressions in Equations 9 and 11 into a single expression that estimates the lower bound of the map phase for any iteration:

$$T_{Mp}(i) = \left\lceil \frac{M - M_{i-1}^{off}}{S_{on}^{M}} \right\rceil \times t_{Mp} \text{ where}$$
$$M_{i-1}^{off} = min\left(\frac{\beta \times T(i-1)}{s}, \frac{S_{off}^{M}}{S_{on}^{M} + S_{off}^{M}} \times M \right) \quad (12)$$

Upper bound for the map phase: By applying the theoretical makespan results detailed in Section 5.1 to the lower bound described in Equation 12, we obtain the following expression for the upper bound:

$$T_{Mp}^{max}(i) = \left(\left\lceil \frac{M - M_{i-1}^{off}}{S_{on}^{M}} \right\rceil - 1 \right) \times t_{Mp} + t_{Mp}^{max} \quad (13)$$

5.4.2 Completion time of the shuffle phase

The shuffle phase is entirely managed by the MapReduce runtime and has no application-specific overhead. Therefore, to estimate the lower bound of the shuffle phase we simply need to apply the shuffle phase approximation function f_{Sh} (obtained from the calibration step) to the shuffle size per reducer d_{Sh} (obtained from the application characterization), which results in the following:

$$T_{Sh}(i) = \left\lceil \frac{R}{S^R} \right\rceil \times f_{Sh}(d_{Sh}), \forall i = 1..n \quad (14)$$

Similarly, for the upper bound we apply the makespan results to obtain:

$$T_{Sh}^{max}(i) = \left(\left\lceil \frac{R}{S^R} \right\rceil - 1 \right) \times f_{Sh}(d_{Sh}) + f_{Sh}^{max}(d_{Sh}), \forall i = 1..n \quad (15)$$

5.4.3 Completion time of the reduce phase

The reduce phase consists of a number of reducers R that compete for a number of parallel reducer slots S^R. In this case, the average completion time of a reducer t_{Rd} depends on both the application-level parameters and the system-level parameters. Specifically:

$$t_{Rd} = t_{Rc} + f_{Rd}(d_{Rd}) \quad (16)$$

The explanation for this is the following: there is an application-dependent computational (t_{Rc}) overhead (obtained through characterization), in addition to the HDFS write overhead (obtained by applying the application agnostic approximation function f_{Rd} to the application-specific output size per reducer d_{Rd}).

Thus, the lower bound of the reduce phase can be estimated as follows:

$$T_{Rd}(i) = \left\lceil \frac{R}{S^R} \right\rceil \times t_{Rd}, \forall i = 1..n \quad (17)$$

Similarly, for the maximum the following applies:

$$t_{Rd}^{max} = t_{Rd}^{max} + f_{Rd}^{max}(d_{Rd}) \quad (18)$$

By applying the makespan results, we obtain the following upper bound:

$$T_{Rd}^{max}(i) = \left(\left\lceil \frac{R}{S^R} \right\rceil - 1 \right) \times t_{Rd} + t_{Rd}^{max}, \forall i = 1..n \quad (19)$$

5.5 Complex iterations

So far we made an important assumption about the applications: each iteration involves a single MapReduce job that is computationally similar to the previous iterations. However, in practice it can happen that iterations are complex and involve a series of steps expressed as separate MapReduce jobs (e.g. PageRank, as described in Section 6.4).

In this section we briefly show how to generalize our approach to address such complex iterations. Let m be the number of MapReduce jobs in a complex iteration i and $T_j(i)$ the runtime of the j-th MapReduce job in the sequence of m jobs. Then, the runtime of each complex iteration i is the sum of the durations of the m MapReduce jobs:

$$T(i) = \sum_{j=1}^{m} T_j(i) \quad (20)$$

For n iterations, the following holds:

$$Total = \sum_{i=1}^{n} \left(\sum_{j=1}^{m} T_j(i) \right) = \sum_{j=1}^{m} \left(\sum_{i=1}^{n} T_j(i) \right) = \sum_{j=1}^{m} Total_j \quad (21)$$

In other words, we can see an application with complex iterations as the equivalent serialization of m sub-applications with simple iterations. In this case, we can simply characterize each of the the m sub-applications, apply our mathematical expressions to estimate their individual completion time and finally sum up the estimations to obtain the final estimation for the original application with complex iterations. Since the system-level parameters are application agnostic, the calibration needs to be performed only once regardless of m.

6 Evaluation

In this section we evaluate the effectiveness of our approach experimentally, using a variety of scenarios and comparisons that involve multiple real-life iterative MapReduce applications.

6.1 Experimental Setup

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

We configure two separate IaaS clouds (*on-premise* and, respectively, *off-premise*), each running its separate OpenStack Icehouse instance. QEMU/KVM 0.12.1 is used as the hypervisor. The VM instances of the same cloud are configured to directly communicate with each other via the links of their compute node hosts. However, all communication outside of the same cloud is passing through a dedicated network node (Neutron) that acts as a proxy and is part

of the default OpenStack distribution. Thus, in a hybrid OpenStack setup the weak link is defined by the end-to-end bandwidth between the two proxies. We control the available bandwidth to cover two representative settings: 1 Gbps and 100 Mbps. These correspond to the case when the user decides to buy premium access to the cloud (i.e. dedicated fast link) vs. regular access.

6.2 Approaches

We compare four approaches throughout our evaluation.

On-Premise Actual: corresponds to the case when all VMs are on-premise and no weak link can cause an I/O bottleneck. In this case, a standard Hadoop deployment is used. We use it as a lower bound for comparison, showcasing what would happen in an ideal scenario where the user has no cost constraints and can afford to invest in additional on-premise resources to achieve the highest performance rather than adopt a hybrid solution.

ARIA (Automatic Resource Inference and Allocation): is a state of art framework that estimates the runtime of a single-site MapReduce job based on its profile (application-level parameters) and then optimally schedules it to meet a given soft deadline [15]. We use ARIA as a comparison in order to show that single-site techniques to estimate the runtime of iterative MapReduce are not accurate enough for use in a hybrid cloud bursting scenario, therefore the need for a specialized model.

Hybrid Actual Runtime: corresponds to the real measured runtime of an iterative MapReduce job using a given on-premise and off-premise configuration of VMs. The Hadoop deployment used to run the MapReduce job is optimized for a hybrid cloud bursting scenario using the rack-local scheduling and asynchronous rebalance techniques described in Section 4. We use this approach for comparison in order to showcase the accuracy of the estimations provided by our approach.

Hybrid Estimated Runtime: corresponds to the estimated runtime of an iterative MapReduce job using our proposal (Section 5), which is optimized for a hybrid cloud bursting scenario where the Hadoop deployment employs rack-local scheduling and asynchronous rebalance (Section 4).

6.3 Methodology

For our experiments, we created a new VM flavor with 4 vCPUs, HDD local storage of 100 GB and 16 GB of RAM. Thus, each compute node has the capacity to host 4 VMs simultaneously. Since some VMs are co-located on the same node, the virtual network interface of all VMs is limited to 1 Gbps, in order to avoid differences between VMs hosted on the same node vs. remote nodes. We use one fat node to provision up to 4 VMs on the on-premise part and three nodes to provision up to 12 VMs on the off-premise part. We deploy Hadoop 2.6.0 initially on-premise only: one VM is used as the Hadoop master (both MapReduce and HDFS), the rest of the VMs are used as Hadoop slaves (both MapReduce and HDFS). Each Hadoop slave is configured with enough capacity to run 4 mappers and 4 reducers simultaneously. Any initial input data is stored on-premise only in the initial HDFS deployment.

First, we run the application on-premise only and record the runtime, for the smallest case (3 VMs). We call this the *baseline* case. An important premise for any other setup (regardless whether on-premise or hybrid) is to show speed-up with respect to the baseline (otherwise it does not make sense to commit more VMs).

Then, using the profiling information, we extract the application-level parameters described in the characterization step (Section 5.3). For completeness, we also run *strong scalability* experiment (constant problem size) by increasing number of VMs from 6 up to 15 in steps of 3. This experiment is not involved in the extraction of the application-level parameters but facilitates the study of the results and corresponds to the *on-premise actual* case.

Second, we use the application-level parameters in order to estimate the runtime with ARIA. Again, we apply ARIA for an increasing number of VMs to show how the runtime scales in comparison with the baseline.

Third, we run another strong scalability experiment where we keep the number of on-premise VMs fixed at three, while adding an increasing number of off-premise VMs: from 3 up to 12 in steps of 3. For each resulting hybrid bursting scenario, we run: (1) the calibration (using the generic benchmark introduced in Section 5.2) to extract the system-level parameters, and (2) each application to obtain the *hybrid actual runtime*.

Finally, we use both the application-level and system-level parameters to estimate the runtime using our proposal for each hybrid cloud bursting scenario, which yields the *hybrid estimated runtime*. Note that many applications we study exhibit complex iterations composed of multiple MapReduce jobs. In this case, we apply the observations from Section 5.5 to compute the hybrid estimated runtime.

6.4 Applications

For the purpose of this work, we use four representative real-life iterative MapReduce applications that cover a broad spectrum: map-intensive, reduce-intensive or both.

Iterative Grep (I-GREP): is a popular analytics tool for large unstructured text. This application 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 job.

KMeans: is a widely used application for vector quantization in signal processing, cluster analysis in data mining, pattern classification and feature extraction for machine learning, etc [26]. It is based on iterative refinement: each iteration aims to improve the partitioning of a multi-dimensional vector into k clusters such that sum of squares of distances between all vectors of the same cluster and

their mean is minimized. This process repeats until the improvement obtained during an iteration is smaller than a predefined epsilon. K-Means was shown to be efficiently parallelizable and scales well using MapReduce [27], which makes it a popular tool to analyze large quantities of data at large scale. Each iteration reads a large amount of input data (which stays immutable during the iterations) while outputting a small quantity of intermediate and output data. From a data-management perspective, it is a good example of a map-intensive application that re-uses the initial input data at each iteration.

PageRank: is a link analysis algorithm [28] that assigns a numerical weight to each element of a hyperlinked set of documents, (e.g. WWW) with the purpose of quantifying its relative importance within the set. It is widely used in web search engines to calculate the ranks of web pages in function of the number of reference links. Its iterative nature is more complex and involves two successive MapReduce jobs: (1) an output-intensive phase where the reduce phase generates twice as much data as the input data read by the map phase; (2) a shuffle-intensive phase where the output of the mappers is equal in size to the input. Thus, PageRank is a good example of a balanced application with complex iterations that is both map-intensive and reduce-intensive, while generating a lot of intermediate data that is not reused.

Connected Components: is a well-known graph problem arising in a large number of applications including data mining, analysis of social networks, image analysis and related problems. It aims at identifying groups of connected vertices in a graph, which is an inherently iterative algorithm [29]: the input is an immutable graph (V, E) with a set of vertices V and a set of edges $E \subseteq V \times V$. The goal of this algorithm is to transform the input graph into a set of star-like subgraphs by iteratively assigning each vertex to its *smallest* neighbor, using a total ordering of the vertices such as the lexicographic order of the vertex labels. The MapReduce-based implementation results in the repeated execution of a job where the output of one iteration is the input of the next one. Connected components is a good example of a reduce-intensive application.

6.5 Results

Using the methodology presented in Section 6.3, we perform an experimental study for each of the real-life applications described in Section 6.4. For all runtime estimations, we compute both the lower and upper bound, and derive the average from the lower and upper bound. In addition, we also study the accuracy of all average estimations (vs. the actual runtime) at fine grain for both the map phase and the reduce phase separately, which provides additional insight with respect to the overall accuracy. For the rest of this section, we discuss the results for each application individually.

The first application we study is I-GREP. The implementation is based on grep, which is included with the Hadoop distribution. We use as input data 20 GB worth of Wikipedia articles, which are queried successively in 50 iterations using 50 different keywords. Each iteration is complex and is composed of two jobs per iteration (search and sort stages).

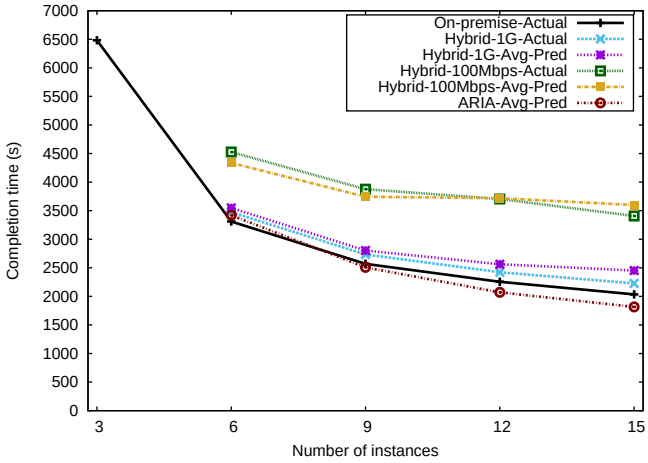


Fig. 4: I-GREP average runtime estimation

The shuffle data for each iteration is less than one MB, which means the map phase dominates the runtime. The baseline case (3 VMs on-premise) has a runtime of 6483 s. Doubling the amount of on-premise VMs leads to 60% less actual runtime, which shows I-GREP has a good scalability potential.

Table 2a shows the actual runtimes and the estimations for the upper and lower bounds. Due to high variability between the map tasks, ARIA shows large differences: for 15 VMs it overestimates the upper bound by 30% and underestimates the lower bound by 50% with respect to on-premise actual. Despite this variability, our approach has better accuracy: for 15 VMs and 100 Mbps weak link, it overestimates the upper bound by 20% and underestimates the lower bound by 10%. In case of 1 Gbps, it overestimates the upper bound by 30% and underestimates the lower bound by 13%.

We depict the average estimations in Figure 4. As can be observed, despite large difference between the ARIA upper and lower bound, the average estimation is much closer to the on-premise actual values. However, the average ARIA estimation produces large errors against the hybrid scenarios: up to 18.5% for 1 Gbps and up to 46.7% for 100 Mbps. This contrasts with the errors produced by our approach: up to 10.1% for 1 Gbps and 5.6% for 100 Mbps. Interesting to observe is the increasing accuracy of our approach when the weak link is of low capacity, while the opposite holds for ARIA.

Table 2b shows finer grain accuracy details about the map and reduce phase. As can be observed, in the 100 Mbps weak link case, the ARIA average estimation has a large error both for the map and reduce phase, which explains the overall error. For the 1 Gbps weak link case, the map phase has a small error but the reduce phase exhibits a large error.

Second, we study the K-Means application, as implemented in the Intel *HiBench* [30] big data suite. We generate 20 GB worth of input using the included data generator, which is processed by K-Means in 30 simple iterations. The baseline in this case (3 VMs on-premise) has a runtime of 6471 s. Doubling the number of on-premise VMs leads to 70% less actual runtime, which shows excellent scalability

Approach	6 VMs	9 VMs	12 VMs	15 VMs
On-premise Actual	3310	2569	2256	2035
ARIA Upper Bound	4226	3324	2892	2639
ARIA Lower Bound	2615	1692	1251	992
100 Mbps weak link (3 VMs on-premise)				
Hybrid Actual	4528	3876	3705	3407
Hybrid Upper Bound	4795	4282	4345	4141
Hybrid Lower Bound	3877	3208	3093	3057
1 Gbps weak link (3 VMs on-premise)				
Hybrid Actual	3474	2734	2422	2227
Hybrid Upper Bound	3953	3299	3087	2947
Hybrid Lower Bound	3144	2307	2038	1959

(a) Total actual runtime (s) vs. predicted runtime (s) expressed as upper bound and lower bound

Prediction	Accuracy vs. Actual	6 VMs	9 VMs	12 VMs	15 VMs
100 Mpbs weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	-18.2	-30.0	-40.0	-43.1
	(%) Red. Error	-56.3	-57.4	-59.4	-59.0
	(%) Total Error	-24.5	-35.3	-44.1	-46.7
Hybrid Avg.	(%) Map Error	-2.6	1.9	5.1	13.4
	(%) Red. Error	-12.4	-25.3	-17.2	-20.8
	(%) Total Error	-4.2	-3.4	0.4	5.6
1 Gbps weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	4.1	-1.8	-8.4	-12.8
	(%) Red. Error	-34.9	-36.5	-37.2	-37.5
	(%) Total Error	-1.5	-8.3	-14.5	-18.5
Hybrid Avg.	(%) Map Error	3.9	2.3	5.6	8.7
	(%) Red. Error	-8.3	3.6	6.7	-15.3
	(%) Total Error	2.1	2.5	5.8	10.1

(b) Accuracy of the average prediction (between lower and upper bound) vs. the hybrid actual runtime broken down by phase

TABLE 2: I-GREP: Map-intensive example of an iterative MapReduce application

Approach	6 VMs	9 VMs	12 VMs	15 VMs
On-premise Actual	3024	2159	1786	1511
ARIA Upper Bound	3501	2499	2020	1739
ARIA Lower Bound	2870	1857	1372	1089
100 Mbps weak link (3 VMs on-premise)				
Hybrid Actual	3872	3207	2893	2743
Hybrid Upper Bound	3900	3374	3266	3091
Hybrid Lower Bound	3649	2997	2799	2713
1 Gbps weak link (3 VMs on-premise)				
Hybrid Actual	3175	2391	1992	1685
Hybrid Upper Bound	3282	2495	2168	1858
Hybrid Lower Bound	3098	2137	1741	1543

(a) Total actual runtime (s) vs. predicted runtime (s) expressed as upper bound and lower bound

Prediction	Accuracy vs. Actual	6 VMs	9 VMs	12 VMs	15 VMs
100 Mpbs weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	-14.7	-29.3	-38.9	-46.9
	(%) Red. Error	-61.6	-63.9	-65.1	-63.9
	(%) Total Error	-17.7	-32.1	-41.4	-48.5
Hybrid Avg.	(%) Map Error	-1.6	1.8	7.5	8.6
	(%) Red. Error	-15.7	-29.1	-20.8	-21.4
	(%) Total Error	-2.5	-0.7	4.8	5.8
1 Gbps weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	2.6	-6.3	-12.0	-12.9
	(%) Red. Error	-41.5	-43.5	-45.1	-45.2
	(%) Total Error	0.3	-8.9	-14.9	-16.1
Hybrid Avg.	(%) Map Error	1.0	-3.5	-2.3	-0.2
	(%) Red. Error	-8.8	1.4	2.6	11.0
	(%) Total Error	0.5	-3.1	-1.9	0.9

(b) Accuracy of the average prediction (between lower and upper bound) vs. the hybrid actual runtime broken down by phase

TABLE 3: KMeans: Map-intensive example of an iterative MapReduce application

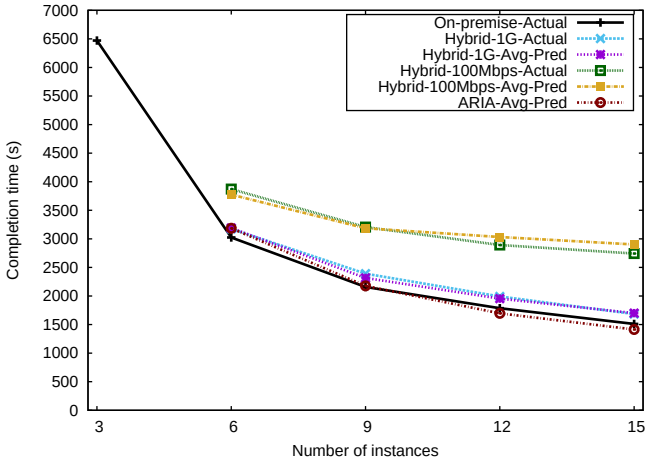


Fig. 5: K-Means average runtime estimation

potential.

Table 3a shows the runtime estimations for the upper and lower bound. Unlike the I-GREP case, K-Means exhibits less variability between the map tasks, which improves the accuracy of the ARIA upper and lower bound estimations with respect to actual on-premise. By comparison, our approach has much closer upper and lower bound estimations with respect to hybrid actual.

Figure 5 depicts the average estimations. The ARIA estimation produces a large error for the hybrid 100 Mbps weak link scenario, reaching almost 50%. For the 1 Gbps case, the error is smaller but still significant at 16%. Our approach reduces the error by an order of magnitude: 5% for the 100 Mbps case and less than 1% for the 1 Gbps case.

Table 3b shows finer grain details about the accuracy of the map and reduce phase. As can be observed, ARIA has low accuracy in the 100 Mbps case for both phases. In the 1 Gbps case, ARIA has low accuracy for the reduce phase. By contrast, our approach has good accuracy for both phases regardless of the weak link capacity.

Third, we study the PageRank application. We generate 2.8 GB of web data hyperlinks that is processed in 5 complex iterations (2 jobs per iteration). Again, we can see a good scalability potential: the baseline runtime of 3145 s (3 VMs on-premise) is reduced by 51% when doubling the number of on-premise VMs.

The runtime results are listed in Table 4a. In this case, the accuracy of the ARIA estimations for the largest scenario (15 VMs) are follows: the upper bound is overestimated by 16% and the lower bound is underestimated by 34%. Our approach on the other has the following accuracy with respect to hybrid actual: for 15 VMs and 100 Mbps weak link, the upper bound is overestimated by 22% and the lower bound is underestimated by 7%. In case of 1 Gbps, the upper bound is overestimated by 12% and the lower bound is underestimated by 16%.

The average estimations are depicted in Figure 6. As can

Approach	6 VMs	9 VMs	12 VMs	15 VMs
On-premise Actual	1511	1053	981	796
ARIA Upper Bound	1765	1287	1059	924
ARIA Lower Bound	1379	892	660	523
100 Mbps weak link (3 VMs on-premise)				
Hybrid Actual	4886	3764	3794	3330
Hybrid Upper Bound	5490	4006	4764	4064
Hybrid Lower Bound	4675	3607	3140	3102
1 Gbps weak link (3 VMs on-premise)				
Hybrid Actual	1585	1130	1059	892
Hybrid Upper Bound	1831	1256	1351	1006
Hybrid Lower Bound	1509	1037	850	767

(a) Total actual runtime (s) vs. predicted runtime (s) expressed as upper bound and lower bound

Prediction	Accuracy vs. Actual	6 VMs	9 VMs	12 VMs	15 VMs
100 Mpbs weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	-1.2	-6.6	-16.1	-13.1
	(%) Red. Error	-77.3	-79.8	-84.5	-85.3
	(%) Total Error	-67.8	-71.0	-77.4	-78.3
Hybrid Avg.	(%) Map Error	-0.1	-6.7	-7.4	-14.2
	(%) Red. Error	4.6	2.2	5.6	10.0
	(%) Total Error	4.0	1.1	4.2	7.6
1 Gbps weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	8.6	2.5	-11.6	-8.6
	(%) Red. Error	-5.8	-6.9	-22.8	-24.2
	(%) Total Error	-0.8	-3.6	-18.9	-18.9
Hybrid Avg.	(%) Map Error	5.2	-4.0	-9.0	-14.0
	(%) Red. Error	5.4	4.7	11.2	6.7
	(%) Total Error	5.4	1.5	3.9	-0.7

(b) Accuracy of the average prediction (between lower and upper bound) vs. the hybrid actual runtime broken down by phase

TABLE 4: PageRank: Balanced example of an iterative MapReduce application with complex (multi-job) iterations

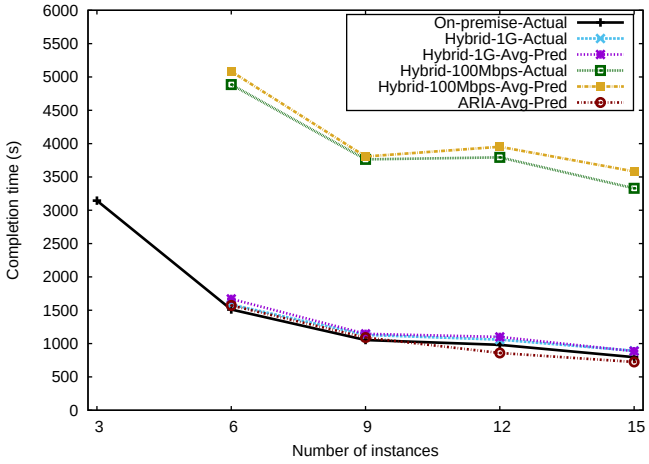


Fig. 6: PageRank average runtime estimation

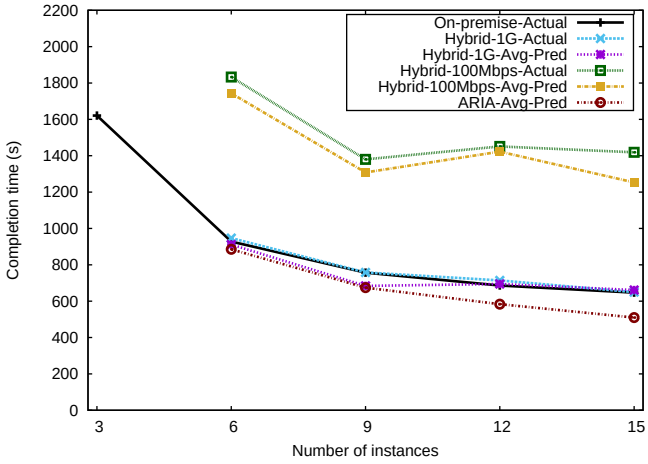


Fig. 7: Connected Components runtime estimation

be observed, when the application is balanced and exhibits both map-intensive and reduce-intensive behavior, the weak link is under I/O pressure, especially in the 100 Mbps case. Therefore, the average ARIA estimation has an error of almost 80% with respect to hybrid actual. In the case of 1 Gbps, the error is smaller at 18%. Nevertheless, our approach exhibits again an error that is an order of magnitude smaller: less than 8% for the 100 Mbps case and less than 1% for the 1 Gbps case.

Table 4b shows finer grain details about the accuracy of the map and reduce phase. For PageRank, the main source of the overall error seems to the reduce phase estimation, which clearly overshadows the map phase error. This hold both for 100 Mbps and 1 Gbps. Our approach has a good accuracy for both phases regardless of the weak link capacity.

The final application we study is Connected Components, which emphasizes the reduce phase. The implementation we use is part of Intel's *BigBench* [31] benchmark, which also includes a data generator. For the scale of our experiments, we generated 300 MB worth of input data (representing interactions in a social network). The application runs for 9 complex iteration, each of which is composed by 3 MapReduce jobs and includes additionally one final job. Connected Components runs in the baseline case (3 on-premise VMs) for 1621 s. Doubling the number of on-premise VMs leads to an actual runtime that is 42% smaller, which shows good scalability potential.

Table 5a shows the upper and lower bounds of the estimations. As can be observed, there is a large error for ARIA versus on-premise actual, especially for the lower bound (more than 300%). In this case, our approach has almost an order of magnitude better approximation (35%) versus hybrid actual.

The average estimations, depicted in Figure 7 exhibit a similar trend as before: in the 100 Mbps case, ARIA has an error of 64% compared with hybrid actual where our approach reduces this to 11%. In the 1 Gbps case, ARIA has a 21% error that our approach reduces to less than 2%.

Table 5b shows finer grain details about the accuracy of the map and reduce phase. Since the reduce phase dominates, large errors in the reduce phase translate to low overall accuracy. ARIA exhibits these large errors in the reduce phase where our approach does not, which explains the better overall accuracy.

7 Conclusions

This paper addresses the problem of how to estimate the runtime of iterative MapReduce applications in hybrid cloud bursting scenarios where on-premise and off-premise

Approach	6 VMs	9 VMs	12 VMs	15 VMs
On-premise Actual	928	757	686	647
ARIA Upper Bound	1178	972	883	810
ARIA Lower Bound	592	377	284	209
100 Mbps weak link (3 VMs on-premise)				
Hybrid Actual	1833	1380	1451	1419
Hybrid Upper Bound	2131	1585	1878	1450
Hybrid Lower Bound	1356	1031	967	1056
1 Gbps weak link (3 VMs on-premise)				
Hybrid Actual	947	758	714	650
Hybrid Upper Bound	1157	848	896	787
Hybrid Lower Bound	665	520	494	536

(a) Total actual runtime (s) vs. predicted runtime (s) expressed as upper bound and lower bound

Prediction	Precision vs. Actual	6 VMs	9 VMs	12 VMs	15 VMs
100 Mpbs weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	-12.9	-15.9	-29.0	-35.1
	(%) Red. Error	-66.8	-66.5	-72.9	-76.8
	(%) Total Error	-51.7	-51.1	-59.8	-64.1
Hybrid Avg.	(%) Map Error	-6.0	-14.2	-11.8	-6.8
	(%) Red. Error	-4.5	-1.3	2.2	-13.8
	(%) Total Error	-4.9	-5.2	-2.0	-11.7
1 Gbps weak link (3 VMs on-premise)					
ARIA Avg.	(%) Map Error	-4.5	-8.0	-15.4	-21.6
	(%) Red. Error	-8.3	-13.6	-14.4	-21.9
	(%) Total Error	-6.5	-11.0	-18.3	-21.6
Hybrid Avg.	(%) Map Error	-2.8	-13.7	-3.8	5.7
	(%) Red. Error	-4.5	-5.2	7.0	-3.5
	(%) Total Error	-3.8	-9.8	-2.7	1.7

(b) Accuracy of the average prediction (between lower and upper bound) vs. the hybrid actual runtime broken down by phase

TABLE 5: Connected Components: Reduce-intensive example of an iterative MapReduce application

VMs that host a MapReduce environment need to communicate over a weak link. Such runtime estimations are a critical tool in aiding the decision of whether the pay-as-you-go cost of cloud bursting justifies the expected speed-up.

To address this problem, we proposed a methodology that combines analytical modeling with synthetic benchmarking to estimate the time-to-solution specifically for a hybrid setup, where the weak link has a decisive impact both on the map and the reduce phase. We illustrated our proposal for the MapReduce runtime, however the principles are generic and can be applied to other runtimes that support similar computations, such as *Spark* [32].

We have demonstrated benefits for our proposal from multiple angles using a mix of map-intensive, reduce-intensive and balanced real-life iterative applications from standardized big data benchmarks that cover a broad spectrum of use cases. Specifically, we have shown that: (1) the upper and lower estimation bound of our approach against the hybrid baseline is significantly more accurate than the single-site counterparts against the on-premise baseline; (2) the average estimation of our approach is always within 1%-10% error regardless of scale and up to one order of magnitude more accurate than single-site state-of-art against the hybrid baseline; (3) our approach shows consistent behavior and accurately estimates both the map and the reduce phase, which means the overall estimation was not obtained by accident through the accumulation of large errors during the map and reduce phase that cancel each other out.

The trade-off for using our approach is the need for a one-time calibration phase that can be reused for all subsequent applications, which is additionally needed compared with single-site estimation techniques that rely solely on characterizing the application. However, cloud settings remain relatively stable over time (e.g., cloud providers rarely change VM flavors), so a user or cloud provider can easily cache the results of the synthetic benchmarks and reuse them subsequently as needed. We argue that this overhead is negligible considering the large improvements in the accuracy of the runtime estimations.

Encouraged by these results, we plan to explore in future work also the elasticity dimension. Specifically, we assume the user uses a fixed number of off-premise VMs while running the application. However, it may happen that a more elastic behavior is desirable where the off-premise configuration may change (e.g., add/remove VMs at each iteration) to adapt to changing goals (e.g., results needed faster than originally anticipated). In this context, the problem of how to formulate an elastic performance model capable to both recommend the best off-premise configuration and estimate the runtime for it is not well understood and of critical practical importance.

REFERENCES

[1] Cisco, "White paper: Cisco vni forecast and methodology, 2016."

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

[3] F. J. Clemente-Castelló, B. Nicolae, R. Mayo, J. C. Fernández, and M. M. Rafique, "On exploiting data locality for iterative mapreduce applications in hybrid clouds," in *BDCAT '16: 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, Shanghai, China, 2016, pp. 118–122.

[4] F. J. Clemente-Castelló, B. Nicolae, K. Katrinis, M. M. Rafique, R. Mayo, J. C. Fernández, and D. Loreti, "Enabling Big Data Analytics in the Hybrid Cloud Using Iterative MapReduce," in *UCC '15: 8th IEEE/ACM International Conference on Utility and Cloud Computing*, Limassol, Cyprus, 2015, pp. 290–299.

[5] T. Gunarathne, T.-L. Wu, J. Qiu, and G. Fox, "Mapreduce in the clouds for science," in *CloudCom '10: 2on IEEE Conference on Cloud Computing Technology and Science*, 2010, pp. 565–572.

[6] 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," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, pp. 363–373, 2014.

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

[8] ——, "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.

[9] B. Nicolae, C. Costa, C. Misale, K. Katrinis, and Y. Park, "Leveraging adaptive i/o to optimize collective data shuffling patterns for big data analytics," *IEEE Transactions on Parallel and Distributed Systems*, 2017, to appear.

[10] B. Nicolae, P. Riteau, and K. Keahey, "Towards Transparent Throughput Elasticity for IaaS Cloud Storage: Exploring the Benefits of Adaptive Block-Level Caching," *International Journal of Distributed Systems and Technologies*, vol. 6, no. 4, pp. 21–44, 2015.

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

[12] K. Chen, J. Powers, S. Guo, and F. Tian, "Cresp: Towards optimal resource provisioning for mapreduce computing in public clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 6, pp. 1403–1412, 2014.

[13] P. Lama and X. Zhou, "Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud," in *ICAC '12: 9th International Conference on Autonomic Computing*, New York, NY, USA, 2012, pp. 63–72.

[14] H. Herodotou, H. Lim, G. Luo, N. Borisov, L. Dong, F. B. Cetin, and S. Babu, "Starfish: A self-tuning system for big data analytics," in *CIRD '11: 5th Biennial Conference on Innovative Data Systems Research*, California, USA, 2011, pp. 261–272.

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

[16] A. Verma and R. H. Cherkasova, Ludmilaand Campbell, *Resource Provisioning Framework for MapReduce Jobs with Performance Goals*, Lisbon, Portugal, 2011, pp. 165–186.

[17] Z. Zhang, L. Cherkasova, and B. T. Loo, "Benchmarking approach for designing a mapreduce performance model," in *ICPE '13: 4th ACM/SPEC International Conference on Performance Engineering*, 2013, pp. 253–258.

[18] ——, "Performance modeling of mapreduce jobs in heterogeneous cloud environments," in *CLOUD '13: 6th IEEE International Conference on Cloud Computing*, Washington, DC, USA, 2013, pp. 839–846.

[19] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI '08: 8th USENIX Conference on Operating Systems Design and Implementation*, 2008, pp. 29–42.

[20] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Tarazu: Optimizing mapreduce on heterogeneous clusters," in *ASPLOS '12: 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 61–74.

[21] J. Xie, S. Yin, X. Ruan, Z. Ding, Y. Tian, J. Majors, A. Manzanares, and X. Qin, "Improving mapreduce performance through data placement in heterogeneous hadoop clusters," in *IPDPSW '10: IEEE International Symposium on Parallel Distributed Processing*, 2010, pp. 1–9.

[22] J. Polo, D. Carrera, Y. Becerra, V. Beltran, J. Torres, and E. Ayguadé, "Performance management of accelerated mapreduce workloads in heterogeneous clusters," 2010.

[23] K. Shvachko, H. Huang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *MSST '10: 26th IEEE Symposium on Massive Storage Systems and Technologies*, 2010.

[24] Apache. Apache rumen.

[25] S. Godard, "Sysstat: System performance tools for the linux os, 2004."

[26] H.-H. Bock, "Clustering methods: A history of K-Means algorithms," in *Selected Contributions in Data Analysis and Classification*, 2007, pp. 161–172.

[27] W. Zhao, H. Ma, and Q. He, "Parallel K-Means clustering based on MapReduce," in *CloudCom '09: 1st International Conference on Cloud Computing*, Beijing, China, 2009.

[28] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Comput. Netw. ISDN Syst.*, vol. 30, no. 1-7, pp. 107–117, 1998.

[29] T. Seidl, B. Boden, and S. Fries, *CC-MR – Finding Connected Components in Huge Graphs with MapReduce*, Berlin, Heidelberg, 2012, pp. 458–473.

[30] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *ICDEW '10: 26th IEEE International Conference on Data Engineering Workshops*, 2010, pp. 41–51.

[31] L. Wang, J. Zhan, C. Luo, Y. Zhu, Q. Yang, Y. He, W. Gao, Z. Jia, Y. Shi, S. Zhang *et al.*, "Bigdatabench: A big data benchmark suite from internet services," in *HPCA '14: 20th IEEE International Symposium on High Performance Computer Architecture*, 2014, pp. 488–499.

[32] A. Spark, "Apache spark: Lightning-fast cluster computing," 2014.

Francisco J. Clemente-Castelló received the B.S. degree in Computer Science from Jaume I University, Castellón (Spain), in 2011, and the M.S. degree in Intelligent Systems from the same university, in 2012. He is currently working towards the Ph.D. Degree in the Department of Computer Science Engineering at the Jaume I University. His main research interests include high performance computing platforms, virtualization, cloud computing technologies and big data analysis.

Bogdan Nicolae is a principal research scientist with Huawei Research Germany. He specializes in scalable storage, data management and fault tolerance for large scale distributed systems, with a focus on cloud computing and high performance architectures. He holds a PhD from University of Rennes 1, France and a Dipl. Eng. degree from Politehnica University Bucharest, Romania. He is interested by and authored numerous papers in the areas of scalable I/O, storage elasticity and virtualization, data and metadata decentralization and availability, multi-versioning, checkpoint-restart, live migration. He is also actively involved in the organization of international conferences (e.g., SC, IPDPS, HPDC, PPoPP, CCGrid, CLUSTER, CLOUD) and editing of journals (e.g., IEEE TCC) relevant to these areas.

Rafael Mayo received the B.S. degree from Polytechnic Valencia University in 1991. He obtained his Ph.D. in Computer Science in 2001 at the same University. Since October 2002 he has been an Associate Professor in the department of Computer Science and Engineering in the Jaume I University. His research interests include the optimization of numerical algorithms for general processors as well as for specific hardware (GPUs), and their parallelization on both message-passing parallel systems (mainly clusters) and shared-memory multiprocessors (SMPs, CCNUMA multiprocessors, and multicore processors). Nowadays he is involved in several research efforts on high performance computing energy-aware systems and cloud computing technologies.

Juan Carlos Fernández received the B.S. degree from Polytechnic Valencia University in 1992. He obtained his Ph.D. in Computer Science in 1999 in this University. Since October in 2002 he is an Associate Professor in the department of Computer Science and Engineering in the Jaume I University of Castellón (Spain). His research interests include the following topics: control algorithms in robot manipulators using parallel computing, parallel implementations of video encoder, energy saving on high performance computing platforms and cloud computing.