HybridMR: A Hierarchical MapReduce Scheduler for Hybrid Data Centers

Bikash Sharma*, Timothy Wood†, Chita R. Das*
*Department of Computer Science and Engineering, Pennsylvania State University
{bikash, das}@cse.psu.edu
†Department of Computer Science, George Washington University
timwood@gwu.edu

Technical Report CSE-12-007, December 2012

***Abstract*—Virtualized environments are attractive because they simplify cluster management, while facilitating cost-effective workload consolidation. As a result, virtual machines, either in public clouds or in private data centers, have become the norm for running many interactive applications such as web servers. On the other hand, batch workloads like MapReduce are typically deployed in a native cluster for avoiding the performance overheads of virtualization. While both these virtual and native environments have their own strengths, we believe it is feasible to provide the best of these two computing paradigms in a hybrid platform, and is the motivation of this paper.**

In this paper, we make a case for a hybrid compute cluster consisting of native and virtual servers and propose a 2-phase hierarchical scheduler, called *HybridMR* for effective resource management. In the first phase, *HybridMR* classifies incoming MapReduce jobs based on the expected virtualization overheads, and uses this information to automatically guide placement between physical and virtual machines. In the second phase, *HybridMR* manages the run-time performance of MapReduce jobs co-hosted with interactive applications in order to provide best effort delivery to batch jobs, while complying with the SLAs of interactive applications. By consolidating batch jobs with over-provisioned foreground applications, the available unused resources are better utilized, resulting in improved resource utilization and energy efficiency. Evaluations on a hybrid cluster consisting of 24 physical servers and 48 virtual machines, with diverse workload mix of interactive and batch MapReduce applications demonstrate that *HybridMR* can achieve up to 40% improvement in the completion times of MapReduce jobs, over virtual Hadoop, while complying with the target SLAs of interactive applications. Compared to native clusters, at the cost of minimal performance penalty, it boosts the resource utilization by 45%, resulting in around 43% energy savings. These results indicate that a hybrid cluster with an efficient scheduling mechanism can provide a cost-effective solution for hosting both batch and interactive workloads.

I. Introduction

Virtualization has evolved as a key technology to support agile and dynamic IT infrastructure which forms the base of large distributed systems like data centers and clouds. Virtualization enables autonomic management of underlying hardware, server sprawl reduction through workload consolidation, and dynamic resource allocations for better throughput and energy efficiency. Consequently, major cloud providers like Amazon EC2, RackSpace and Microsoft Azure utilize server virtualization to efficiently share resources among customers, and allow for rapid elasticity.

Despite these numerous benefits, virtualization introduces an extra software layer to the system stack, incurring overheads to native performance. Analyses with generic benchmarks have shown the virtualization overheads to be around 5% for computation and 15% for I/O workloads [8]. While these virtualization overheads have continued to fall with the introduction of virtualization-aware hardware [22], the effects are still large enough that many companies like Google and Facebook still prefer physical machines (PMs) over virtual machines (VMs) to run their core applications like Web search [3]. For data analytics frameworks such as Hadoop MapReduce [7] that allow for efficient large scale distributed computation over massive data sets [14], virtualized cloud platforms seem like a natural fit for providing elastic scalability. However, virtualization in clouds is also known to incur performance overheads, particularly when used for I/O intensive MapReduce activities [19], [20]. As a result, MapReduce users are often left with the choice of either maximizing performance with a native cluster or obtaining ease of use and resource efficiency with a virtualized environment.

Today's data centers offer two different modes of computing platforms - native clusters and virtualized clusters. Both these environments have their own strengths and weaknesses. For example, a native cluster is better for batch workloads like MapReduce from performance standpoint, but usually suffers from poor utilization, and high hardware and power cost. A virtual cluster, on the other hand, is attractive for interactive workloads from consolidation and cost standpoints, but may not provide competitive performance like a native cluster. Intuitively, a hybrid platform consisting of virtualized as well as a native cluster should be able to exploit the benefits of both environments for providing a better cost-effective platform. In this paper, we explore this design alternative, which we call *Hybrid compute cluster*, and demonstrate its advantages for supporting both interactive and batch workloads.

Transactional applications like interactive web services and virtual desktop environments are prime candidates for virtualization. For supporting the SLA requirements of interactive applications, resources are generally over-provisioned, which leads to poor utilization [13]. To exploit the potentials of both native and virtual cluster, we leverage the over-provisioning of bursty interactive applications by smartly consolidating batch

MapReduce jobs using the spare resources available on a virtualized platform. This allows for reaping the benefits of high consolidation in multi-tenant systems. This hybrid infrastructure presents different trade-offs across various design metrics like performance, cost, energy and resource utilization between native, virtual and hybrid design choices. For example, native cluster incurs higher cost in terms of power and physical hardware procurement, but lowers SLA violations, whereas virtualized cluster achieves higher utilization, elasticity and lower energy cost, but higher SLA violations. Hybrid compute cluster achieves a better balance between all these design criteria, making it a desirable cluster configuration option.

For facilitating such a hybrid server platform, this paper presents the design and implementation of a 2-phase hierarchical scheduler, called *HybridMR*, that judiciously allocates virtual and physical resources to applications. Contrary to the traditional workload placement schemes that completely isolate batch MapReduce and interactive workloads, *HybridMR* consolidates the workload mix in a heterogeneous infrastructure to achieve higher performance, utilization and energy efficiency targets. The design of such a scheduler requires the knowledge of how different MapReduce jobs are impacted by virtualization overheads, estimates of their resource needs, and an understanding of how batch jobs will impact the performance of interactive jobs co-hosted in VMs on the same host. *HybridMR* addresses these challenges through its 2-level scheduler design. Its first phase places MapReduce jobs on physical or virtual nodes depending on the expected virtualization overheads. Once a set of MapReduce jobs have been selected to run on the virtual cluster, along with interactive applications, the second phase scheduler decides *how much* resources that can be safely assigned to them. For this, it makes use of developed predictive models for understanding the runtime resource interference between the interactive and batch jobs, and employ dynamic resource management techniques to provide the best effort delivery to MapReduce jobs, while upholding the SLAs of interactive applications. *HybridMR* targets institutional and enterprise intranet environments, where the data centers run in a hybrid fashion, comprising of evolving cluster mix of native and virtual nodes.

In summary, we make the following contributions.

- We make a case for hybrid compute clusters with native and virtual machines for co-hosting both transactional and batch workloads for exploiting the best of these two computing worlds – native performance with virtualization benefits.
- Through detailed analysis, we demonstrate the trade-offs and benefits of running Hadoop in a virtualized platform, and leverage the insights in the design of *HybridMR*.
- We have designed and implemented *HybridMR*, a 2-phase hierarchical scheduler, for the effective placement of MapReduce jobs in a virtualized platform, while upholding the SLAs of interactive applications, through further dynamic resource management.
- Evaluations on a hybrid cluster consisting of 24 physical servers, and 48 virtual machines, with diverse workload mix of transactional and MapReduce applications, demonstrate that *HybridMR* can achieve up to 40% improvement in job completion times over virtual Hadoop, 45% improvement in resource utilization, and 43% savings in energy, both relative to native Hadoop. Further, we demonstrate the possibility to dynamically vary the native and virtual cluster configurations to accommodate variations in workload mix for maximizing the performance and energy benefits. These results suggest that a hybrid cluster configuration could be a better cost-effective solution than either-or native and virtual modes of computation.

II. MapReduce in Virtual Environment

HybridMR targets leveraging the unused spare resources in virtual clusters running interactive applications by consolidating batch MapReduce jobs. In this context, we address the following questions that are critical for running batch workloads like Hadoop MapReduce on a hybrid cluster.

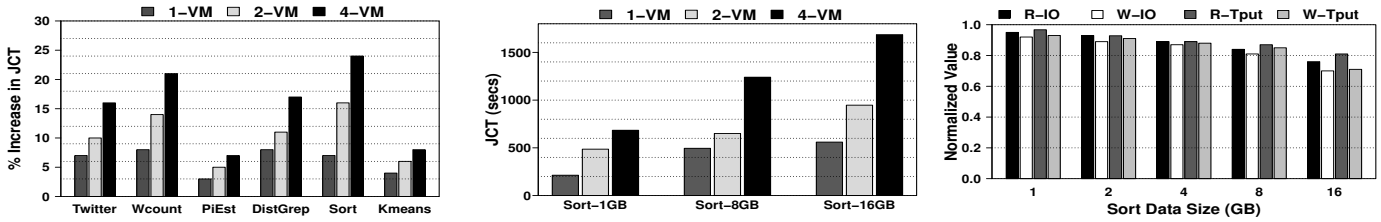
Q1. What are the challenges, benefits and trade-offs of running Hadoop on a virtual versus equivalent native cluster?

Q2. How does the 'data sticky-ness' of Hadoop affect the overall system performance in virtual Hadoop?

Q3. What system and application level changes are required to realize Hadoop's deployment on hybrid compute clusters?

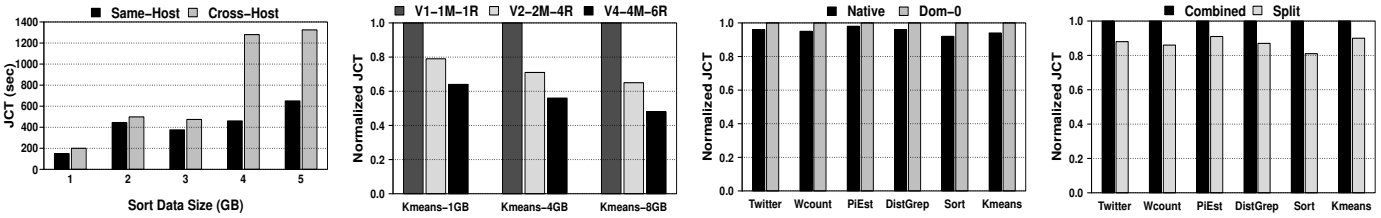
First, we perform detailed analysis and present empirical evidences demonstrating various challenges, benefits and trade-offs involved in deploying Hadoop MapReduce in a virtual cluster. We also highlight the various key design choices that impact Hadoop virtualization. Second, we present the split architecture of Hadoop that addresses the movement of large data across a virtual cluster during live VM migration. For this study, each VM is configured with 1 vCPU and 1024 MB Memory, and runs on dual-core, 4 GB RAM servers. We use a total of 1-24 PMs and 1-48 VMs, depending on the requirements of each experiment. Further details about the experimental platform, including the MapReduce benchmarks used are described in Section IV.

We start with comparing the performance of MapReduce benchmarks on native versus virtual environments. In Figure 1(a), the Y-axis represents the percentage increase in job completion time (JCT), with respect to a physical cluster. We can observe that with the increase in the number of VMs per PM, the performance (as quantified by JCT) of I/O bound jobs like Twitter, Wcount, DistGrep, Sort in virtual cluster is 7-24% worse than the physical cluster. For CPU bound jobs like PiEst, Kmeans, the performance difference is less than 8%. Moreover, for all benchmarks, as the size of input data increases, the performance gap widens, as evident in Figure 1(b) (only Sort shown). To investigate this behavior, we benchmark the performance of Hadoop Distributed File System (HDFS) in terms of read and write I/O, read and write throughput on a virtual cluster normalized with respect to the corresponding native cluster. We use Hadoop TestDFSIO [7] benchmark, and the results are shown in Figure 1(c). We notice that the virtual cluster performs relatively worse than the native cluster, and again this performance gap increases with the increase in data



(a) Performance of virtual Hadoop MapReduce (b) Impact of data size increase (c) HDFS performance on virtual Hadoop

Fig. 1: **Illustration of the virtualization overheads on Hadoop performance. Y-axis in (a), (c) normalized w.r.t. equivalent physical cluster. Figure 1(c): R/W-IO: Read/Write IO in MB/sec (Avg. IO rate); R/W-Tput: Read/Write-Throughput in MB/sec (total number of bytes/sum of processing times). In these experiments, total 48 VMs are used.**



(a) Network I/O effect (b) More CPU cycles effet (c) Native Vs. Dom-0 (d) Hadoop *Split* architecture

Fig. 2: **In (a), 16 VMs are used; 48 VMs are used in (b) and (c). In (b): V1, V2, V4 denote 1, 2, 4 VMs per PM. M and R denote number of Map and Reduce tasks. Y-axis in (b) and (c) is normalized w.r.t. Native; Y-axis in (d) is normalized w.r.t. Combined.**

size. The reasons for this poor performance are: (i) increase in contention for shared I/O resources across multiple VMs when large amount of data is transferred between map and reduce phase; (ii) poor performance of HDFS when multiple VMs concurrently read/write data blocks from/to HDFS; and (iii) presence of more stragglers, hence more speculative tasks, causing inefficient resource utilization [14]. amount of time. Data rate (which is less than the capacity of the link), is the rate at with which bits are placed on the link.

In another experiment, we study the impact of Hadoop VMs crossing multiple hosts due to large size of virtual cluster, limited host resources, and scattered data. For this, we create a 16 VMs virtual Hadoop cluster. In *Cross-Host*, the 16 VMs are equally distributed across 8 dual-core PMs, while in *Same-Host*, all the 16 VMs are equally consolidated on 2 PMs. Figure 2(a) shows the JCT of Sort in these two cases. The poor performance of *Cross-Host* compared to Same-Host is due to increase in the network I/O delay caused by remote VMs communication in *Cross-Host*. Further, the JCTs in each case increase as the input data scales up. An interesting observation here is that despite the *Cross-Host* having access to more cores (each VM gets 1 vCPU) when compared to *Same-Host* (each VM gets 0.25 vCPU), the performance of the former is still worse than the latter. Thus, when the data size and the number of concurrent running map/reduce tasks increase, the network communication overheads become the main resource bottleneck, and decides the overall performance.

Hadoop MapReduce can also perform better in virtual environment in certain scenarios. For example, CPU-bound jobs like Kmeans can benefit from having more VMs available, as shown in Figure 2(b). Further, this performance increase is higher for larger data size. This is because, a CPU-bound job can leverage more number of map/reduce tasks to finish fast due to the availability of more compute cycles, and hence more slots with more VMs on multi-core hosts. Note that in Figure 1(a), CPU-bound Kmeans tends to perform worse with number of VMs per PM. However, this is opposite to the result shown in Figure 2(b). This is because, for Figure 2(b) scenario, more number of map and reduce tasks can be launched to exploit the free CPU cycles (*i.e.,* slots), compared to the scenario in Figure 1(a), where fewer number (default 2) of concurrent mappers and reducers are present.

We next explore leveraging Xen's split architecture to run Hadoop in the privileged domain (Dom-0). Results in Figure 2(c) show that Dom-0 provides near native performance, with overheads less than 5%. This opens the possibility of a "flexibly virtualized" cluster, where some machines can be easily transitioned from running VMs to quasi-native applications in Dom-0 [22]. This supports hybrid clusters, whose configuration can be adjusted flexibly and on-demand.

Finally, one of the other concerns of deploying Hadoop on virtualized environment is the challenge to deal with the inherent 'data sticky-ness' of Hadoop, *i.e.,* how to deal with the movement of large amount of data living on Hadoop VMs, which may lead to high inefficiency both in terms of time and network communication overheads, thus defeating many of the benefits of virtualization. This also affects elasticity, *i.e.,* adding/removing Hadoop VMs to increase/reduce compute resources is ineffective due to the tight coupling between the compute and data storage layer. A prospective workaround to this problem is to place the TaskTracker (compute nodes) and DataNode (storage node) on separate VMs [33]. This *split* architecture (Figure 3) maintains data locality, and removes the constraint of moving large amounts of data around the cluster during VM migrations, since the data stays resident in the HDFS data layer, while the amount of map/reduce (compute) VMs can vary as required. We did some initial evaluations to assess the performance implications of this *split* architecture.

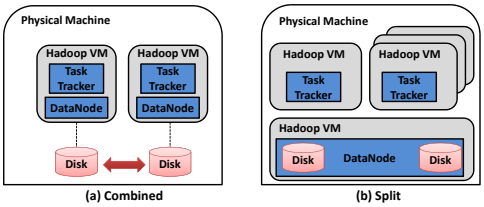


Fig. 3: **Hadoop Split Architecture.**

Figure 2(d) shows the results. We observe an average of 12.8% improvement in JCT over the default Hadoop (*Combined*), demonstrating some improvement over default virtual Hadoop. We adopt this *split* architecture in the paper.

To summarize, the above analysis clearly suggest that the performance of virtualized Hadoop is dependent on and governed by a multitude of design factors, and thus coming up with an optimal configuration both at the job and cluster level is non-trivial, making scheduling and resource management difficult. Some of the key observations are (i) Hadoop performance on virtual cluster is relatively worse than the equivalent physical cluster. The magnitude of performance degradation varies depending on the nature of the job. For example, jobs which are I/O and network intensive like Sort suffer more performance degradation than CPU-bound jobs like PiEst; (ii) input data size directly affects the magnitude of performance difference; (iii) Hadoop's performance difference between native and Dom-0 is marginal; (iv) *Split* Hadoop architecture optimizes its deployment on virtual environment.

III. Design of HybridMR

This section describes the overall architecture of *HybridMR* (as shown in Figure 4), which operates in two phases. In the first phase, *HybridMR* attempts to classify incoming MapReduce jobs based on the expected virtualization overheads, and uses that information to automatically guide placement between physical and virtual machines. The second phase performs dynamic resource management to minimize interference and improve performance of collocated interactive and batch jobs. Specific details of these phases are described below.

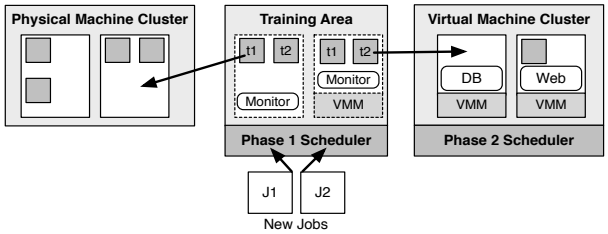


Fig. 4: **Overview of *HybridMR*.**

A. Phase I Scheduler

The main goal of this phase is to differentiate between workloads that should be scheduled on PMs or VMs running in a hybrid cluster. Since, our objective is to harness the spare resources on VMs running interactive applications, the interactive applications by default are assigned to the virtual cluster on their arrival, and the placement of the MapReduce jobs is governed by this phase. Thus, when a MapReduce job arrives, it is initially started separately on a small training cluster containing both physical and virtual environments, respectively (see Figure 4). We make use of statistical profiling technique to estimate the JCTs of MapReduce jobs (see Section III-A1). By comparing the estimated JCTs of the two instances of the job, corresponding to its run on native and virtualized servers, the level of performance overhead (quantified by JCT) incurred by the virtualization layer is estimated. If the overhead is not significant, then the job is selected for deployment on the virtual cluster, else it is run on a separate physical cluster. To estimate JCT, we leverage a job profiling technique, whose details are described next.

1) Job Profiling: We profile MapReduce jobs to estimate their JCTs prior to execution. MapReduce jobs are data-intensive and massively parallel, hence their JCTs are predominantly dependent on two factors: (i) input dataset size;

Algorithm 1 MapReduce Job Profiling Algorithm.

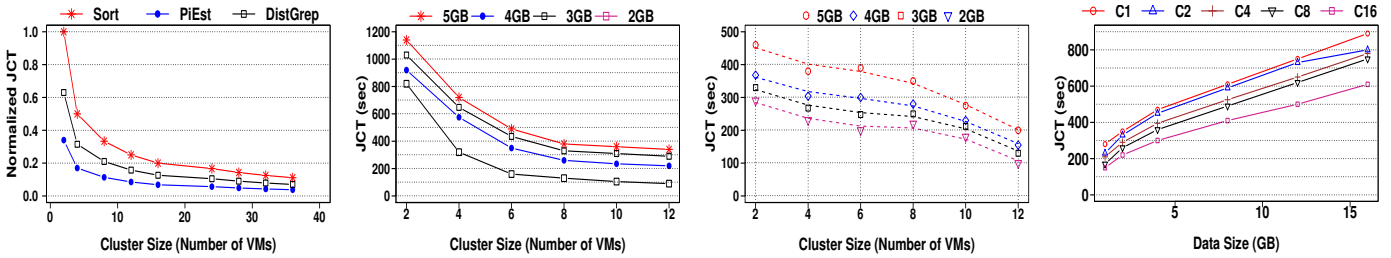
Input: Q: queue of incoming jobs, each specifying the input data size and/or cluster size; $DB_{profile}$: profile database, containing historic observations of job completion times (JCTs) (end-to-end and separate for map and reduce phases), along with corresponding cluster sizes and input data sizes.

1: **for** each job J_i in Q=J_1,J_2,...,J_n **do**
2: **if** LOOKUP_CLUSTERSIZE($DB_{profile}, J_i$) $\neq NULL$ & LOOKUP_DATASIZE($DB_{profile}, J_i$) $\neq NULL$ **then**
3: $JCT estimated$=Retrieve($DB_{profile}$, J_{csize}, J_{dsize})
4: **else if** $DB_{profile}$ does not contain exact match for J's input configuration **then**
5: **if** $DB_{profile}$ contains different data size values for the same cluster size (see Figure 5 (d)) **then**
6: Do linear extrapolation to get $JCT_{estimated}$
7: **else if** $DB_{profile}$ contains different cluster size values for the same data size **then**
8: Do separate Map and Reduce phase based extrapolation to get $JCT_{estimated}$ (see Figures 5 (a), (b), (c))
9: **end if**
10: **end if**
11: **return** $JCT_{estimated}$
12: **end for**

Algorithm 2 Job Placement Algorithm.

Input: Q: queue of incoming jobs; $Inputload$: number of clients for transactional or data size for MapReduce job; $P\_CLUSTER$: cluster of physical nodes; $V\_CLUSTER$: cluster of virtual nodes; $JCT_{desired}$: vector of jobs desired completion times.

1: **for** each job J_i in Q=J_1,J_2,...,J_n **do**
2: **if** $J_i \in$ transactional workload **then**
3: Place J_i on V_CLUSTER
4: **else if** $J_i \in$ batch MapReduce workload **then**
5: Profile J_i using Algorithm 1 to obtain the vector of estimated job completion time ($JCT_{estimated}$[]).
6: **if** $JCT_{estimated}[i] \geq JCT_{desired}[i]$ **then**
7: Place J_i on $P\_CLUSTER$
8: **else**
9: Place J_i on V_CLUSTER
10: **end if**
11: **end if**
12: **return** jobs assigned to $P\_CLUSTER$ and $V\_CLUSTER$
13: **end for**



(a) End-to-end job (b) Map Phase (c) Reduce Phase (d) Input Data Size

Fig. 5: **Dependence of job completion time on cluster size (end-to-end, separate for map and reduce phase), and input data size. In (d), C1, C2, C4, C8, C16 represent virtual cluster with 1, 2, 4, 8, 16 nodes, respectively. *Sort* is used in (b), (c) and (d).**

and (ii) resource set size, *i.e.,* number of nodes in the cluster. The estimation scheme accumulates a database of historic observations (in terms of job completion times corresponding to different input dataset sizes and cluster sizes). During training, a job is run on a representative small cluster and with a smaller dataset, and extrapolation techniques are used to estimate the runtime if the job were run on the full cluster and dataset. The profile database maintains separate run-times for map and reduce phases to account for the differences across phases. The exact association of a job's profile with the cluster size and input size is described next.

Cluster size: To quantify the dependence of cluster size on job completion time, we measure the total time as well as the time taken by its map and reduce phases separately, against different cluster sizes. In Figure 5(a), we observe that for a given data size, the JCT of a job follows an inverse relation to the cluster size. Same inverse relation follows for the completion time of map phase as shown in Figure 5(b). However, the reduce phase completion time follows a piece-wise non-linear relation with the cluster size as shown in Figure 5(c). This is due to the fact that the reduce phase's dependence on data size is highly erratic.

Data size: For a given cluster size, the JCT is linearly proportional to the input data size, as demonstrated in Figure 5(d). This indicates that by simple linear extrapolation, we can accurately estimate the JCT of a new job instance with a different data size. However, when sufficient historical job profiles are not available, we can still profile on a subset of data, and then use linear regression based extrapolation to get the JCT for the actual input size.

Thus, by building the job profiles using different data sizes and/or cluster sizes, this module estimates the job completion time. The end-to-end mechanism for job profiling based on the above logic is outlined in Algorithm 1.

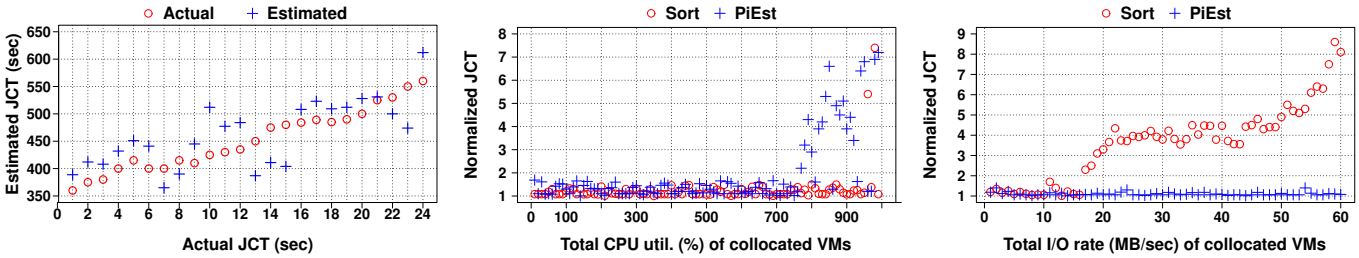
We build our database of job profiles using the average across three runs of each job instance. Note that similar procedure for profiling MapReduce jobs has been explored in previous studies [10], [32]. There are also other more complex profiling techniques [17], [32] for MapReduce jobs, where profiling is done for fine-grained stages of MapReduce framework. Also, the technique can be extended for online profiling [10], [32]. However, we observed that this simple profiling technique works best in practice for our purpose, incurring low overheads, while producing reasonable prediction accuracy. For example, Figure 6(a) shows the actual JCTs and estimated JCTs obtained from profiling of Sort on 10 GB. We got similar results for the other MapReduce benchmarks, and our profiling scheme introduced an average error of 10.8% with standard deviation of 9.7%.

2) Placement of MapReduce Jobs: When a job is submitted to the system, depending on the type of the job (transactional or batch), and its desired completion time, the heuristic outlined in Algorithm 2 determines its initial placement on the native or virtual nodes. Note that, Phase I simply steers the initial placement of the job, the exact configuration of VMs or PMs where the job will be run during its lifetime is determined by Phase II scheduler, as described next.

B. Phase II Scheduler

The goal of Phase II Scheduler is to efficiently manage the resources of the virtual cluster across transactional and batch MapReduce jobs, with an objective to comply with the SLAs of interactive jobs, while providing the best effort performance delivery to the MapReduce applications. Figure 7 shows the overall architecture of Phase II Scheduler. It is composed of two main components: (i) *a Dynamic Resource Manager (DRM)* and (ii) *an Interference Prevention System (IPS)*. The DRM monitors the available capacity on each VM to guide placement of MapReduce tasks within the virtual cluster. DRM records the resource consumption and completion times of each task run in the virtual cluster. This information is then used to build a model for each MapReduce job that correlates the resource allocation to the task completion time, allowing the scheduler to make intelligent decisions about where to place the remaining tasks of the job. The IPS is an online monitor that observes the performance of the interactive applications within the cluster to detect when they are not receiving sufficient resources to meet their demands. If this is detected, then the MapReduce tasks causing the interference are adjusted to reduce their impact. The VM running the task can have its resource share decreased, the VM can simply be paused, or it could even be migrated to a different host as part of different means to mitigate the observed interference. Even if the VM is fully stopped, the correctness of the corresponding MapReduce job is not affected, since the MapReduce master would regard this task as a prospective straggler and initiate its speculative execution [14].

To illustrate the level of interference which may be caused due to collocated VMs and/or contending tasks within the



(a) Job Profiling Error (b) CPU Interference (c) I/O Interference

Fig. 6: **(a) Profiling error in Phase I. (b), (c) show slowdown of JCT due to CPU and I/O interference from collocated VMs.**

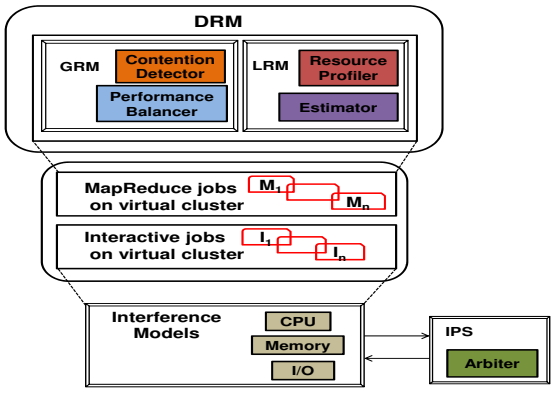


Fig. 7: **Components of Phase II Scheduler.**

same VM, we perform a study, where 4 VMs are deployed on a quad-core physical server, and run a mix of CPU and I/O bound MapReduce jobs. Each VM is pinned to a core, and 8 threads are run concurrently, sharing a single disk. We benchmark the completion time of CPU-bound PiEst and I/O-bound Sort. We first run each one of them on a single VM in isolation and note the JCT. We then measure the corresponding JCT when these jobs are run together with 3 other instances of PiEst and Sort, running on other VMs. Figure 6(b) shows the JCT of PiEst normalized to the corresponding JCT when run in isolation. The X-axis represents the total CPU utilization, represented as the percentage of a single thread. We observe the slowdown in the run-time when the CPU interference from other background jobs increases, while I/O-bound Sort remains unaffected. Similar observation follows for I/O-bound Sort (Figure 6(c)), where we can see exponential increase in JCT due to increased I/O contention from other collocated VMs. Thus, understanding the run-time interference, and performing dynamic resource adjustment is required to mitigate such performance degradation. This approach is adopted by our Phase II Scheduler, whose details are described next.

1) Dynamic Resource Manager (DRM): The goal of DRM is to build interference prediction models based on the run-time resource profiles of collocated tasks of MapReduce jobs and other co-hosted interactive jobs in order to estimate the task slowdown caused by negative resource contention. The estimations are used to orchestrate the optimal resource allocations via dynamic resource management to minimize the interference, and improve performance. DRM consists of two main components (a) *Global Resource Manager (GRM)*, and (b) *Local Resource Manager (LRM)*. The GRM consists of two sub-components: (i) a *Contention Detector* that dissects the cause of resource contention and identifies both resource-deficit and resource-hogging tasks and (ii) a *Performance Balancer* that leverages the run-time resource estimations from each LRM to suggest the resource adjustments based on the global coordinated view of all tasks. The LRM also consists of two sub-components: (i) a *Resource Profiler* that collects and profiles the run-time resource usage of tasks at each node and (ii) an *Estimator* that constructs statistical estimation models of a task's run-time performance as a function of its resource allocations. Due to space constraints, we are unable to provide specific details. For further details about each of these components, please refer to our prior work [30].

DRM performs the following two functions: (i) Resource bottleneck detection – Here, the GRM based on the resource usage feedback from all LRMs identifies *which task* is experiencing bottleneck for *which resource*, on *which TaskTracker*. The rationale and scheme behind this detection mechanism is outlined in [30]. (ii) Resource bottleneck mitigation: The LRMs after getting information about both resource deficit and resource hogging tasks from the GRM, invoke their Estimator module to estimate the tasks completion times. The Estimator builds predictive models for the completion time of a task as a function of its resource usage/allocation. The difference between the estimated and the current resource allocation is the resource imbalance that has to be dynamically allocated to the resource deficit task.

The Estimator module is responsible for building predictive performance interference models for tasks' run-time resource usage/allocations (we focus on CPU, memory and I/O resources). In a virtualized environment, the interference is contributed by both co-located tasks within the same VM, and the co-hosted VMs on the same host. It employs statistical regression models to obtain the estimated optimal resource allocation for the next epoch of a task's run, which operates in two stages. The first stage inputs a time-series of progress scores of a given map/reduce task and outputs a time-series of estimated completion times corresponding to multiple epochs in its life time. The second stage uses a time-series of past resource usage/allocations across the task's run-time starting from its start till the current time of estimation. The estimation model thus built outputs the estimated resource allocation for the next epoch as a function of its cumulative run-time. Separate estimation models are built for CPU, memory and I/O resource profiles of a task. We use the linear regression

model for CPU, and piece-wise linear regression model for memory, as derived from our previous work [30]. In this paper, we also model the I/O interference and build an estimation model. We use an exponential regression model to capture I/O interference, similar to [11]. For controlling the I/O bandwidth allocation to individual task, we use the recently released Linux kernel support for I/O throttling using cgroups [1]. Complete details about the construction of models, along with the specifics of dynamic resource allocation mechanism, are detailed in our previous work [30].

Algorithm 3 Arbitration Algorithm.

1: Obtain the list of map/reduce tasks, $TASK\_LIST_{interference}$, interfering with collocated interactive jobs from the *Estimator*
2: Evaluate the estimated interference for each task in $TASK\_LIST_{interference}$ using prediction model (see Section III-B1)
TaskInterference[]=GetEstimatedInterference()
# Get list of all available VMs in $V\_CLUSTER$
3: AvailableVMs[] = GetAllAvailableVMs()
4: **while** InterferenceTasks[] $\neq$ NULL **do**
5: Get best available VM using BestFit bin-packing heuristic [10]
6: VM_{best}=GetVMByBestFit()
7: Use Min-Min heuristics [11] to schedule the 'least-interference' task on VM_{best}
8: Adjust resources of VM_{best}
9: **end while**

2) Interference Prevention System (IPS): For the transactional workloads collocated with the MapReduce jobs in the virtualized environment, we build separate performance interference models, because their application characteristics are different from MapReduce jobs.We explored building both linear regression model as well as non-linear exponential regression model to quantify the CPU interference. However, we stick to a linear regression model as it fits well in our case in terms of offered simplicity and accuracy (see [30]). For memory, a piece-wise linear regression model well captures the memory interference [30]. For I/O, we use non-linear exponential regression based interference model. Specific details of the regression models used are similar to [9].

The IPS uses these performance interference models for interactive applications to dynamically orchestrate their resource allocations in the virtual cluster. The IPS continuously tracks the performance of interactive jobs. If at some instant, the performance falls outside allowable SLA bounds, the IPS invokes its *Arbiter* module to determine the specific task(s) and resource(s) that are causing this performance degradation. The Arbiter employs its arbitration scheme (Algorithm 3) to mitigate the interference, and restore the performance of the interactive applications. The Arbiter takes into consideration the interference from other collocated VMs and co-scheduled MapReduce jobs and packing of VMs criteria, while deciding on the exact migration of VMs to other hosts, or reassignment of interfering tasks to other VMs.

IV. Results

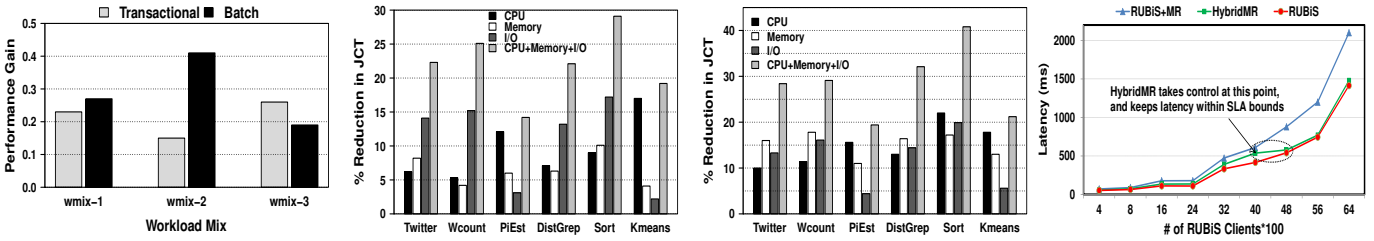
Infrastructure: Our experimental testbed contains a mix of physical and virtual nodes constituting native and virtual sub-clusters, respectively. The *native sub-cluster* contains 24 physical nodes. Each node has a dual 64-bit, 2.4 GHz AMD Opteron processor, 4 GB RAM, and Ultra320 SCSI disk drives, connected with 1 Gbps Ethernet. This cluster is installed with Hadoop v0.22.0 with 2 replicas per HDFS block, and 2 map, 2 reduce slots per node. Hadoop FairScheduler [15] scheduling policy is used. The *virtualized sub-cluster* contains 24 physical nodes which are virtualized with Xen v3.4.2 hypervisor [5], and host 2 VMs per PM to create an equivalent 48 nodes virtual cluster. Each VM runs RedHat AS4 with kernel v2.6.18, and the VMs are connected by 1 Gbps Ethernet. Each VM is configured with 1 GB RAM and 1 virtual CPU.

Benchmarks: For evaluations, we use a workload mix consisting of both transactional applications as well as batch workloads. For transactional workloads, we use three representative applications: (i) RUBiS [28], which models an online auction site; (ii) TPC-W [31], which models a three-tier online book store; and (iii) Olio [26], which is a Web 2.0 social events application. For batch workloads, we use the following representative MapReduce jobs.

- *Sort:* sorts 20 GB of text data (I/O-bound).
- *Wcount:* computes frequencies of words in 20 GB of text data (Memory + I/O-bound).
- *PiEst:* estimates Pi (10 million points) (CPU-bound).
- *DistGrep:* finds match of randomly chosen regular expressions on 20 GB of text data (I/O).
- *Twitter:* uses 25 GB twitter data [4] to rank users (Memory + I/O-bound).
- *Kmeans:* clusters 10 GB data (CPU-bound).

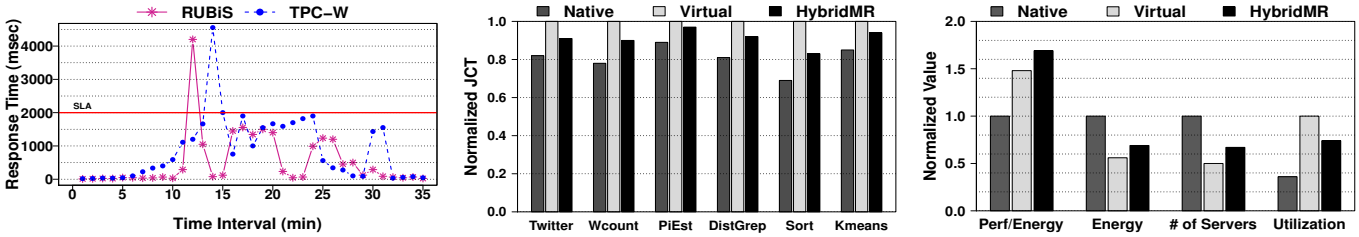
These MapReduce jobs are chosen based on their popularity and being representative of real MapReduce benchmarks, with diverse resource mix. We use Yokogawa's WT210 power meter to measure server power. For better precision, we run all our experiments 3 times, and use the average value. Unless otherwise specified, all experiments are performed in the hybrid cluster consisting of 48 VMs over 24 PMs.

Performance Benefits of *HybridMR*: We begin with demonstrating the performance benefits of *HybridMR*'s Phase I Scheduler in the initial placement of MapReduce jobs. Towards this, we perform an experiment, where we measure the average JCTs of interactive applications and MapReduce jobs, when scheduled by Phase I Scheduler, normalized against the corresponding JCTs measured with random placement of jobs (i.e., first-come-first-served discipline). In Figure 8(a), we plot this normalized JCT, labeled as 'Performance Gain'. We observe that through efficient placement by the Phase I Scheduler, both interactive and batch jobs benefit in performance, and the magnitude of gain varies depending on the workload mix characteristics. We consider 3 workload mixes: wmix-1, wmix-2 and wmix-3, where each represents 50% interactive + 50% batch jobs, 20% interactive + 80% batch jobs, and 80% interactive + 20% batch jobs, respectively. Intuitively, this performance gain is achievable due to the fact that certain MapReduce jobs which are resource intensive and have more stringent performance bounds when placed on native Hadoop



(a) Phase II Scheduler placement (b) Single job improvement (c) Multiple jobs improvement (d) SLA compliance of RUBiS

Fig. 8: **Performance benefits of *HybridMR* on a virtualized platform.**



(a) Performance of interactive applications (b) JCT improvement (c) Potential savings

Fig. 9: ***HybridMR* achieves better balance between Native and Virtual. Y-axis in (b) and (c) is normalized w.r.t. maximum.**

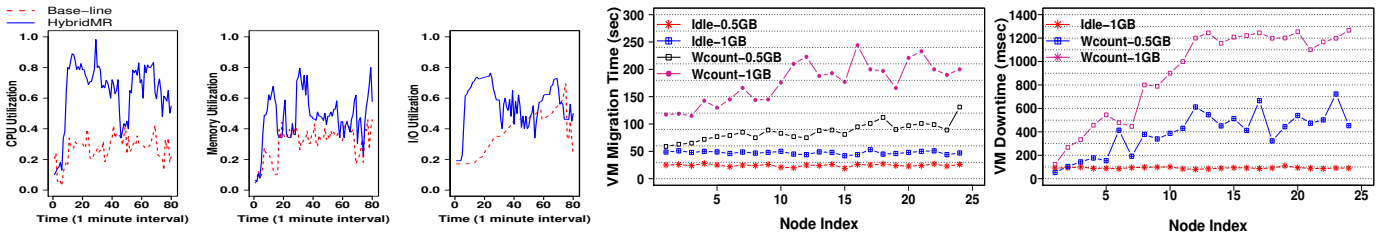
cluster reduces the interference across other MapReduce and interactive jobs placed on the virtual cluster.

We next analyze the potential benefits of Phase II Scheduler towards improving the performance of MapReduce jobs scheduled on the virtual cluster. In this experiment, we use 48 VMs virtual Hadoop cluster. We compare the performance of each of the six MapReduce jobs when run standalone (*Single*). We measure the JCT when Phase II Scheduler-based resource orchestration is enabled ($JCT_{hybridmr}$) versus without it ($JCT_{default}$). We compute percentage reduction in JCT as $(|JCT_{default} - JCT_{hybridmr}|)/JCT_{default}$*100. Figure 8(b) shows the percentage reduction in JCT of the six MapReduce jobs. Here, each legend corresponds to the case when only CPU, Memory, I/O or all three of them are managed by *HybridMR*. Across all jobs, *CPU+Memory+I/O* yields maximum reduction in JCTs, while the reduction with *CPU*, *Memory* or *I/O* alone varies in accordance with the benchmark resource characteristics. Specifically, we observe an average and a maximum JCT reduction of 22% and 29.1%, respectively, with *CPU+Memory+I/O* mode. Note that larger jobs (w.r.t. both large input size and long running job) like Sort, Kmeans benefit more when compared to other relatively shorter jobs like PiEst, DistGrep. This is due to the fact that larger jobs run in multiple map/reduce phases, and thus *HybridMR* has more opportunities to dynamically coordinate resources and improve their JCTs. Similar benefits are observed in the JCT of each job when run in the presence of other 5 concurrent running MapReduce jobs (*Multiple jobs*). Figure 8(c) shows the results. Note that since *Multiple jobs* scenario induces more inteference than corresponding *Single job*, more performance benefits are observed in *Multiple jobs*. We observe an average and a maximum reduction of 28.5% and 40.8% in the JCTs with *CPU+Memory+I/O*. In Figure 10(a), we also observe high CPU, Memory and I/O utilization achieved by *HybridMR*.

Next, we analyze the impact of *HybridMR* on the SLAs of interactive applications like RUBiS. We perform an experiment, where we increase the number of RUBiS clients and observe the impact on the latency of web server. Figure 8(d) shows the results. Here, the top blue curve corresponds to the case where the RUBiS virtualized cluster co-hosts other MapReduce jobs assigned in the FIFO order by the default MapReduce scheduler. The bottom red curve denotes the case where RUBiS runs in isolation with no MapReduce jobs. The middle green curve depicts the scenario with *HybridMR* scheduler. Here, we observe that for low client workloads, *HybridMR* is able to co-host MapReduce jobs with RUBiS and keep its latency within bounds. However, when *HybridMR* detects RUBiS client latency is exceeding a threshold, its Phase II Scheduler can adaptively migrate or pause the execution of co-hosted interfering MapReduce tasks to bring down the latency similar to the bottom red curve (RUBiS in isolation). This shows the dynamic adaptation of *HybridMR* in keeping up with the SLA of interactive applications.

In a related analysis, we demonstrate the effectiveness of *HybridMR* in maintaining the SLAs of interactive RUBiS and TPC-W applications, when they run collocated with other MapReduce jobs. In Figure 9(a), we observe that in the time interval 1-10 minutes, the response times of both RUBiS and TPC-W are below the user defined SLA (2 sec in our experiment). However, around 12th (RUBiS) and 14th (TPC-W) time instant, the response time exceeds SLA. *HybridMR* quickly identifies this, and is able to migrate the collocated map/reduce tasks from the VMs running RUBiS and TPC-W to mitigate the interference. Consequently, the response time falls within limits again. Similar behavior follows for Olio application, but we skip the analysis in interest of space.

Cross-Platform Performance Comparison: We next demonstrate the benefits of *HybridMR* in scheduling a mix of interactive and batch jobs. In this experiment, we evaluate



(a) Improvement in utilization (b) Impact of VM migration (c) Impact of VM downtime

Fig. 10: **In (a), *HybridMR* boosts resource utilization. (b) and (c) show performance impact of live migration of Hadoop VMs.**

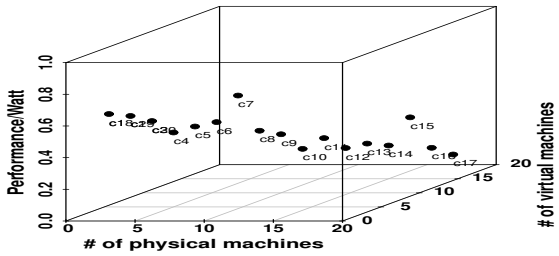


Fig. 11: **Design trade-offs analysis.**

three design choices for cluster configuration. In the first design choice (*Native*), the workload mix is scheduled on a 24-node native Hadoop cluster. In the second design choice (*Virtual*), the workload mix is scheduled on an equivalent 24-node virtual cluster (hosted on 12 PMs, each PM contain 2 VMs). The third design choice (*HybridMR*) corresponds to the case, where the workload mix is scheduled on an equivalent 24-node hybrid cluster, consisting of 12 PMs + 12 VMs (consolidated on 6 PMs with 2 VMs each). Thus, *Native*, *Virtual* and *HybridMR* requires 24, 12, and 18 physical servers, respectively. From Figure 9(b), we see the performance of MapReduce jobs is worse for *Virtual* because of the possible virtualization overheads. *Native* achieves the best performance as expected. *HbyridMR*'s performance is intermediate between *Native* and *Virtual*. However, as we observe in Figure 9(c), *HybridMR* achieves the highest *Performance/Energy* which is an important design metric. In terms of other metrics like energy, utilization, and number of physical servers required, *HybridMR* achieves a better balance between the two extreme choices – *Native* and *Virtual*.

Design Trade-off Analysis: We next discuss the flexibility of *HybridMR* in terms of splitting a given physical infrastructure into different hybrid cluster configurations, each consisting of a fixed number of VMs and PMs, and represents a particular level of performance and energy. To highlight this empirically, we split our testbed (consisting of 24 PMs hosting 48 VMs) into 20 different cluster configurations ($C_1 - C_{20}$), where each C_i contains a randomly selected number of PMs and VMs, and runs workload mix of interactive and MapReduce jobs. For each configuration, we measure the average completion times (performance), average power consumed and average utilization across all jobs and nodes. In Figure 11, each configuration corresponds to a particular $\langle \#ofPMs, \#ofVMs, performance/power \rangle$ tuple in the 3D space. Such analysis can provide some useful insights. For example, a cluster administrator may use some hints from a similar empirical analysis while deciding how to partition the infrastructure into hybrid cluster to meet a desired level of performance and power.

Impact of live migration of Hadoop VMs: In the IPS module (Section III-B2), we leverage live migration of VMs as a mechanism to migrate interfering map/reduce tasks. To understand the performance implications of Hadoop VMs migration, we perform some empirical analysis. We run Wcount on different data set size on a 24 VMs Hadoop cluster, and migrate randomly all the 24 VMs during the job run. Figure 10(b) and (c) show the migration time and downtime of each VM during its migration across hosts. From the figure, we observe: (i) larger the amount of memory involved, longer is the migration time, while the downtime dependence on memory is ad-hoc; (ii) migration time of VMs running Wcount is relatively longer than the corresponding idle Hadoop cluster; and (iii) downtime of each VM running Wcount shows wide variation, possibly due to skewness of each Hadoop VM, due to difference in resident data blocks, concurrent map/reduce tasks, and interference from other collocated VMs. From this analysis, we conclude that live migration of Hadoop VMs incurs some overheads, especially the downtime. However, due to the inherent fault tolerance characteristics of Hadoop (*i.e.,* replication), the downtime period can be balanced by regenerating the data blocks from other available copies. However, the MapReduce jobs still run to finish in the face of the migration induced downtime.

V. Related Work

Resource Scheduling in Hadoop MapReduce: Scheduling techniques for dynamic resource management of MapReduce jobs have been recently addressed [23], [27], [29], [30], [32]. Two popular recent resource managers for Hadoop MapReduce are Mesos [18] and YARN [16]. Most of the works in this category primarily focus on different scheduling policies for MapReduce jobs to reduce their completion times, improve cluster resource utilization and energy efficiency.

MapReduce and virtualization: There has been a recent push to improve the performance of MapReduce on virtualized platforms. Amazon Elastic MapReduce [6] is a publicly offered web service that runs on virtualized cloud platform. Serengeti [2] is an open source project initiated by VMware to enable rapid deployment of Hadoop cluster on a virtual platform. Cardosa et al. [10] proposed techniques for MapReduce provisioning in virtual cloud clusters, with emphasis

on energy reduction. Managing MapReduce workloads using Amazon virtual spot instances is studied in [12]. Resource allocation in Hadoop MapReduce cluster using dynamic prioritization is proposed in [29]. Preliminary evaluations of Hadoop MapReduce performance on virtualized clusters is recently done in [19], [33]. Virtual machine scheduling heuristic to improve Xen scheduler, targeting MapReduce workloads, is addressed in [20]. Harnessing unused CPU cycles in interactive clouds for batch MapReduce workloads has recently been motivated in [13]. Bu et al. [9] proposed an interference-aware and locality-aware scheduling algorithm for optimizing MapReduce in virtualized environment.

Interference based resource management in clouds: With the advent of cloud computing, resource management in virtualized clouds has emerged as an important research avenue. There exists quite a few literatures in this context. For example, Q-Clouds [25] leverages online feedback control technique to dynamically manage the resource allocation to VMs. TRACON [11] is an interference-aware scheduling algorithm for data-intensive applications in virtual environment. Automatic resource provisioning in MapReduce cloud clusters is explored in [10]. [21] presents an empirical study on the performance interference effects in virtual environments

We share our motivation of hybrid cloud clusters with [13], [22], [24]. We believe our work differs from others in the following manner. First, a detailed empirical evaluation and performance analysis study of Hadoop MapReduce on virtual environment have not been addressed in prior literature. Second, scheduling of heterogeneous workloads (mix of transactional and batch MapReduce jobs) on a hybrid cluster (consisting of both native and virtual environments) to exploit the spare resources available due to over-provisioning of interactive applications, has not been explored before. Third, our proposed hierarchical scheduler, *HybridMR*, uniquely focuses on the performance enhancement of MapReduce jobs, collocated with other interactive jobs in a virtual environment while complying with the SLAs of the foreground jobs. Fourth, no previous studies have paid much attention to the benefits and design trade-offs of hybrid compute clusters consisting of both native and virtual servers, and hosting heterogeneous workload mix.

VI. Conclusions

This paper presents a 2-phase hierarchical scheduler, called *HybridMR*, for hybrid server platforms consisting of a mix of native and virtual machines to leverage the benefits of both paradigms. In the first phase, *HybridMR* profiles incoming MapReduce jobs to determine the estimated virtualization overheads, and uses this information to automatically guide placement between physical machines and virtual machines. In the second phase, *HybridMR* builds run-time resource prediction models, and performs dynamic resource orchestration to minimize the interference within and across collocated MapReduce and interactive applications. Detailed evaluations on a hybrid cluster consisting of 24 physical servers and 48 virtual machines, with diverse workload mix of interactive and batch MapReduce applications demonstrate that *HybridMR* achieves up to 40% improvement in job completion time of MapReduce jobs over a virtual Hadoop, while satisfying the SLAs of interactive applications. Further, *HybridMR* provides 45% improvement in resource utilization and around 43% energy savings compared to a native Hadoop with minimal performance penalty. Besides, we show that it is possible to dynamically change the native and virtual cluster configurations to accommodate variations in workload mix for maximizing the performance-energy envelope. These results suggest that a hybrid cluster configuration is a cost-effective solution than either-or native and virtual modes of computation.

References

[1] Cgroup support for I/O throttling. http://tinyurl.com/io-throttle.
[2] Hadoop on vmware. http://serengeti.cloudfoundry.com/.
[3] No virtualization cloud computing. http://tinyurl.com/nohadoopVM.
[4] Twitter traces. http://an.kaist.ac.kr/traces/WWW2010.html.
[5] Xen hypervisor. http://www.xen.org.
[6] Amazon. MapReduce. http://aws.amazon.com/elasticmapreduce/.
[7] Apache. Hadoop. http://hadoop.apache.org.
[8] Paul Barham and et al. Xen and the art of virtualization. In *ACM SOSP*, 2003.
[9] X. Bu, C. Xu, and J. Rao. Interference and Locality-Aware Scheduling for MapReduce Service in Virtual Clusters. Technical report, Univ. of Colarado.
[10] M. Cardosa, P. Narang, A. Chandra, H. Pucha, and A. Singh. Steamengine: Driving mapreduce provisioning in the cloud. In *IEEE HiPC*, 2011.
[11] Ron C. Chiang and H. Howie Huang. Tracon: interference-aware scheduling for data-intensive applications in virtualized environments. In *SC*, 2011.
[12] N. et al. Chohan. See spot run: using spot instances for mapreduce workflows. In *HotCloud*, 2010.
[13] Ross Benjamin Clay. Enabling MapReduce to Harness Idle Cycles in Interactive-User Clouds. Master's thesis, NC State University, 2011.
[14] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *ACM Comm.*, 2008.
[15] Hadoop. Fair scheduler. http://hadoop.apache.org/common/docs/r0.20.2/fair_scheduler.html.
[16] Hadoop. Hadoop YARN. http://tinyurl.com/hadoop-yarn.
[17] H. Herodotou and S. Babu. Profiling, what-if analysis, and cost-based optimization of mapreduce programs. *PVLDB*, 4(11):1111–1122, 2011.
[18] B. Hindman and et al. Mesos: a platform for fine-grained resource sharing in the data center. In *USENIX NSDI*, 2011.
[19] Shadi Ibrahim and et al. Evaluating MapReduce on Virtual Machines: The Hadoop Case. In *IEEE CloudCom*. 2009.
[20] H. Kang and et al. Enhancement of xen's scheduler for mapreduce workloads. In *ACM HPDC*, 2011.
[21] Y. Koh and et al. An analysis of performance interference effects in virtual environments. In *IEEE ISPASS*, 2007.
[22] T. Kooburat and M. Swift. The best of both worlds with on-demand virtualization. In *USENIX Hot topics in operating systems*, 2011.
[23] Gunho Lee and et al. Heterogeneity-aware resource allocation and scheduling in the cloud. In *USENIX HotCloud*, 2011.
[24] H. Lin and et al. Moon: Mapreduce on opportunistic environments. In *ACM HPDC*, 2010.
[25] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *EuroSys*, 2010.
[26] Olio. Web 2.0 social events application. http://incubator.apache.org/olio.
[27] J. Polo and et al. Resource-Aware Adaptive Scheduling for MapReduce Clusters. In *USENIX Middleware*, 2011.
[28] RUBiS. E-commerce application. http://rubis.ow2.org.
[29] T. Sandholm and K. Lai. Mapreduce optimization using regulated dynamic prioritization. In *SIGMETRICS*, 2009.
[30] B. Sharma, R. Prabhakar, S. Lim, M.T. Kandemir, and C.R. Das. Mrorchestrator: A fine-grained resource orchestration framework for mapreduce clusters. In *IEEE CLOUD*, 2012.
[31] TPC-W. E-commerce benchmarking. http://www.tpc.org/tpcw/.
[32] A. Verma and et al. ARIA: Automatic Resource Inference and Allocation for Mapreduce environments. In *ICAC*, 2011.
[33] Vmware. Elastic hadoop for cloud. http://tinyurl.com/elastic-hadoop.