Adaptive Cluster Sizing for Residual Resource Harvesting in Interactive Clouds

R. Benjamin Clay, Zhiming Shen, Xiaosong Ma, Xiaohui Gu

Department of Computer Science, North Carolina State University, Raleigh, North Carolina
{rbclay,zshen5}@ncsu.edu, {ma,gu}@csc.ncsu.edu

Abstract. The popularity of cloud-based interactive computing services (e.g., virtual desktops) brings new management challenges: each VM leaves abundant but fluctuating residual resources while being intolerant to latency, potentially caused by aggressive VM consolidation. In this paper, we design and implement RHIC, an autonomous management framework to harness such dynamic residual resources aggressively without slowing the harvested interactive services. RHIC builds a batch cluster using a hybrid of residual and dedicated resources and intelligently discovers and maintains the ideal cluster size and composition, given goals such as cost/energy minimization or deadlines. RHIC employs black-box techniques and requires only system-level metrics to model workload performance. We demonstrate the effectiveness and adaptivity of our RHIC prototype with two parallel data analytics frameworks, Hadoop and HBase. Our results show that RHIC finds near-ideal cluster sizes/compositions across workloads and goals. Further, it significantly outperforms alternative approaches and tolerates high instability in the harvested interactive cloud.

1 Introduction

Interactive cloud offerings are expanding, providing people with virtual computing laboratories, remote desktop environments, and online collaboration tools. For example, North Carolina State University's Virtual Computing Laboratory (VCL) [1] is a production cloud system providing virtual desktops with a variety of applications to more than 13,000 students and faculty at NCSU and other nearby schools. These new platforms bring great convenience in accessing popular applications and tools, with little software/hardware/licensing cost. Meanwhile, such systems also yield significant amount of *residual*, or unused, resources, due to overprovisioning and the bursty, unpredictable nature of interactive workloads. Traditional techniques such as virtual machine (VM) packing are unlikely to be performed aggressively with interactive cloud workloads, as a result of users' bursty resource consumption patterns, combined with response time requirements (detailed discussion on this is given in §3). Due to such requirements for conservative workload consolidation, there will likely exist *significant* amounts of residual resources left from interactive workloads. By aggressively harnessing such resources, cloud service users and providers will benefit from higher cloud utilization, as well as considerable energy savings, as the *incremental* energy cost of running additional applications using residual CPU is low [2].

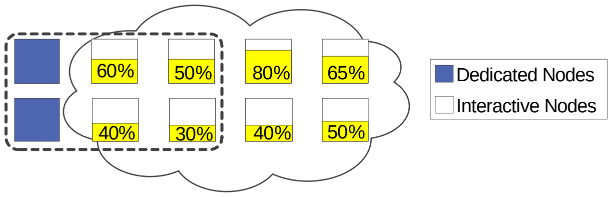


Fig. 1: **Hybrid cloud computing design.** Interactive users are “padded” with volunteers, which harvest residual CPU cycles unobtrusively.

Harvesting residual resources in such a context requires a well-designed infrastructure that considers performance, cost-effectiveness, and system reliability. In particular, using interactive nodes alone will suffer from performance and stability issues. Prior studies [3–5] have proposed a hybrid batch cluster design where *volunteer* nodes supplement a core set of stable *dedicated* nodes. Similar to previous efforts [4, 5] using EC2 SPOT instances [6], and motivated by preliminary experiments (§3), we choose to adopt such an asymmetric architecture. Different from prior works, a subset of transient interactive nodes are “padded” with volunteer VMs, which consume residual resources while automatically deferring to the interactive user via hypervisor prioritization. Figure 1 shows this hybrid cluster design. In this cluster design, the dedicated nodes have node-local storage capacity, while the volunteer nodes only use their local storage for temporary data, due to the lack of robust shared storage (i.e. Amazon’s S3) in VCL and similar-sized clouds. This choice allows volunteers to be lightweight and agile, avoids expensive data replication and consistency operations as volunteers join and leave, and provides a performance baseline to mitigate stragglers. More details on the applicability and tradeoffs of this approach can be found in §3.

In this setting, the cloud administrator is faced with the following question: *Given an arbitrary batch job, and limited knowledge about the interactive workloads, what hybrid cluster size and composition will give the best performance for the cost?* This problem can be formulated as a dynamic, virtualized cluster-sizing problem with new challenges. Unlike traditional cluster-sizing scenarios, the highly-dynamic nature of this environment introduces substantial complications when modeling performance, determining an ideal cluster size and selecting cluster composition. For example, Table 1 shows the diverse range of monetary costs and energy consumption among different batch workloads. These results are dependent on the specific batch inputs, foreground workloads and pricing structure chosen, as well as our cluster hardware, network and energy characteristics. Differences in these factors yield different ideal cluster sizes.

Existing work has addressed several related problems, including MapReduce cluster sizing [7–10], volunteerism/hybrid clusters for MapReduce [3–5], and workload consolidation [11, 12]. However, these prior studies were not designed to address the unique challenges faced in harvesting residual resources from interactive users, particularly (1) the high degree of temporal and spacial transience in residual resources, and (2) the dedicated node I/O saturation constraint with our proposed asymmetric architecture. More detailed related work discussion is given in §2.

To address such unique challenges in accurately discovering and maintaining the ideal hybrid cluster size for arbitrary batch workloads, either white-box or black-box performance modeling can be used, but each has downsides. Black-box performance modeling using system-level metrics enables generalization and unobtrusiveness, but such metrics can be very noisy. White-box modeling allows higher sensitivity to the limitations of a particular framework, and potentially greater accuracy, but limits generalization. Real-world parallel batch workloads are commonly composed of short jobs [13] and novel jobs [7, 14]. As a result, profiling must be completed quickly with no *a priori* knowledge to yield reliable estimates early.

Table 1: **Cost and energy ranges for different batch workloads**, with 6 dedicated nodes and 0-36 volunteers on NCSU's ARC cluster [15].

	Cost ($)		**Watt Hrs**	
Workload	**Min**	**Max**	**Min**	**Max**
Wordcount	4.42	6.38	1273	1957
Grep	2.40	5.83	710	894
Pi	9.25	16.63	2963	5461
Word Co-oc.	7.72	11.41	2230	3987

To tackle these challenges, we present the Resource Harvester for Interactive Clouds (RHIC), a generic management framework which autonomically optimizes a hybrid cluster running within residual resources. RHIC can provide intelligent cluster sizing for a wide range of throughput-oriented parallel batch workloads. To accomplish this, RHIC combines profiling with black-box performance modeling to make resizing decisions in an iterative, online fashion. We profile the CPU, memory and I/O consumption of each workload and build self-tuning models to translate these system-level metrics into job performance estimates. Finally, we tailor this approach to the hybrid cluster design, by predicting residual resource availability at the volunteers and directly managing I/O saturation at the dedicated nodes. Our multi-faceted approach handles dynamic and unpredictable behavior from a wide range of independent sources, aggregating unstable resources into a reliable batch platform. Through extensive evaluation, we show that RHIC robustly delivers accurate performance estimates and quickly discovers the best cluster size for novel workloads.

We consider the major contributions of this work as follows:

- To the best of our knowledge, we are the first to propose batch cluster sizing as a tool for residual resource harvesting in interactive clouds.
- We present an adaptive cluster sizing solution that uses a combination of online profiling and performance modeling to quickly discover and maintain efficient hybrid cluster sizes.
- We develop black-box batch job performance models which map aggregate residual resources to goal performance. Our self-tuning models rely on system metrics and a progress score, which allows generalization to a wide range of throughput-oriented parallel batch workloads.
- We carried out real-system evaluation with up to 42 nodes, using real traces collected from production interactive cloud systems and representative batch distributed analytics workloads. Our results show that RHIC achieves high accuracy in enforcing minimum cost/energy and deadlines for a range of workloads, as compared to an exhaustive survey and an alternative control-theory algorithm. In addition, RHIC is able to automatically adapt to different optimization goals, system resource configurations, and different interactive/batch workload combinations.

In the rest of the paper, we give an overview of related work in Section 2, outline background information in Section 3, and discuss the design of RHIC in Section 4. In

Section 5 we present the results and analysis of our evaluation. Finally, we conclude in Section 6.

2 Related Work

Our work is most-relevant to prior efforts towards MapReduce cluster sizing, but also relates to to contributions from several other areas.

Cluster sizing for parallel batch workloads. Several works have been recently published which perform cluster sizing for parallel batch workloads [7–10]. Of these, our efforts are most-closely related to those which combine modeling with online adjustment and feedback [7–9]. Jockey [7] is a system for meeting deadlines in MapReduce clusters using offline profiling/simulation, coupled with an online control loop which can adapt to cluster availability. Conductor [8] also combines modeling and online adjustment to meet deadlines and minimize cost for MapReduce, taking into account data upload and migration overheads. RAS [9] is a MapReduce scheduler that profiles the resource requirements of Map/Reduce tasks and then attempts to allocate sufficient slots for each running job to meet soft deadlines. Starfish [10] is a system for optimizing cluster size for arbitrary MapReduce workloads and hardware, using a combination of workload profiling and and configuration parameter modeling.

Compared to the aforementioned efforts, RHIC addresses a unique permutation of traditional cluster sizing for parallel batch workloads. As a result, we consider several sub-problems which are specific to our harvesting theme, including foreground demand prediction, heuristic node selection, I/O saturation awareness, I/O curve discovery and heterogeneity-tolerant performance modeling. In summary, the differences between RHIC and the aforementioned MapReduce cluster-sizing efforts are as follows: (1) the uniquely unstable environment in which we operate, (2) our support for novel, short-lived jobs, and (3) the general applicability of our modeling approach to a broad class of parallel batch workloads.

Because we rely on the foreground user for dynamic residual CPU and static residual memory availability, each volunteer node offers a varying contribution to the job's completion time. As a result, node or task-level performance modeling [7–10] will not adequately capture the performance of a given cluster. Our insight regarding aggregate residual CPU availability and its direct effect on cluster performance (§4.3) led to RHIC's CPU-centric modeling approach. Further, hybrid clusters have significant I/O restrictions since dedicated nodes provide all persistent storage. We take a unique approach to discovering and modeling I/O bottlenecks (§4.2) as a result. Wieder et al. [8] do consider data staging and migration costs in their performance model, but do not account for the effects of disk contention and I/O load imbalance on whole-cluster performance. This may be a side-effect of their evaluation using only a k-means workload, which computes 5.33GB/hr of input data for the whole cluster. In contrast, our test workloads of Grep and Wordcount compute roughly 210GB/hr and 140GB/hr respectively, using 6 dedicated nodes (comparable to their 5-node local cluster). Further, although their models can adapt to changes in SPOT price, they do not address the substantially-increased replication overhead in the case of SPOT arrivals/departures, which was a significant motivation for our choice of a hybrid cluster design.

RHIC can optimize novel and short-lived jobs (which are common [13, 7, 14]) with no *a priori* knowledge, using a combination of online profiling and adaptive, guided scaling (§4.2). All prior efforts require either previous executions of the target job [7, 10, 9] or key performance characteristics [8]. While those with online adjustment [7–9] could adapt to some deviation from the profile performance (as Wieder et al. [8] demonstrate), the pervasive and dynamic nature of volunteer heterogeneity may necessitate RHIC's online learning and reactive approaches to CPU (§4.2) and I/O (§4.3).

Finally, RHIC offers a highly-generic performance modeling interface, which only requires a job progress score and average task length (§4.3). The models employed by prior works have various levels of dependency on the target workload, from MapReduce as a concept [8, 9] to specific MapReduce implementations [10, 7]. Because we envision RHIC as a harvesting platform which manages a throughput-oriented parallel batch job, we built it with to be workload-independent and evaluate this capability (§5.5). Further, because volunteers are lightweight and transient, we believe RHIC could be applied to multi-stage jobs [7] by managing each stage independently.

Parallel Filesystem Resizing. Lim et al. [16] use a control-theory approach to elastically resize an HDFS cluster, considering oscillation prevention and the tradeoff between improved performance and rebalancing overhead. While these factors are potentially-relevant to our problem, this work does not consider any heterogeneity among participating nodes, and is designed for workloads which are SLO-oriented instead of throughput-oriented. Further, Lim's tradeoff function between rebalancing bandwidth and response time penalty is achieved via manual tuning, and is specific to their workload and cluster hardware.

Hybrid MapReduce, Volunteerism and Cluster Sharing. Prior works use Amazon EC2 Spot Instances to perform MapReduce jobs [4, 5, 17], whose transience is similar to interactive cloud nodes. Two approaches are taken to handle SPOT instance instability: (1) using SPOT instances to supplement a core set of dedicated, non-SPOT nodes [4, 5], and (2) using Amazon's cloud storage service to preserve intermediate results [17]. Our approach is most-similar to the former, in that robust aggregated storage is unavailable in our environment and a hybrid cluster design is necessary to provide stability. Both of these works [4, 5] elect to host data only on core nodes, but do not consider the performance impact of I/O in such an offloading scenario. Although Lee et al. [4] highlight a similar problem space to our work, they have not proposed any concrete solution for automatically modeling workload performance or determining ideal cluster size.

MOON [3] is a modification of Hadoop designed to operate under passive volunteerism, where a foreground workload and MapReduce are interleaved temporally but not spatially. MOON makes a number of platform-level modifications to improve I/O performance and reduce stragglers in this scenario. Mesos [13] is a framework for batch framework co-location above a shared distributed filesystem, which leverages an offer-based scheduler to enforce fairness. Mesos uses lightweight virtualization to guarantee each slot a static resource allocation. Both works do not consider active volunteerism, where two workloads are asymmetrically sharing the same resources at the same time, or use cluster sizing meet user-defined goals such as deadlines or cost.

Workload Consolidation. Co-locating workloads on the same physical host is a well-established technique [11, 12] which offers benefits for the cluster host including improved utilization, better energy efficiency and higher profits. We believe that our harvesting approach is complementary to workload consolidation, due to the level of overprovisioning required by interactive users. RHIC can transparently harvest whatever residual resources are available after consolidation, with the expectation that the bursty interactive user will leave some free during periods of “think time”.

3 Background

As mentioned above, we leverage a *hybrid* cluster design to harvest residual resources. This platform was proposed by prior works, which examined MapReduce running on *passive* volunteers [3] and EC2 SPOT instances [4, 5]. In Section 3.1, we justify our choice by showing that this cluster design is appropriate for our environment. Further, in Section 3.2 we discuss techniques to make the hybrid cluster effective in our environment, using a combination of hypervisor mechanisms and I/O offloading. Finally, in Section 3.3 we present a preliminary evaluation of the energy and cost benefits of a hybrid cluster relative to a traditional dedicated cluster.

3.1 Hybrid Cluster Rationale

Table 2: **CPU consumption, burst and reservation characteristics collected from NCSU’s VCL.** CPU data are collected from real user session traces (to be described in more details in Section 5.1). Reservation data cover all sessions during the years 2004-2010.

Metric	**Matlab**	**Photoshop**	**OpenOffice**	**C Dev**
CPU Consumed (avg)	19.8%	7.0%	2.8%	22.5%
CPU Consumed (stdev)	23.2%	16.2%	12.4%	24.27%
CPU burst height (avg)	39.9%	25.8%	31.0%	27.7%
CPU burst length (avg)	6.9 sec	2.0 sec	1.3 sec	47.4 sec
Reservation (avg)	93 min	74 min	70 min	120 min
Reservation (stdev)	90 min	79 min	91 min	99 min

Our hybrid approach is based on the characteristics of interactive cloud workloads observed on VCL, the aforementioned production cloud system. Table 2 summarizes statistics information collected from VCL remote desktop sessions. It shows that while reservations (user sessions) are fairly long, their durations have a very high standard deviation, meaning the length of any given reservation is highly unpredictable. Meanwhile, CPU bursts are quite short-lived, even for the more computation-intensive workloads such as Matlab.

Such highly dynamic behavior renders traditional approaches such as workload consolidation [18, 12] less appealing. Conservative consolidation approaches will be able to maintain interactive users’ QoS requirement, but will inevitably waste resources. Aggressive approaches, on the other hand, may face severe performance penalties in case of requirement conflicts. In particular, small-to-medium clouds such as the VCL often do not offer shared storage to enable live VM migration. Offline migration typically requires minutes to complete the image transfers. Even with shared storage, the short CPU bursts and highly variable session durations seen in interactive workloads will require frequent live migration and may lead to heavy thrashing.

In the hybrid cluster design, the dedicated nodes have node-local storage capacity, while the volunteer nodes only use their local storage for temporary data. This design addresses the dynamic and unreliable nature of residual resources, unused by interactive tasks, in several ways. First, it keeps volunteer nodes lightweight and agile. This makes it much easier to use/discard a node due to foreground interactive load shifts, as well as to grow/shrink the virtual cluster based on the background job's needs. Second, expensive tasks performed by the underlying distributed file system, such as replication and data rebalancing, will not be unnecessarily performed on volatile volunteers. Third, through mechanisms such as task replication and reliable dedicated nodes, this hybrid design can aggressively harvest dynamic residual resources while preventing stragglers from severely delaying job completion.

When MapReduce is used as the background computation framework, only the Map tasks are outsourced to the interactive nodes, due to the high cost of lost Reduce tasks [19, 5]. Each volunteer node buffers intermediate results and periodically spills to Reduce tasks running on the dedicated nodes.

3.2 Applying the Hybrid Cluster Design

For this hybrid cluster design to work in our target scenario, we need to verify two assumptions, as discussed below.

(1) *Existing hypervisors provide effective prioritization that keeps foreground interactive loads isolated from volunteers.* To verify this, we examined the effectiveness of relying on the work-conserving schedulers in Xen and KVM, two widely used open-source hypervisors, when co-locating foreground and volunteer VMs on the same physical host. Our results are shown in Figure 2 and indicate that the Xen credit scheduler in particular appears quite effective at preserving interactive performance, with an average of foreground slowdown of 1%. Although KVM lags behind in this area, slowdown is less than 7% in all cases, even when volunteer disk I/O load was high due to large volumes of temporary data. Our conclusions in this area are backed by previous studies of hypervisor isolation [20].

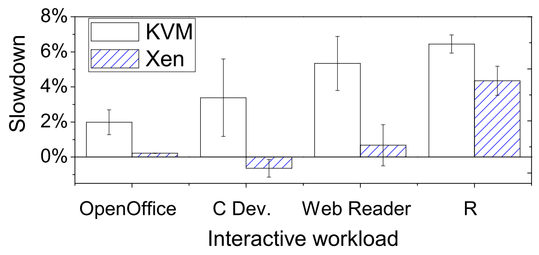


Fig. 2: **Slowdown of interactive workloads padded with MapReduce volunteers** running Word Cooccurrence (high CPU and large intermediate data). Interactive workloads taken from the Linux `bltk` and AT&T's R benchmark.

(2) *When running our target background workloads, the dedicated nodes have sufficient residual disk and network bandwidth to act as storage servers, supporting computation offloading to a substantial set of volunteers.* To verify this, we compared the availability of disk and network bandwidth when running a background MapReduce job on 2 dedicated nodes alone and when supplementing these nodes with 8 volunteer nodes

(with no foreground workloads, to create maximum remote I/O pressure). Figure 3 plots the disk and network utilization level (collected with the `iostat` and `dstat` tools respectively) for the two most I/O-intensive workloads in our MapReduce background test set, Wordcount and Grep. It illustrates that (1) substantial disk and network bandwidth is available on each Hadoop node executing MapReduce jobs, (2) using volunteers significantly speeds up the job execution while increasing I/O bandwidth utilization, and (3) disk bandwidth consumption is significantly higher than that of network, and may become a performance limiting factor when more volunteers are used.

Our later evaluation (Figures 9-10) further corroborates this conclusion. Wordcount, which is a moderately I/O-intensive workload, reaches its cost/energy minimums at a 3:1 ratio of volunteers:dedicated nodes on our test cluster, which contains nodes with 16 cores and one SATA II disk. Therefore, at the ideal cluster size, each SATA II disk supports 64 Wordcount Map tasks (one per core). Grep, which is much more I/O-intensive, reaches its minimums around a 1:1 ratio, and therefore supports 32 Map tasks per disk.

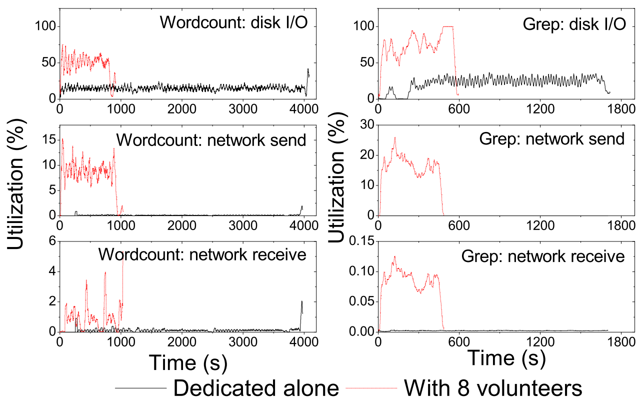


Fig. 3: **Bandwidth utilization traces** on 2 dedicated nodes, with and without 8 volunteers, for the duration of a MapReduce job.

3.3 Proof of Concept: Hybrid Cluster Energy and Cost

To verify the the energy and cost profit enabled by our proposed hybrid cluster approach, we experimented with 2 dedicated nodes and a varying number of volunteers (2 to 8). Figure 4 shows the monetary and energy cost savings when running our background workloads on the hybrid cluster, as compared to using a regular Hadoop cluster with the same number of nodes (dedicated plus volunteers). For example, we directly compare 2 dedicated plus 2 volunteers to 4 regular nodes. Each bar shows the average results over tests with different volunteer counts (each with multiple trials), with the error bar depicting the *range* of results. The foreground workload on volunteer nodes is Photoshop. This set of experiments used nodes from NCSU's HGCC cluster that have a relatively low idle power level of 34%, which is less favorable to hybrid cloud computing. As can be seen here, our hybrid cluster design is indeed able to deliver significant cost savings: 20-29% energy and 20-40% monetary cost reductions on average across all four background workloads.

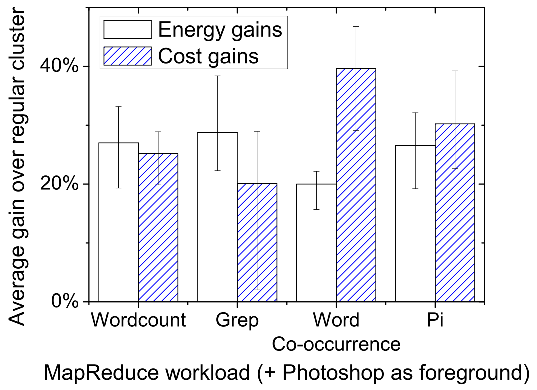


Fig. 4: **Average energy and cost savings by using a hybrid cluster design.** Error bars in this chart represent the range of savings instead of standard deviation.

Interestingly, Wordcount and Grep have higher energy savings, while the other two applications see more monetary cost savings. This is primarily due to the fact that Wordcount and Grep are I/O-bound and do not scale as well as the two compute-bound workloads. For compute-bound workloads, volunteers are much cheaper, slightly weaker replacements for dedicated nodes, but operate at a higher incremental energy consumption. For I/O-bound workloads, cost savings are in general lower, as the cheap volunteers cannot help with performance as substantially by offering much-needed I/O bandwidth. In an extreme case, where Grep uses 2 dedicated and 8 volunteer nodes, severe I/O bottlenecks reduces the monetary cost gain to 2%. Energy wise, however, our hybrid cluster design is more tolerant to I/O-bound workloads, as CPUs blocked by I/O consume little incremental power.

In addition, as illustrated by the error bars, even with volunteers running homogeneous foreground workload types, there is a considerable room to optimize for cost via cluster sizing.

4 Resource Harvesting Framework Design

4.1 Overview

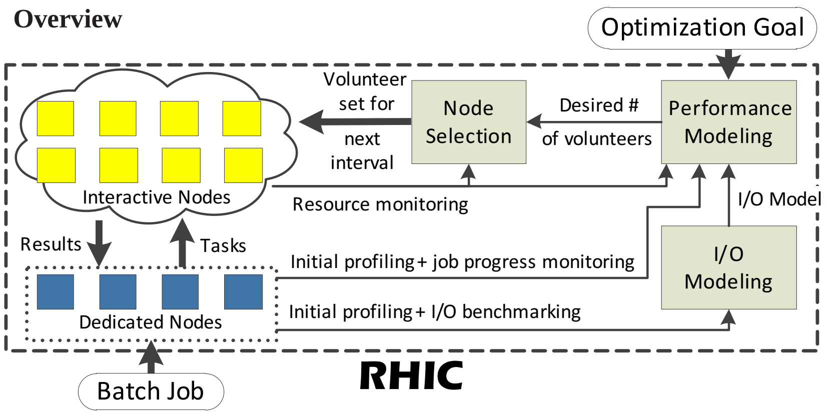


Fig. 5: **RHIC components and data flow**

RHIC combines online profiling with periodic job progress and system resource monitoring, to adaptively scale the volunteer node set throughout a *background* (batch) job's execution. Figure 5 shows RHIC's major components, which collaborate to periodically re-evaluate the batch job's performance and make cluster sizing decisions.

RHIC starts a batch job execution with a profiling phase, where the dedicated nodes run alone. This allows us to seed our I/O model by viewing the background job running without I/O pressure generated by the diskless volunteers. This profiling data enables two techniques which we discuss in detail in §4.2: (1) dedicated disk I/O bandwidth utilization lets us determine core parameters for our initial scaling algorithm and (2) dedicated CPU consumption allows us to avoid extrapolation for small cluster sizes, which are critical for I/O-heavy workloads. In addition, throughout the entire job execution, RHIC continues to monitor system status information, such as interactive node resources, dedicated node I/O saturation, and job progress. With the initial profiling and the continuous monitoring, respectively, RHIC automatically observes and adapts to both the background job's behavior and changes in the foreground workload.
After this initial profiling phase, RHIC iterates between the following steps:

1. Profile interactive and batch VMs for the length of the *evaluation interval*
2. Update models with profiled data:
   (a) I/O models (§4.2)
   (b) Job progress model (§4.3)
   (c) Interactive resource models (§4.4)
3. Use updated models to find the best cluster size, by repeatedly:
   (a) Choose a cluster size to evaluate
   (b) Choose the "best" volunteers to meet the desired cluster size (§4.4)
   (c) Model I/O pressure under the potential cluster composition (§4.2)
   (d) Estimate completion time and performance (§4.3)
4. Select the best-performing cluster size and actuate it (§4.4)

In our prototype implementation, we set the initial dedicated-only profiling phase to be one minute, the continuous resource and job progress monitoring frequency to be once a second, and the cluster resizing evaluation frequency (evaluation interval length) to be once a minute. With our moderate testbed (6 dedicated and 36 interactive nodes), RHIC can exhaustively evaluate all possible volunteer counts (0-36) in $250ms$. However, for scalability, we have also implemented an alternative search module using simulated annealing.

To handle the dynamic set of interactive nodes, each contributing varying amount of resources, and to achieve online performance modeling independent of the actual workload and even batch execution framework, RHIC relies on three key insights derived from our experiments. These insights, as listed below, help us to simplify our performance model, identify chief performance constraints, and focus on the behavior of aggregate resources from volunteers:

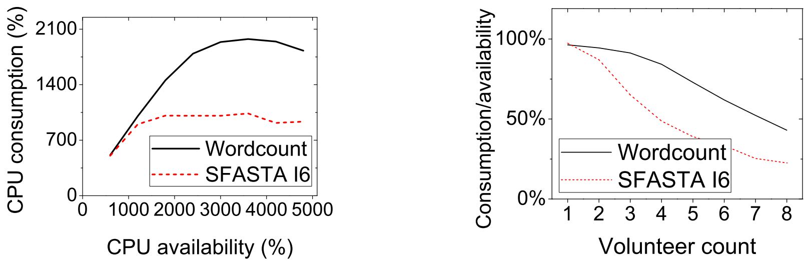
– ***Insight 1:*** In our proposed hybrid execution mode, the disk I/O bandwidth afforded by the dedicated nodes is a major factor limiting the *effective* CPU resource contribution of a volunteer node.

- ***Insight 2:*** The overall progress of a batch job is determined by the *aggregate* CPU contribution from all selected volunteers, independent of the contribution distribution among these nodes.
- ***Insight 3:*** Although each foreground interactive workload has unpredictable resource usage bursts, its *average* usage amortized over a longer period of time tend to be much more stable and hence predictable.

In the rest of this section, we discuss in detail the above insights and several major RHIC components as well as their interactions. Note that for simplicity, discussion is based on homogeneous hardware across the node pool. Given that most cloud systems have a limited number of hardware types, our profiling and modeling can be easily adapted to enumerate performance/cost behavior of different hardware.

4.2 Modeling Workload I/O Behavior

Our proposed harvesting method is based on the observation that, for typical distributed batch execution model such as MapReduce, there is residual I/O/network bandwidth for dedicated nodes to support additional volatile, diskless volunteer nodes. We have verified this with experiments §3. However, as the number of volunteers grows, eventually the I/O bandwidth on the dedicated nodes is likely to become the chief performance/scalability limiting factor (Insight 1). This constraint has not been considered in related prior work [4, 5]. RHIC, in contrast, builds an I/O model at runtime for the target batch job to identify the existence of I/O bottlenecks.



(a) CPU availability vs. consumption (b) CPU consumption/availability ratio

Fig. 6: **Impact of I/O bottlenecks on effective volunteer CPU contribution**, using 2 dedicated and 1-8 volunteers, with a maximum of 8 cores (800% CPU) on each.

Figure 6 illustrates the interaction between the volunteer CPU contributions and the I/O contention at the dedicated nodes for two sample MapReduce workloads. It shows the aggregate *effective CPU contributions* from the volunteers at each aggregate residual CPU availability level, averaged over the Map phase (excluding stragglers). The effective contribution is measured from the background VM usage, while the availability calculated from the foreground VM usage. We verified that the leveling off point in these curves corresponds to the dedicated node I/O saturation point. This figure also demonstrates that the onset of the I/O saturation is highly workload dependent. With a more I/O-intensive workload (SFASTA in this case), the saturation comes earlier and results in a lower effective CPU contribution rate. Figure 6b plots the effective CPU contribution to availability ratio over different volunteer counts. It illustrates that the MapReduce job consumes a constantly declining portion of the aggregate CPU resources available. As a result, we base our I/O modeling on $\{CPU_{consumed}, CPU_{available}\}$ pairs for the given workload and hardware, derived at runtime.

Saturation Point Estimation For each background job, RHIC builds an I/O curve that tracks available CPU on the X-axis and consumed CPU on the Y-axis, to predict the effective CPU contribution for a given volunteer set. RHIC uses data from the initial profiling, as well as continuous sampling, and applies regression to build this I/O curve.

To avoid the inaccuracy caused by extrapolation or sampling well beyond the I/O saturation point, it is important to estimate an approximate location of the dedicated node I/O saturation onset, in terms of the aggregate volunteer CPU contribution. The saturation point also indicates the upper bound of volunteers needed, regardless of optimization goal, as beyond this point more volunteers will not return additional performance. RHIC bases its I/O saturation point estimate on I/O bandwidth consumption data collected in the dedicated-only initial profiling phase. Assuming a linear relationship between CPU contribution and I/O demands, it calculates the number of volunteers each dedicated node can support:

$$Volunteer\,CPU_{supportable} = (\frac{100\%}{BWUtil_{Avg}} - 1) \times CPU_{max} \tag{1}$$

Here $BWUtil_{avg}$ is the average disk bandwidth utilization measured on the dedicated nodes during the profiling phase, and CPU_{max} is the maximum CPU available on any node (such as 400% for 4 cores). $Volunteer\,CPU_{supportable}$ is the volunteer CPU contribution *each* dedicated node could support, in addition to its own demand.

Next, we calculate the range of potential I/O saturation onset points, using best and worst-case estimates. The best-case estimate represents completely-balanced I/O load (each dedicated node serving equal volunteer demand) and the worst-case completely-imbalanced (one dedicated node serving all volunteer demand). Below we derive the pair of estimates based on $Volunteer\,CPU_{supportable}$, where $Count_{dedicated}$ is the number of dedicated nodes:

$$Saturation_{best} = Volunteer\,CPU_{supportable} \times Count_{dedicated} \tag{2}$$

$$Saturation_{worst} = Volunteer\,CPU_{supportable} \times 1 \tag{3}$$

Using the best and worst case estimates above, RHIC intelligently increases the size of the volunteer pool using Algorithm 1. It samples the worst case estimate and halfway between the best and worst case, then uses linear regression to guess the actual saturation onset point. RHIC then verifies the occurrence of I/O saturation using the disk sensors on the dedicated nodes, based on the disk bandwidth utilization metric from the `iostat` utility. In practice, we have found that this approach quickly finds the I/O saturation point with satisfactory accuracy. In addition, this allows us to sample system metrics under a range of cluster sizes, improving the breadth of our models. This approach is unnecessary for deadlines, because the deadlines must be achievable on the "near" side of the I/O saturation point.

I/O Curve Building with Clustering and Curve-fitting Next, we complete the I/O curve that maps aggregate volunteer CPU availability to aggregate effective volunteer CPU contribution. RHIC uses Mean-Shift clustering [21] to pre-process raw $\{CPU_{consumed}, CPU_{available}\}$ data points. This allows us to avoid a critical flaw in

Algorithm 1 Initial pool scaling algorithm used for cost and energy minimization. The cluster is pushed to the saturation point using a combination of Eq. 2-3 and first-degree regression (line 18). This approach also accounts for the effects of diminishing marginal returns (DMR) when I/O saturation is not the primary limit on scalability, which is often the case when the price of volunteer is relatively high.

```
1: // Perform one profiling period with dedicated nodes alone
2: $profileData \leftarrow$ RunPeriod($\emptyset$) , $Period \leftarrow 1$
3: // Calculate saturation range using Eq. 2-3
4: $(Saturation_{worst}, Saturation_{best}) \leftarrow$ ComputeSaturationPoints($profileData$)
5: // Predict the ideal, ignoring I/O saturation scalability limitations, as a ceiling
6: $Max\,Volunteers_{DMR} \leftarrow$ PredictIdealIgnoringIO($profileData$)
7: // Push to the I/O saturation point, accounting for diminishing marginal returns
8: while (IOSaturation()$\neq True$) AND ($Volunteers < Max\,Volunteers_{DMR}$) do
9:    if $Period$ == 1 then
10:       $nextSize \leftarrow min(Saturation_{worst}, Max\,Volunteers_{DMR})$
11:   else if $Period$ == 2 then
12:       $nextSize \leftarrow min(\frac{Saturation_{best}+Saturation_{worst}}{2}, Max\,Volunteers_{DMR})$
13:   else if $Period$ == 3 then
14:       if $Max\,Volunteers_{DMR} < Saturation_{best}$ then
15:           $nextSize \leftarrow Max\,Volunteers_{DMR}$
16:       else
17:           // Linear extrapolation using the $BWUtil$ observed in periods 0-2
18:           $Volunteers \leftarrow$ ExtrapolateSaturation()
19:       end if
20:   else
21:       $nextSize \leftarrow$ size($Volunteers$) $+Saturation_{worst}$
22:   end if
23:   // Find volunteers to satisfy the next size, and run the period
24:   $Volunteers \leftarrow$ FindVolunteers(nextSize)
25:   RunPeriod($Volunteers$) , $Period$ += 1
26: end while
```

using curve-fitting for decision making, where incorrect decisions reinforce themselves by repeated sampling in the same area of the curve, skewing R-squared summations. Also, clustering allows us to tolerate changes in the I/O landscape, such as increased Reducer disk I/O load in MapReduce, by *aging* data points.

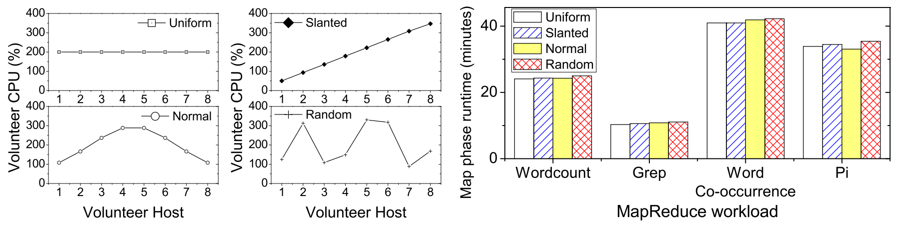
Finally, RHIC performs first-degree spline fitting on the cluster centers to build the I/O curve. This approach allows us to deliver interpolated values tightly constrained to the observed curve. This is important, especially near the saturation point, because minimization decisions hinge on marginal cost/gains. Prediction of the effective aggregate CPU contribution from a given set of volunteers can then be performed with interpolation, based on the projected aggregate CPU availability from these volunteers.

This approach assumes that the network bandwidth (as well as third-party network contention) is either static or is not a limiting factor. We believe such an assumption is practical based on network bandwidth consumption measurements (§3) and experience with VCL and several other 100+ node clusters. We plan to relax this assumption in the future, using bandwidth monitoring and topology awareness.

4.3 Background Job Performance Modeling

Background job performance modeling is the core of RHIC's sizing intelligence. It relies on two mechanisms: (1) the I/O modeling presented in § 4.2, and (2) CPU availability predictions for candidate interactive nodes (to be discussed in § 4.4).

As mentioned earlier, RHIC's performance modeling is based on the observation that the aggregate CPU contribution from the selected volunteers, regardless of the distribution of CPU resource availability on individual volunteer nodes, is the chief factor determining a job's completion time on a hybrid cluster (Insight 2). Figure 7 shows experimental results demonstrating this performance behavior. In these tests, we collected the execution time of four MapReduce workloads under four different CPU allocation distributions among the volunteers. According to each given distribution, a volunteer is allocated 4 cores with a CPU cap between 50%-350% (with one core's entire CPU resource counted as 100%), while the total CPU allocations from all 8 volunteers are fixed at 1600%. Figure 7a illustrates the shape of the distributions used. Figure 7b shows that the duration of the Map phase is nearly constant across all distribution types, for all MapReduce workloads tested. In other words, frameworks like Hadoop are quite tolerant to heterogeneity in node processing capabilities, possibly due to the adoption of mechanisms such as the well-proven LATE algorithm [22].



(a) Volunteer CPU distributions (b) Completion times

Fig. 7: **Impact of residual CPU resource distribution on MapReduce job completion time.** Shown for 2 dedicated and 8 volunteer nodes.

The above observation allows us to build our performance (and consequently cost) modeling on the **collective** behavior of the dynamic interactive nodes. Rather than micro-managing volunteer nodes according to their foreground resource usage bursts, RHIC bases its decision on the aggregate, sustained CPU availability from candidate volunteer node sets.

Completion Time Estimation and Transition Damping More specifically, RHIC predicts that "a background job will complete at time y if it receives a sustained total volunteer CPU contribution of x". For this, we developed a simple model based on the processing rate R_{proc}, shown in Equation 4. Here $J_{completed}$ is the current fraction of processing completed, $T_{elapsed}$ is the time elapsed, and $CPU_{consumed}$ is the effective aggregate CPU consumption (including both dedicated and volunteer contributions) over $T_{elapsed}$. R_{proc} is re-evaluated periodically during the background job execution.

$$R_{proc} = \frac{J_{completed}}{T_{elapsed} \times CPU_{consumed}} \tag{4}$$

By calculating the fraction of remaining work $J_{remaining} = J_{total} - J_{completed}$, we can then invert Equation 4 and produce a Map completion time estimate $T_{remaining}$, given a predicted aggregate CPU contribution $CPU_{predicted}$, as shown in Equation 5. $CPU_{predicted}$ is calculated by applying the I/O model (§4.2) to the predicted aggregate available CPU (§4.4), which estimates the aggregate volunteer CPU which is *sustainable* by the dedicated I/O infrastructure. Finally, to tolerate stragglers, we add a small padding value to our completion time estimate, based on the average length of background tasks experienced thus far.

$$T_{remaining} = \frac{J_{remaining}}{CPU_{predicted} \times R_{proc}} \tag{5}$$

To avoid oscillation or thrashing, we estimate the transition time required by a volunteer pool size change. Volunteer additions require a fixed setup overhead, which we profile. Volunteer removals are trickier - as we allow deselected volunteers to *drain* running tasks when we remove them. We predict the draining duration based on the observed average volunteer task length. In both cases, the transition time is accounted for in making completion time predictions.

Goal Estimation Based on the completion time estimate, RHIC generates performance scores (to be minimized) for candidate volunteer sets, given one of the three goals it currently supports:

(1) *Deadlines:* To satisfy a deadline requirement, RHIC computes the performance score as the difference between the estimated job completion time and the deadline.

(2) *Monetary cost:* With pay-as-you-go cloud computing, volatile volunteer nodes with no CPU resource guarantee are likely to be charged at a lower rate. Given a certain pricing policy defining such discounts, RHIC calculates the performance score as the predicted overall cost based on the completion time estimate.

(3) *Energy:* Energy estimation is more complex and requires the offline construction of an energy model for the specific hardware used. In this paper, we focus exclusively on CPU power consumption, considering prior findings that CPU typically dominates energy consumption in modern systems [23]. Our energy modeling takes the well-established approach of running a micro-benchmark that thoroughly enumerates the relationship between CPU utilization, frequency and power consumption [23, 24]. Based on the data collected, we use multiple regression methods to derive a power model that estimates power consumption at an arbitrary utilization and frequency level. This model model is subsequently used by RHIC to compute the performance score as the predicted power consumption with the given volunteer set, over the length of the job.

Recall that our hybrid cluster design is partially motivated by the energy savings allowed by piggybacking background workloads on interactive foreground tasks. While all power consumption on dedicated nodes is billed to the background user, he/she is only responsible for the *incremental* energy consumption incurred by the background job on the volunteer nodes, because these nodes would not be powered on otherwise. Therefore, in modeling the background job power consumption, we exclude the baseline (idle) power consumption as well as the predicted foreground power on volunteers.

4.4 Volunteer Node Selection and Management

Finally, given a desired aggregate volunteer set size, RHIC must select which specific interactive nodes to use in an efficient and scalable manner. This selection is based on continuous interactive node residual resource monitoring and prediction, as discussed below.

As shown earlier, common interactive cloud workloads have highly bursty behavior, making load consolidation backed by VM migration difficult. However, for running background jobs that yield to the interactive foreground tasks, it is the sustained CPU resource availability that matters. Fortunately, we found that although individual CPU usage spikes appear random and unpredictable, the average CPU utilization over a period of time can be effectively estimated using near-term history data (Insight 3).

Recognizing this, RHIC employs an online foreground workload CPU demand model using the CPU consumption readings from the interactive nodes sampled once a second. We considered four common prediction methods: moving average, auto-regression, auto-correlation, plus a hybrid of signature-based Fast Fourier Transform (FFT) and Markov chains used in our previous work [25]. We evaluated all four under a range of conditions which simulate our intended environment: 10, 20, 30 and 60 minutes of history, and 5 and 10 minutes of lookahead (prediction window). These conditions were chosen because we desire more than just a short lookahead, but simultaneously do not expect a long history to be available due to interactive node transience.

Figure 8 shows the accuracy of these four prediction methods, and moving average yields the most-accurate predictions, most likely due to the short training window. Moving average and auto-correlation show identical performance, but this occurs because auto-correlation falls back to a moving average when it is unable to achieve a match. As a result, we have selected moving average as our prediction algorithm, and we maintain a prediction model for each interactive node regardless of whether it is currently selected as a volunteer.

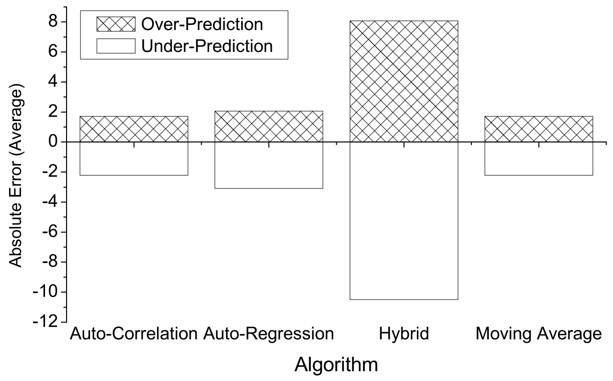


Fig. 8: **Accuracy of four different prediction algorithms** for the foreground traces which we use. Absolute error is shown, with a value range of 0-100.

For memory, we assume that the foreground VMs have pre-specified memory caps based on their workload, as in the case of Amazon EC2 and VCL instances. Background memory usage, on the other hand, is estimated during the initial profiling phase.

In selecting specific volunteers from the interactive node pool, we adopt a greedy algorithm for better scalability. First, we perform screening and predicted CPU contribution adjustment based on memory constraints, according to control interfaces exported by the background execution platform. For MapReduce-like platforms, we adjust the number of simultaneous worker processes (such as Map slots) on each volunteer to fit within its residual memory capacity. If this kind of performance knob is unavailable, we instead discard any nodes which do not have the minimum memory required. Next, candidate nodes are sorted according to their adjusted residual CPU availability level. Finally, RHIC makes volunteer selections by evaluating different prefix sets of the candidate list toward a given optimization goal using the I/O-aware performance model introduced earlier. If the current volunteer set is no longer optimal, adjustment is made by including nodes with the highest or discarding nodes with the lowest predicted CPU contribution. Intuitively, this approach reduces the number of volunteers used, contributing to lower overall monetary and energy cost. Also by performing this greedy selection, the search is limited to a linear rather than exponential space, in regard to the candidate interactive node pool size.

In addition, we use a periodic threshold-based “replacement” process to identify and replace volunteers that experience a significant decrease in residual CPU availability. This is necessary because our node selection algorithm only discards nodes when RHIC chooses to lower the volunteer count. To do this, we periodically perform iterative checking by comparing the most-available unused node with the least-available used one. If the difference in their CPU availability is above a threshold, we replace the used node with the unused node. This process is repeated till the CPU availability difference falls under the threshold.

Interactive node churn presents an issue for our search-driven cluster sizing scheme, because interactive nodes can arrive/leave unexpectedly and change the ideal batch cluster size. In such a situation, a naïve response would be to perform another round of searching to find the best cluster size, in light of the altered interactive pool. However, because interactive nodes can leave *en masse*, i.e. at the end of a class lab, there exists the possibility of significant thrashing by the search process as it tries to react to a series of successive arrival/departure events. To prevent the search process (§4.1) from causing significant overhead under high interactive node churn, we take a *deferment* strategy. When a node arrives or leaves, we attempt to enforce the decision made at the end of the last evaluation interval, deferring new decisions to the end of the current evaluation interval.

5 Experimental Evaluation

In this section, we evaluate RHIC in five key areas after giving an overview of our test platform in Section 5.1. First, in Section 5.2, we establish that RHIC can accurately discover and achieve near-ideal cluster sizing decisions under a given optimization goal, in comparison to an exhaustive search of possible cluster sizes. in Section 5.3, we then compare the performance, stability, and adaptability of RHIC to an alternative algorithm, based on fuzzy control theory. Next, we validate RHIC’s performance under increased cloud instability in Section 5.4. While all the above experiments uses Apache Hadoop [26] as the background cluster platform, in Section 5.5 we show that in addition RHIC is general enough to harvest excess resources using a lightweight compute layer on top of HBase [27]. Finally, we briefly discuss RHIC's overhead in Section 5.6. Unless otherwise noted, we run each test three times, report the average, and show error bars denoting the standard deviation when it is above 2% of the mean.

5.1 Test Workloads, Platform, and Settings

Background Workloads For evaluating RHIC, we use Hadoop and and a thin compute layer running over HBase as the background job execution frameworks. Hadoop and HBase are widely used open-source implementations of the Google MapReduce and Big Table systems, respectively.

We used four representative MapReduce workloads: Wordcount (counting word appearances), Grep (regular expression search), Word Co-occurrence (counting word pair appearances), and Pi (calculating π using a quasi-Monte Carlo algorithm). Grep and Wordcount process 70GB of data, Word Co-occurrence processes 11GB, and Pi requires no input data. Job execution times are typically between 20 and 40 minutes. We study the execution of the Map phase, which for the all of background workloads dominates the total MapReduce execution time. This is backed by findings from production clusters that Map-only jobs are common, the Map phase dominates MapReduce jobs, and input data is the majority of stored bytes [28].

We used two representative workloads on top of HBase: Compress (offline LZO compression of text cells), which is I/O-intensive, and Raytrace (image generator), which is CPU-intensive. Both are likely to run during off-peak hours against semi-structured data stored in a production HBase cluster, and hence are suitable for throughput-oriented volunteer harvesting. Compress could conceivably be used on user messages, profile data, wall posts etc., motivated by high compression costs which cannot be borne by frontend servers during peak hours. Raytrace is representative of image/tile generation workloads for multiplayer games creating randomly-generated worlds, such as Minecraft [29].

Foreground Workloads We used four popular VCL workloads: Matlab, Photoshop, OpenOffice, and C Development. We reproduced the foreground VM execution based on real session traces collected from VCL [30]. The length and *churn rate* of the foreground VM sessions are generated using a normal distribution with the parameters derived from 2004-2010 VCL log data. Finally, we set static memory allocations for foreground VMs using per-workload-type normal distributions extracted from VCL reservation logs. Because our foreground load generation is based on traces, we can repeatedly generate identical loads using the same traces and random number seeds.

Test Platform Our main test platform is NCSU's ARC cluster [15], which, unlike current public clouds, allows us to perform power measurements. It has 108 nodes interconnected via InfiniBand, each with 16 2GHz cores on two processors, 32GB RAM, a SATA disk drive and the KVM hypervisor. We use Infiniband for our experiments, but due to KVM's virtualization overhead, we can only achieve approximately 500 MBit/sec speeds (VM to VM). We restrict dedicated VMs to 16GB of RAM, while foreground and volunteer VMs share 8GB RAM total.

To calculate background power consumption, we replay the same foreground workload by itself and calculate the difference. For monetary cost evaluation, unless otherwise noted, we adopt a sample pricing policy following the relative costs of EC2 `m2.xlarge` On-Demand and Spot Instances at the time of writing. This sets the per-node rate to $1.00/hour for dedicated nodes and $0.42/hour for volunteer nodes.

5.2 Exhaustive Evaluation

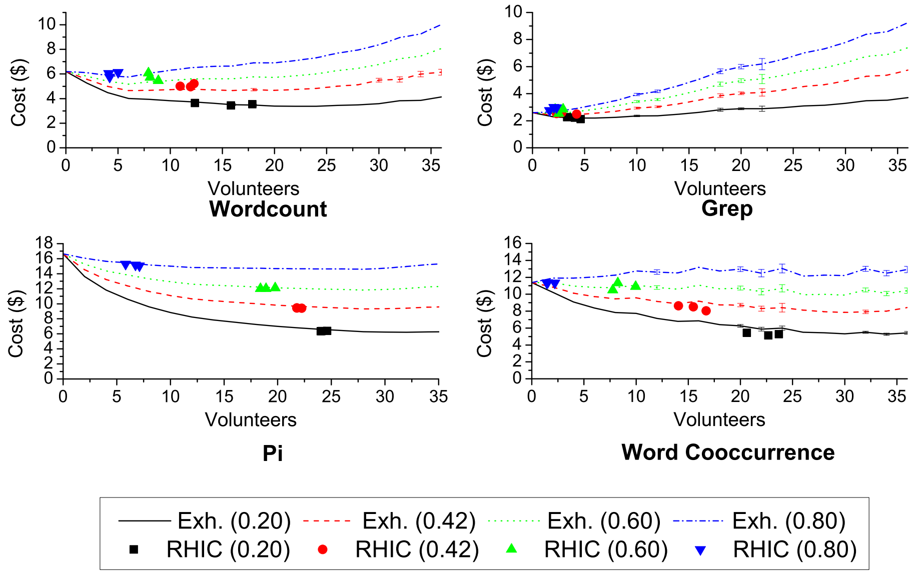


Fig. 9: **Monetary cost minimization results with different pricing policies.** Dedicated nodes are fixed at $C_d = \$1.00/hr$, with four different volunteer rates: $C_v = \{\$0.20/hr, \$0.42/hr, \$0.60/hr, \$0.80/hr\}$, represented in the key as Exh.(C_v) and RHIC(C_v).

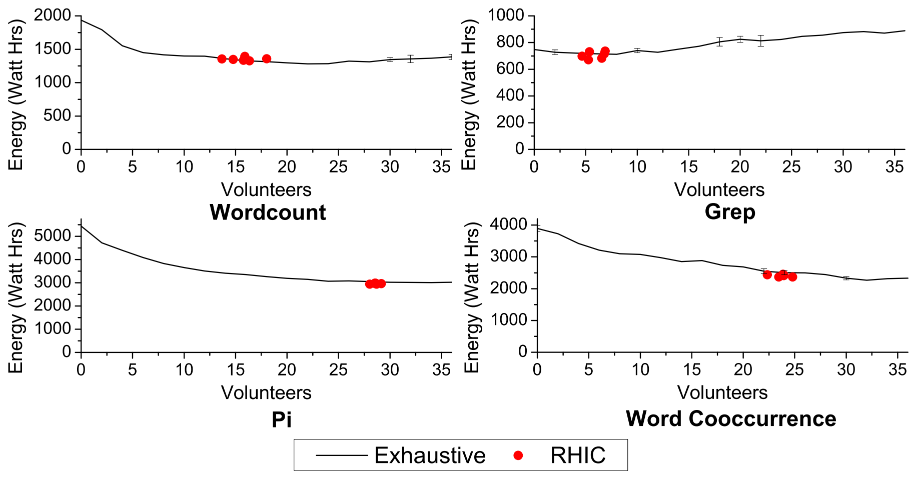


Fig. 10: **Energy minimization results.** Energy data collected from power meters attached to the ARC cluster.

First, we performed an exhaustive evaluation over the valid volunteer cluster size range, for each of our four MapReduce test workloads (Wordcount, Grep, Pi and Word Co-occurrence). We then ran RHIC under identical conditions to verify its ability to quickly find the ideal cluster size. Our hybrid cluster is composed of 6 dedicated nodes and 0-36 volunteers. We collected exhaustive datapoints every 2 volunteers, from $v = \{0, 2, 4, \ldots, 36\}$, and repeated each test twice.

For a fair comparison, we ensured that every run (exhaustive or RHIC) had an identical foreground workload "mix", composed of the same traces starting the same points in time. This mix is composed of a randomized selection of traces and start points taken in equal proportion from each of the 4 foreground workloads described in Section 5.1: 25% each from Matlab, Photoshop, OpenOffice and C Development. This seeded mix allowed us to collect foreground-only energy consumption and subtract it from the total, for generation of the background energy curves shown in Figure 10.

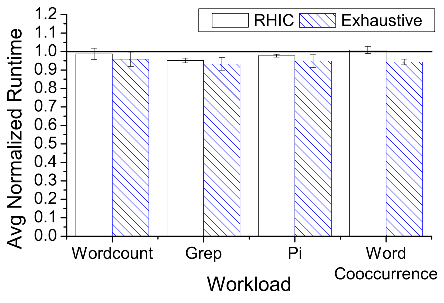
To generate the exhaustive performance survey, we developed a "targeted" version of our framework which maintains a specific number of volunteers using the same volunteer node selection and management mechanism (§4.4) as RHIC. Further, the random number generator which drives interactive node churn is seeded identically. This ensures that if RHIC and the targeted framework choose X interactive nodes at the same point in the background job, they will receive the same volunteer set.

Figures 9 and 10 show the performance of RHIC relative to the exhaustive search for cost and energy minimization, respectively. It can be clearly seen that (1) supplementing dedicated nodes with volunteers does bring monetary cost and energy benefits, (2) different volunteer cluster sizes result in a large range in execution cost, generating an up to 72% saving in monetary cost and 47% in energy comparing the most and least optimal settings, (3) the behavior of the cost/energy curves are highly workload-dependent, and (4) RHIC is able to identify the optimal or near-optimal cluster sizing solution automatically. The only notable anomaly is that RHIC undershoots the energy minimum for Co-occurrence by approximately 4 volunteers. This is because Co-occurrence is ultimately CPU-bound but has non-trivial I/O demand, which RHIC cautiously explores. Unlike Pi's energy minimization, for which RHIC immediately pushes to 36 volunteers, RHIC takes 3 steps up to 36 volunteers with Co-occurrence to ensure that I/O bottleneck does not occur. This is exacerbated by the long straggler phase in Co-occurrence, due to long task lengths, during which most volunteers sit idle.

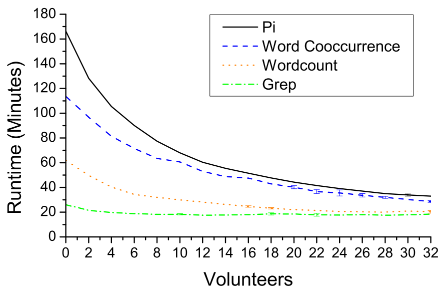
Figure 11 shows soft deadline enforcement results. Three deadlines were chosen for each background wrokload across the spectrum of achievable completion times, each tested twice for 6 total datapoints per background workload. In Figure 11a, the horizontal black bar marks the normalized deadline (@1.0). The exhaustive bar represents the closest setting which would achieve the deadline identified by the exhaustive tests. In Figure 11b, the range of possible completion times is shown for each workload to demonstrate the wide array of choices RHIC faces. Again, RHIC achieves near-ideal volunteer cluster sizes all workloads and deadline settings in most cases. More specifically, it misses 5 out of 24 deadlines, but by less than 3% on average.

5.3 Optimization Technique Evaluation

Conceptually, RHIC is based on the combination of online profiling and model-guided optimization. Given the highly-volatile nature of our harvesting environment and the



(a) Deadline enforcement performance.



(b) Workload completion time ranges.

Fig. 11: **Deadline enforcement performance.**

need for continual adjustment, a control theory approach could be a valid alternative. In this section, we compare RHIC's performance with that of an alternative scheme based on fuzzy control for minimization, as well as a naïve threshold algorithm.

Traditional control systems are well-suited for problems where the goal is clearly defined (i.e. deadlines) but the situation becomes complex when it is not (i.e. minimization). To address both cases, we turn to fuzzy control systems. Fuzzy control has been previously applied to minimization problems in server clusters [31], which is used as the basis for our fuzzy controller (FUZZY) design. Its 2-period historical comparison is similar to hill-climbing. For minimization goals, we use the cost or energy per unit of progress. For deadline goals, we minimize the difference between the current completion rate and the "ideal" completion rate, which will undershoot the deadline by a small padding value (default at 5%).

For Liu et al.'s Rules #1 & #3, we increment/decrement by a parameter $p > 1$ which is varied in our experiments. For Liu et al.'s Rules #2 & #4, we only increment/decrement by a single volunteer, since the intuition is that FUZZY is near the minimum. These rules are shown in Figure 12. To adapt the minimizing fuzzy controller design to our scenario, we use the interactive node selection and cluster management (§4.4) modules from RHIC for similar reasons as our "targeted" mode, discussed above. The fuzzy controller's logic is as follows:

1. Evaluate the efficiency of the previous evaluation interval
2. Compare the previous evaluation interval to the evaluation interval before it
3. Avoid action if the change in efficiency is below a threshold
4. Otherwise choose an action based on the fuzzy rules

FUZZY requires 2 initial "start points" because it is based on a historical comparison of two time-steps - the performance and control decisions of the previous two steps are used as the basis of next step. We use the same profiling phase as RHIC for the first step, to allow FUZZY to determine memory demands of the background workload at runtime. However, the second step must be manually determined, so we set it to a range of fixed values, as explained below.

In addition, we included a naïve "threshold" algorithm, blind to goals, which instead chooses interactive nodes with residual resource availability above a threshold, specified

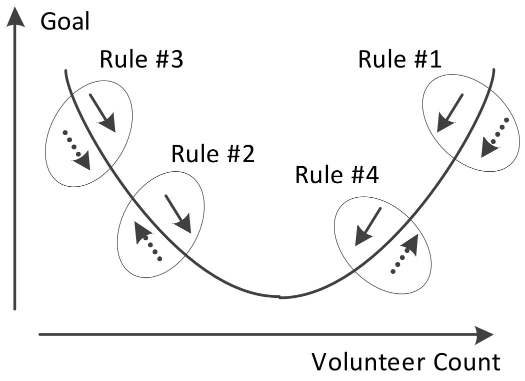


Fig. 12: **Fuzzy rules.** Original figure credit to Liu et al. [31]. Dotted lines represent the change which was observed, and solid lines represent the subsequent action which should be taken.

as a percentage of maximum CPU resources. For example, with a threshold of 0.50 and maximum CPU level of 400% (4 cores with 100% each), the threshold algorithm would select all interactive nodes with greater than 200% residual CPU availability.

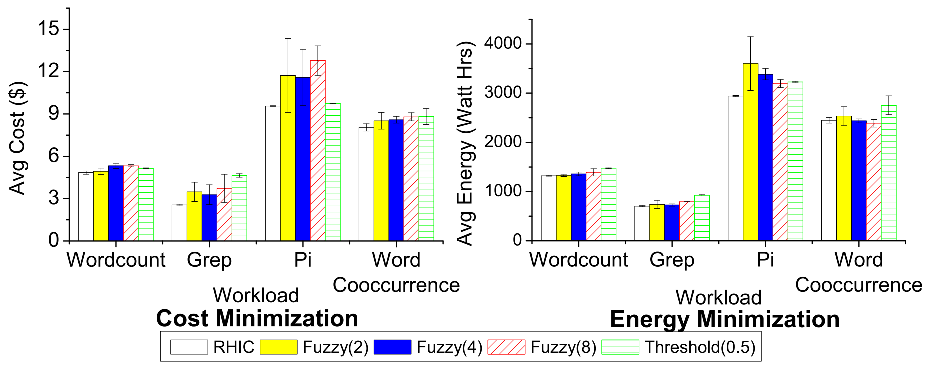
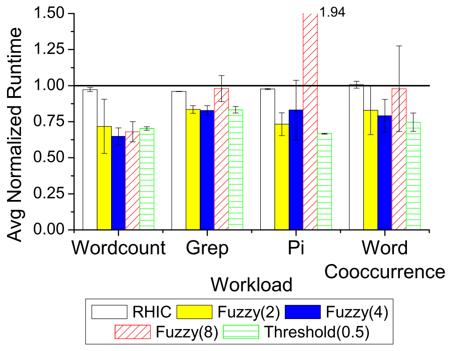


Fig. 13: **Cost and energy minimization performance.**

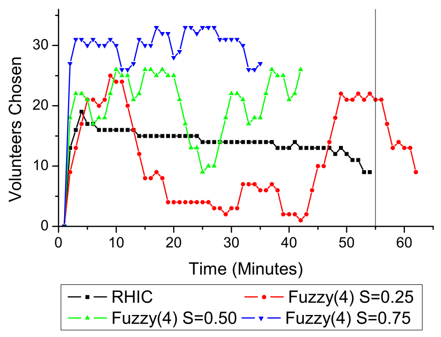
We evaluated the two alternative methods plus RHIC with all three goal criteria across our four background workloads, again using a hybrid cluster of 6 dedicated and 0-36 volunteers. The results are shown in Figure 13 (minimization) and Figure 14 (deadline enforcement).

For FUZZY, we varied p, the parameter that determines the magnitude of cluster size changes when the controller believes it is moving in the “correct” direction, among three values: 2, 4, and 8. For each background workload, goal and p value, we evaluated FUZZY with 3 different start points: $S = 25\%, 50\%, 75\%$ of the total volunteer pool (36 volunteers). This was motivated by the observation that FUZZY’s performance is heavily influenced by S, which can be seen in Figure 14b.

From Figure 13, we see that RHIC delivers better minimization results, and more significantly, much more stable performance. FUZZY, on the other hand, yields considerably higher variance even within an individual p setting, influenced heavily by S. Pi,



(a) Deadline performance. The horizontal black bar shows the normalized deadline (@1.0)



(b) Example choices for Pi. The vertical black bar shows the deadline (@55).

Fig. 14: **Deadline enforcement performance.** One deadline was chosen for each background workload, in the middle of its achievable completion time range.

among all workloads, sees the highest variance, as it offers the largest possible range of cost and energy values (as shown in Figure 9). The simple Threshold approach is more stable than FUZZY, but often yields inferior minimization results compared to the other two methods.

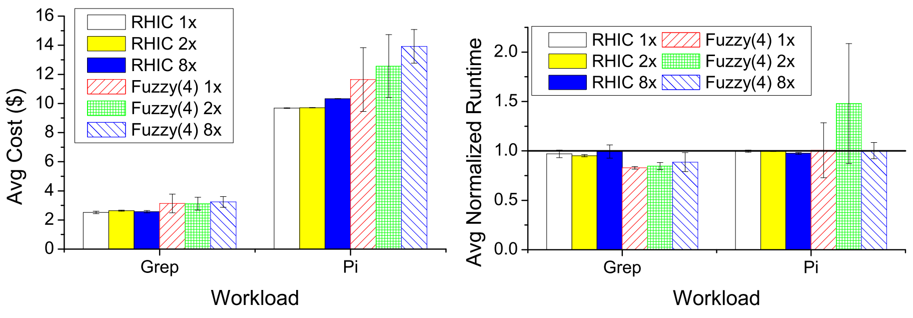
From Figure 14a, we see that RHIC delivers job completion time highly consistent and close to the deadline (within 4% difference), while FUZZY and Threshold behave too conservative or too aggressive (violating the deadline by almost doubling the desired completion time) in most cases, again with much higher variance. This is backed up by the example cluster sizing choices for Pi in Figure 14b: FUZZY produces much more dramatic cluster size scaling, while RHIC stays quite stable after the initial ramping up stage.

FUZZY's poor decision-making stems from two root causes. First, Hadoop's global progress indicator is not smoothly linear, due to task reporting and I/O delays. RHIC uses repeated sampling and averaging to address this issue. Second, FUZZY does not account for changes in the foreground CPU demand. One possible solution to this would be to use volunteer CPU consumption instead of volunteer count in FUZZY's fuzzy rules. However, our experience is that CPU consumption reporting itself is very noisy due to task turnover and I/O buffering, which RHIC addresses using problem-tailored curve fitting. Therefore, we opted against adding this capability to FUZZY, under the reasoning that it would simply shift the unreliability issue to another metric.

5.4 Impact of Environment Stability

In all preceding experiments, we used the VCL's natural churn rate, as discussed in Section 5.1. In this section, we attempt to quantify the impact of *increased churn* on RHIC's ability to conduct cluster sizing optimization.

In Figure 15 we show RHIC's cost minimization and deadline enforcement performance under $1\times$(baseline), $2\times$ and $8\times$ the normal churn rate, for monetary cost minimization and deadline enforcement. Here $2\times$ indicates that nodes join and leave twice as frequently, with half the mean and half the standard deviation of the baseline. For comparison, we also include the performance of FUZZY with $p = 4$, which was the most-accurate parameter found in § 5.3. Due to length limits, we show only two background workloads, the most I/O-intensive (Grep) and the most CPU-intensive (Pi).



(a) Cost Minimization

(b) Deadline compliance, with horizontal black bar shows the normalized.

Fig. 15: **Impact of increased churn on RHIC and Fuzzy(4) performance, at different interactive node churn rates.**

Overall RHIC clearly outperforms FUZZY in both minimization performance and consistency, largely due to the decision-deferment technique discussed in Section 4.4. More specifically, RHIC is resilient to the high interactive node turn-over rates and achieves near-identical cost minimization performance in all cases but Pi with 8× churn. This exception is because the volunteer pool is smaller on average at such high churn rate: we begin all experiments with 100% of nodes available, with many nodes absent by the end of the job. For the CPU-intensive Pi, it is impossible to reach the desired cluster size producing the minimum cost.

5.5 Generalization to Other Background Frameworks

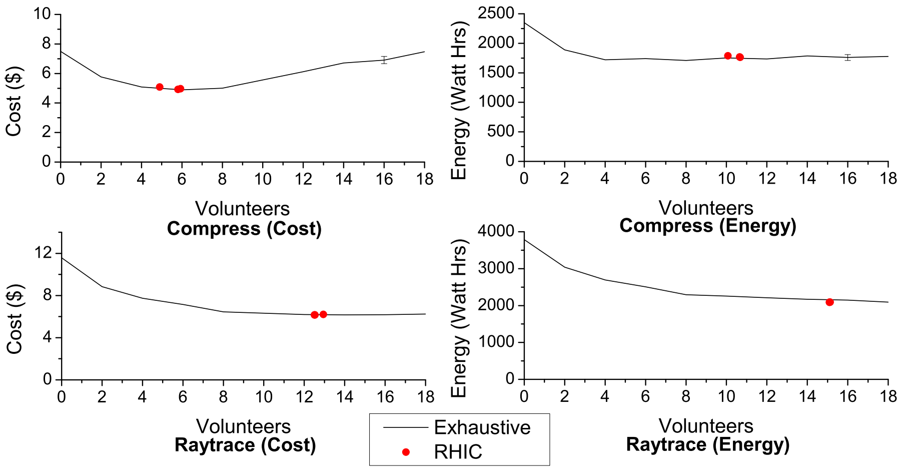


Fig. 16: **Cost minimization performance for HBasePCF.**

To demonstrate that RHIC is generalizable to non-MapReduce batch processing systems, we wrote a parallel compute framework (*HBasePCF*) in 800 lines of Python to perform batch jobs on top of HBase. In accordance with RHIC's requirements, HBasePCF only exports a progress score and average task length, and HBase is built on top of the HDFS distributed file system hosted on the dedicated nodes. We used HBasePCF to perform one I/O-intensive and one compute-intensive job, Compress and Raytrace respectively, as introduced in $\S$5.1.

Our hybrid cluster is composed of 3 dedicated nodes and 0-18 volunteers. For monetary cost, dedicated nodes are priced at $\$1.00/hr$ and volunteers at $\$0.42/hr$. For exhaustive search, we sampled within the valid volunteer cluster size range again with a step size of 2 ($v = \{0, 2, 4, \ldots, 18\}$), and repeated each test twice.

Figure 16 shows the cost and energy minimization performance of RHIC alongside the exhaustive search. Like in the case of Hadoop, RHIC achieves near-minimum performance in all cases, correctly adapting to the characteristics of each workload for each optimization goal.

5.6 Overhead

RHIC's overhead can be measured in two dimensions: the amount of resources RHIC itself takes to run, and its latency in making a cluster-sizing decision. By instrumenting RHIC's control VM, which resides on the Hadoop master, we found that it consumes less than 30% CPU (on one core) on average and takes less than $250ms$ to make an exhaustive cluster sizing decision for 36 volunteers. This overhead would be lower for a more efficient search algorithm ($\S$4.1). Since cluster sizing decisions are made once per evaluation interval (set to one minute in our experiments), all the periodic resource and job progress monitoring supporting RHIC's decision making brings less than 0.5% overhead on volunteer or candidate interactive nodes.

6 Conclusion and Future Work

In conclusion, we have outlined the design of RHIC, an autonomic management framework for harvesting resources with throughput-oriented parallel batch workloads. By combining black-box modeling and online profiling, RHIC is able to quickly discover and maintain optimal cluster sizes for a range of goals, including deadline satisfiction as well as cost and energy minimization. Through RHIC, we have found that it is possible to tolerate the high degree of instability in cloud nodes hosting interactive services to run background jobs with no *a priori* knowledge on either the foreground or background workloads. Finally, RHIC requires only system-level metrics and a progress score, which yields broad applicability to an entire class of embarassingly-parallel analytics workloads.

Our work is only a first step towards a full-featured harvesting batch platform. We are interested in identifying an ideal hybrid cluster composition for a given workload and performance goal, scaling both the dedicated and volunteer sides, potentially with topology awareness. Further, we plan to extend our system to flexibly harvest more resource types, including memory, local disk and residual network bandwidth.

References

1. NCSU: NCSU Virtual Computing Lab. `http://vcl.ncsu.edu/`
2. Li, J., Deshpande, A., Srinivasan, J., Ma, X.: Energy and performance impact of aggressive volunteer computing with multi-core computers. In: MASCOTS '09

3. Lin, H., Ma, X., Archuleta, J., Feng, W.c., Gardner, M., Zhang, Z.: Moon: Mapreduce on opportunistic environments. In: HPDC ’10
4. Lee, G., Chun, B.G., Katz, R.H.: Heterogeneity-aware resource allocation and scheduling in the cloud. In: HotCloud ’11
5. Chohan, N., Castillo, C., Spreitzer, M., Steinder, M., Tantawi, A., Krintz, C.: See spot run: using spot instances for mapreduce workflows. In: HotCloud’10
6. Amazon: Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`
7. Ferguson, A.D., Bodik, P., Kandula, S., Boutin, E., Fonseca, R.: Jockey: Guaranteed job latency in data parallel clusters. In: EuroSys ’12
8. Wieder, A., Bhatotia, P., Post, A., Rodrigues, R.: Orchestrating the deployment of computations in the cloud with conductor. In: NSDI ’12
9. Polo, J., Castillo, C., Carrera, D., Becerra, Y., Whalley, I., Steinder, M., Torres, J., Ayguad, E.: Resource-aware adaptive scheduling for mapreduce clusters. In: Middleware ’11
10. Herodotou, H., Dong, F., Babu, S.: No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In: SOCC ’11
11. Wood, T., Shenoy, P., Venkataramani, A., Yousif, M.: Black-box and gray-box strategies for virtual machine migration. In: NSDI ’07
12. Urgaonkar, B., Shenoy, P., Roscoe, T.: Resource overbooking and application profiling in shared hosting platforms. In: OSDI ’02
13. Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A.D., Katz, R., Shenker, S., Stoica, I.: Mesos: a platform for fine-grained resource sharing in the data center. In: NSDI ’11
14. Agarwal, S., Kandula, S., Bruno, N., Wu, M.C., Stoica, I., Zhou, J.: Re-optimizing data-parallel computing. In: NSDI ’12
15. NCSU: ARC Cluster. `http://moss.csc.ncsu.edu/~mueller/cluster/arc/`
16. Lim, H.C., Babu, S., Chase, J.S.: Automated control for elastic storage. In: ICAC ’10
17. Liu, H.: Cutting mapreduce cost with spot market. In: HotCloud ’11
18. Hermenier, F., Lorca, X., Menaud, J.M., Muller, G., Lawall, J.: Entropy: a consolidation manager for clusters. In: VEE ’09
19. Wang, G., Butt, A., Pandey, P., Gupta, K.: A simulation approach to evaluating design decisions in mapreduce setups. In: MASCOTS ’09
20. Deshane, T., Shepherd, Z., Matthews, J.N., Ben-Yehuda, M., Shah, A., Rao, B.: Quantitative comparison of xen and kvm. In: Xen Summit ’08
21. Cheng, Y.: Mean shift, mode seeking, and clustering. IEEE Trans. Pattern Anal. Mach. Intell. (1995)
22. Zaharia, M., Konwinski, A., Joseph, A.D., Katz, R., Stoica, I.: Improving mapreduce performance in heterogeneous environments. In: OSDI ’08
23. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: ISCA ’07
24. Kansal, A., Zhao, F., Liu, J., Kothari, N., Bhattacharya, A.A.: Virtual machine power metering and provisioning. In: SOCC ’10
25. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In: SOCC ’11
26. Foundation, A.S.: Hadoop. `http://hadoop.apache.org/`
27. Foundation, A.S.: Hbase. `http://hbase.apache.org/`
28. Chen, Y., Alspaugh, S., Katz, R.H.: Design insights for mapreduce from diverse production workloads. Technical Report EECS-2012-17, University of California at Berkeley
29. Mojang: Minecraft. `http://www.minecraft.net/`
30. Clay, R.B.: Enabling mapreduce to harness idle cycles in interactive-user clouds. Master’s thesis, North Carolina State University
31. Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J.L., Parekh, S.: Online response time optimization of apache web server. In: IWQoS’03