Building and Scaling Virtual Clusters with Residual Resources from Interactive Clouds

R. Benjamin Clay*, Zhiming Shen*, Xiaosong Ma*†
*Dept. of Computer Science, North Carolina State University.
† Computer Science and Mathematics Division, Oak Ridge National Laboratory
{rbclay,zshen5}@ncsu.edu, ma@csc.ncsu.edu

ABSTRACT

The popularity of cloud-based interactive computing services (e.g., virtual desktops) brings new management challenges. Each interactive user leaves abundant but fluctuating residual resources while being intolerant to latency, precluding the use of aggressive VM consolidation. In this paper, we present the Resource Harvester for Interactive Clouds (RHIC), an autonomous management framework that harnesses dynamic residual resources aggressively without slowing the harvested interactive services. RHIC builds ad-hoc clusters for running throughput-oriented "background" workloads using a hybrid of residual and dedicated resources. These hybrid clusters offer significant gains over normal dedicated clusters: 20-40% cost and 20-29% energy in our testbed. For a given background job, RHIC intelligently discovers/maintains the ideal cluster size and composition, to meet user-specified goals such as cost/energy minimization or deadlines. RHIC employs black-box workload performance modeling, requiring only system-level metrics and incorporating techniques to improve modeling accuracy under bursty and heterogeneous residual resources. 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 28 workload/goal combinations, with 5% average error for cost minimization and 3% for energy, relative to exhaustive searches, and runtimes 2% under deadlines. Further, RHIC significantly outperforms alternative approaches, tolerates high instability in the harvested interactive cloud, works with heterogeneous hardware and imposes only 0.5% overhead.

1. INTRODUCTION

Interactive cloud offerings are expanding, providing virtual computing laboratories, remote desktop environments and online collaboration tools. For example, North Carolina State University's Virtual Computing Laboratory (VCL) [31] is a production cloud system hosting virtual desktops with a variety of applications for more than 13,000 students at NCSU and other nearby schools. These new platforms bring individual users easy access to popular applications/tools with low management overhead. Such systems also yield significant *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 in this environment, due to users' bursty resource consumption patterns combined with response time requirements. Conservative workload consolidation, on the other hand, will likely leave significant amounts of residual resources idle, as we show in §3.1. By aggressively harnessing such resources, cloud 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 [22].

Harvesting residual resources in this 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 [9, 21, 25] have proposed a hybrid batch cluster design where *volunteer* nodes supplement a core set of stable *dedicated* nodes, in some cases using EC2 SPOT instances [3]. As shown in Fig. 1, a set of transient interactive nodes are "padded" with volunteer VMs running a background batch job, which consume residual resources while automatically deferring to the interactive user via hypervisor prioritization. This co-location of interactive and batch workloads is advantageous due to orthoganal temporal characteristics (§4.2), and has been described previously [22, 24]. In preliminary experiments (§3.2), we demonstrate 20-29% energy and 20-40% cost gains over normal dedicated clusters with only 1% slowdown of interactive workloads.

Shared nothing clouds such as the VCL lack robust shared storage, like Amazon's Elastic Block Store [2], making migration more costly for both foregroud and background users. As a result of this high cost and users' bursty resource consumption, we employ an I/O asymmetric design for the background cluster, where only the dedicated nodes provide persistent storage. The *volunteer VMs* use their local storage for temporary data only, while the foreground VMs are hosted entirely from local storage, as shown in Fig. 3. This choice allows volunteers to be lightweight and agile by avoiding data-loss and expensive replication as volunteers join and leave: volunteers are only sent data which they will immediately process, and are not relied upon to host data in the long-term. This approach is necessary because, in contrast to prior work on passive volunteerism for MapReduce [25], interactive nodes are much shorter-lived and unlikely to return in the near future. Finally, the hybrid cluster design provides a performance baseline to mitigate stragglers, caused by bursty and unreliable volunteers, via speculative execution of delayed volunteer tasks by dedicated nodes.

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, which brings new challenges not studied in prior work. Unlike in 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, Fig. 2 shows the diverse range of monetary costs and energy consumption among different batch workloads. These results are dependent on the specific batch in-

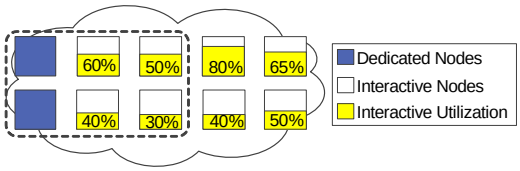


Figure 1: Sample hybrid cloud computing system, with 8 interactive nodes running interactive services. A background job runs on 2 dedicated and 4 volunteer nodes.

	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
Co-oc.	7.72	11.41	2230	3987

Figure 2: Cost and energy ranges for batch workloads on a hybrid cluster, with 6 dedicated nodes and 0-36 volunteers.

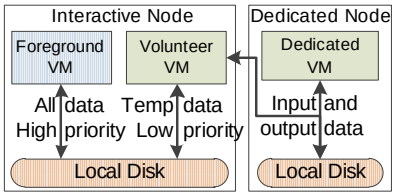


Figure 3: Disk layout in the hybrid cluster design. The hypervisor is used to prioritize foreground disk access.

puts, foreground workloads and pricing structure chosen, as well as 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 [15, 19, 32, 39], volunteerism/hybrid clusters for MapReduce [9, 21, 25] and workload consolidation [40]. However, these prior studies were not designed to consider the unique challenges in harvesting residual resources from interactive users, particularly (1) the high degree of temporal and spatial transience in residual resources, and (2) the dedicated node I/O saturation constraint in our target asymmetric architecture. More detailed related work discussion is given in §2.

To address such unique challenges in 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 noisy. White-box modeling allows higher sensitivity to the limitations of a particular platform, and potentially greater accuracy, but limits generalization. Real-world parallel batch workloads are commonly composed of short jobs [20] and novel jobs [1, 15]. As a result, profiling must be completed quickly with no *a priori* knowledge to yield reliable estimates early.

In this paper, we present the Resource Harvester for Interactive Clouds (RHIC), a generic management framework which autonomically optimizes a hybrid cluster running within residual resources. RHIC provides 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 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. The major contributions of this work are:

- To the best of our knowledge, we are the first to propose batch cluster sizing as a tool for resource harvesting in interactive clouds, with the goal of making the background job itself energy and cost efficient.
- 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. RHIC only relies on system monitoring data and a progress score from the background job, which allows generalization to a wide range of throughput-oriented workloads.
- We carried out an evaluation of over 400 runs on a hybrid cluster of 42 nodes, using real traces collected from production interactive clouds and representative batch analytics workloads. Our results show that RHIC achieves high accuracy across 28 workload/goal combinations in minimizing cost/energy (5%/3% error as compared to exhaustive surveys), and enforcing deadlines (2% under on average). In addition, we demonstrate RHIC's performance against alternative algorithms, tolerance for increased instability and hardware heterogeneity, and low overhead.

In the rest of the paper, we give an overview of related work in §2, provide background motivation in §3, and discuss RHIC's design in §4. In §5 we present our evaluation and in §6 we conclude.

2. RELATED WORK

Our work is related to contributions from several other areas:

Volunteer computing. Volunteer computing (VC), known widely through projects such as Condor [26, 35] and BOINC [4], has a long history as both a computation paradigm and a method of harvesting wasted cycles. While passive VC has traditionally formed the bulk of interest in this research area, advancing multitasking technology has made it feasible and attractive to perform active volunteer computing [18, 22], where the user and harvester coexist temporally. Active and passive VC are similar in spirit, with active VC posing additional challenges in maintaining interactive user experience [18, 23] and delivering consistent background performance using unreliable residual resources [6].

The focus of this work is related to the second challenge mentioned above: how bursty residual resources can efficiently provide a stable batch execution platform that meets performance and/or cost goals. RHIC's novelty is in modeling the relationship between batch workload progress and resource availability, with techniques to mitigate burstiness, heterogeneity and other artifacts of our hostile environment. While both passive and active VC are important prerequisites to RHIC, our design and claims are orthogonal.

Cluster sizing for parallel batch workloads. Several works have been recently published which perform cluster sizing for parallel batch workloads [15, 19, 32, 39, 41]. Of these, our efforts are most-closely related to those which combine modeling with online adjustment and feedback [15, 32, 39]. Jockey [15] 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 [39] also combines modeling and online adjustment to meet deadlines and minimize cost for MapReduce, taking into account data upload and migration overheads. RAS [32] 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 [19] is a system for optimizing cluster size for arbitrary MapReduce workloads and hardware, using a combination of workload profiling and and configuration parameter modeling. Yu et al. [41] describe a system for modeling batch workload perfor-

mance and allocating masters and workers to avoid resource waste.

Compared to the aforementioned efforts, RHIC addresses a unique permutation of traditional cluster sizing for parallel batch workloads. 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 [15, 19, 32, 39, 41] will not adequately capture the performance of a given cluster. Our insight regarding aggregate residual CPU availability and its direct effect on cluster performance (§4.4) 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 (§4.3) in response. Wieder et al. [39] 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. Yu et al. [41] consider data transfer time and cluster balance, but not I/O saturation at master nodes or imbalanced demand from heterogeneous workers.

RHIC can optimize novel and short-lived jobs (which are common [1, 15, 20]) with no *a priori* knowledge, using a combination of online profiling and adaptive scaling. All prior efforts require either previous executions of the target job [15, 19, 32, 41] or key performance characteristics [39]. While those with online adjustment [15, 32, 39] could adapt to some deviation from the profile performance (as Wieder et al. [39] demonstrate), the dynamic nature of volunteer heterogeneity directly inspired RHIC's online learning and reactive approaches to CPU (§4.3) and I/O (§4.4).

Finally, RHIC offers a highly-generic performance modeling interface, which only requires a job progress score and average task length. The models employed by prior works have various levels of dependency on the workload, from MapReduce as a concept [32, 39] to specific MR frameworks [15, 19]. Because we envision RHIC as a harvesting platform which manages throughput-oriented parallel batch jobs, 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 [15] by managing each stage independently.

Hybrid MapReduce, Volunteerism and Cluster Sharing. Prior works use Amazon EC2 Spot Instances to perform MapReduce jobs [9, 21, 27], 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 [9, 21], and (2) using Amazon's cloud storage service to preserve intermediate results [27]. 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 [9, 21] 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. [21] highlight a similar problem space to our work, they have not proposed any concrete solution for automatically determining ideal cluster size.

MOON [25] enhanced Hadoop to operate under passive volunteerism, where a foreground workload and MapReduce are interleaved temporally but not spatially. Mesos [20] is a framework for batch framework co-location above a shared distributed filesystem. Both works do not consider our target scenario, with two workloads asymmetrically sharing resources, or perform cluster sizing.

Table 1: CPU consumption, burst and reservation characteristics collected from NCSU's VCL. CPU data are collected from real user session traces (described in more detail in §5.1). Reservation data covers 750,000 sessions from 2004-2010.

Metric	Matlab	Photoshop	Office	C Dev
CPU Consumed (μ)	19.8%	7.0%	2.8%	22.5%
CPU Consumed (σ)	23.2%	16.2%	12.4%	24.3%
CPU burst height (μ)	39.9%	25.8%	31.0%	27.7%
CPU burst length (μ)	6.9 sec	2.0 sec	1.3 sec	47.4 sec
Reservation (μ)	93 min	74 min	70 min	120 min
Reservation (σ)	90 min	79 min	91 min	99 min

Workload Consolidation. Co-locating workloads on the same physical host is a well-established technique [40] complementary to our approach. RHIC can transparently harvest whatever residual resources are available after consolidation, with the expectation that the user will leave some free during periods of "think time".

3. BACKGROUND

As mentioned earlier, we leverage a *hybrid* cluster design [9, 21, 25] to harvest residual resources. In §3.1, we justify our cluster design choice by showing that it is appropriate for our environment. Then, in §3.2 we validate assumptions regarding the feasibility and profitability of adopting this approach.

3.1 Hybrid Cluster Design Rationale

A hybrid cluster is composed of dedicated nodes accelerated by lightweight volunteer VMs, providing a large performance boost limited only by the size of the dedicated cluster and the scalability of the workload. Volunteer VMs are hosted alongside foreground VMs in a pairwise fashion, arbitrated by the hypervisor, with volunteers granted minimum priority and foreground VMs granted the maximum. Volunteers are pre-loaded on foreground nodes, booting at the same time as foreground VMs and waiting latent until needed. When in use, volunteers are sent only the data which they need for immediate computation, returning outputs to the dedicated nodes as they are completed.

Our hybrid design is motivated by the characteristics of interactive cloud workloads observed on the VCL. Table 1 summarizes statistics information collected from VCL remote desktop sessions. It shows that while user reservations are fairly long, their durations have very high variances, indicating unpredictable session lengths. Further, CPU bursts are quite short-lived, even for the more computation-intensive workloads (such as Matlab). The low average utilization is indicative of the significant residual resource availability in this cloud, which will go to waste and yield poor energy efficiency unless they are captured. **The sheer volume of wasted resources in a cloud of this size (over 1000 nodes) justifies examining hybrid cluster design**, as a cost-effective alternative to buying and operating separate batch clusters.

Such highly dynamic behavior renders traditional approaches such as workload consolidation [36] less appealing. Conservative consolidation approaches can maintain interactive users' QoS requirements but will inevitably waste resources. Aggressive approaches, on the other hand, may face severe performance penalties in case of resource conflicts. In particular, **shared nothing** clouds have significant migration costs because both the disk image and memory contents must be transferred, often requiring minutes even with high bisection bandwidth. Although live migration is possible

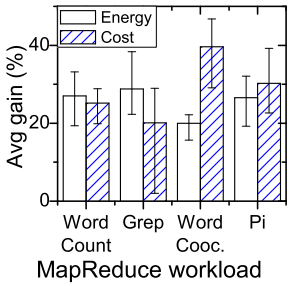


Figure 4: Energy and cost savings by using a hybrid cluster design, over a regular dedicated-only cluster. Error bars represent the range of savings.

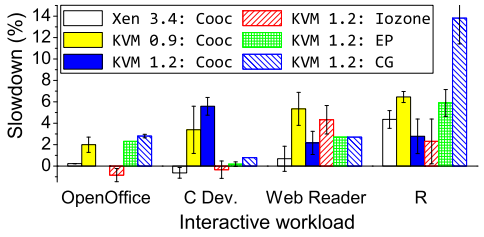


Figure 5: Slowdown of interactive foreground workloads padded with volunteers. Foreground workloads include members of `bltk` and AT&T's R benchmark. Background workloads include several resource-intensive benchmarks: Word Cooccurrence (Cooc), Iozone, and NAS PB (EP, CG), which run on all cores using node-local MPI.

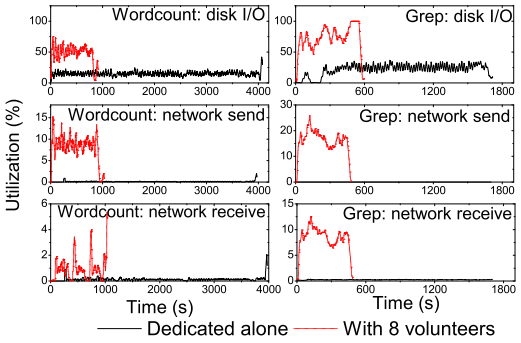


Figure 6: Disk and network bandwidth utilization on 2 dedicated nodes, with and without 8 volunteers. Disk utilization is measured by the % of time the CPU spent blocked on I/O.

both with shared and non-shared storage [30], the short CPU bursts and highly variable session durations seen in interactive workloads will require frequent migration and may lead to heavy thrashing.

In the hybrid cluster design, the dedicated nodes have node-local storage capacity, while the volunteer VMs only use their local storage for temporary data, as shown in Fig. 3. 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, making it much easier to use/discard a node due to foreground interactive load shifts and to dynamically scale the virtual cluster size. 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 residual resources while preventing stragglers from delaying job completion. Finally, this approach helps *insulate foreground VMs from heavy I/O contention on local disks* by offloading most background disk traffic to the dedicated nodes.

3.2 Validating Key Assumptions

Here we validate three key assumptions used in our design:

1. Savings over dedicated clusters. To verify the energy/cost benefits of the proposed hybrid cluster approach, we experimented with 2 dedicated nodes and 2-8 volunteers, priced/metered as discussed in §5.1. Fig. 4 shows sample monetary and energy savings when running Hadoop workloads on a hybrid cluster, as compared to using a regular Hadoop cluster with the same number of nodes (dedicated + volunteers). E.g., we directly compare 2 dedicated + 2 volunteers to 4 regular nodes. The foreground workload on volunteer nodes is Photoshop. The hybrid cluster design is shown to deliver significant savings: 20-29% energy and 20-40% cost.

2. Foreground users can be isolated from volunteers. Modern hypervisors have been shown to offer effective performance isolation [12], partially demonstrated by today's high VDI densities [11, 37]. We further verified this with our own experiments by testing work-conserving schedulers in the Xen and KVM hypervisors. These tests co-located foreground and background VMs, with the foreground given the maximum CPU, disk and network priority, and the background VM minimum. We tested our most resource-intensive background workload (Word Cooccurrence) on three hypervisors, as well as I/O and CPU benchmarks on the latest version of KVM. Our results (Fig. 5) indicate that the performance impact is low despite virtual desktop applications' sensitivity to I/O latency. Xen is quite effective in performance isolation, with an average slowdown of 1%. KVM 1.2 delivers $< 6\%$ slowdown

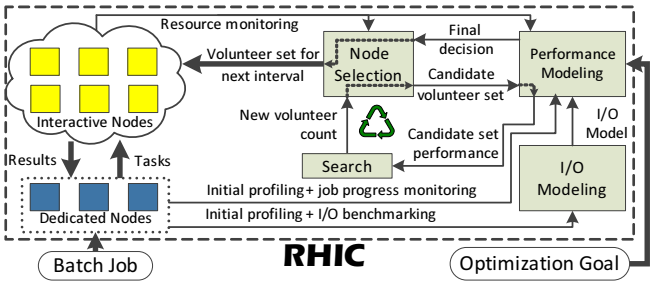


Figure 7: RHIC components and data flow

with Word Co-occurrence, and does quite well alongside intensive microbenchmarks with the exception of R paired with CG, due to CG's high memory bandwidth demand. To our knowledge, no hypervisor currently arbitrates memory bandwidth usage.

3. Dedicated nodes have sufficient residual disk bandwidth to offload computation to volunteers. To verify this, we examined 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 volunteers (with no foreground workloads, to create maximum I/O pressure). Figure 6 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 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 therefore more bottleneck-prone. This reinforces our choice to (1) *accelerate a dedicated cluster with volunteers* and (2) *identify the appropriate number of volunteers for a given dedicated cluster.* Our later evaluation (§5.2) further corroborates this conclusion.

4. FRAMEWORK DESIGN

4.1 Overview

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. Progress scores are commonly exported by batch frameworks as a function of their fixed input sizes, in contrast to streaming or transactional workloads. Fig. 7 shows RHIC's major components (and their interactions), which collaborate to periodically re-evaluate cluster sizing decisions. RHIC starts a batch job execution with a profil-

ing 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, and gather background job characteristics such as memory requirements.

Throughout the rest of the job execution, RHIC continues to monitor system status, such as interactive node resource usage, dedicated node I/O saturation level 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. The background job's execution is partitioned into *evaluation intervals*. At the beginning of each interval, a search algorithm generates candidate volunteer counts to be evaluated. For each volunteer set size, interactive nodes are selected to meet this quota by the *node selection* component (§4.2), based on online node resource monitoring data. Their predicted resource availability is supplied as input to the I/O model, generated by the *I/O modeling* component (§4.3), which identifies the I/O saturation point on the dedicated nodes and determines whether a given set of volunteers will incur dedicated-side disk bottlenecks. Finally, completion time and goal performance is predicted for the cluster by the *performance modeling* component (§4.4). The best candidate volunteer pool is used until the end of the interval, when the process repeats.

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 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.

Throughout this section, we make reference to a synthetic metric which we call *productivity*, which represents a volunteers' ability to perform work on behalf of the background workload. Productivity is measured in units of CPU utilization (%), but through the modeling process is adjusted to account for foreground CPU demand and memory restrictions, as well as I/O bandwidth restrictions. We explain how this metric is formulated in §4.2-4.3, and how RHIC uses it to model workload performance in §4.4.

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 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:*** Although each foreground interactive workload has unpredictable resource usage bursts, its *average* usage over a longer period of time tends to be more stable.
- ***Insight 2:*** In our proposed hybrid execution mode, the disk I/O bandwidth afforded by the dedicated nodes can be a major factor limiting the *effective* productivity of a volunteer.
- ***Insight 3:*** The overall progress of a batch job is determined by the *aggregate* productivity from all selected volunteers, largely independent of the productivity distribution among these nodes.

In the rest of this section, we discuss in detail the above insights and the interaction between several major RHIC components. Note that for simplicity, our initial discussion is based on homogeneous hardware across the node pool. However, in §4.6, we address this shortcoming by explaining a thin translation layer that allows RHIC manage and model different node types.

4.2 Volunteer Selection and Management

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 residual resource monitoring and prediction, as discussed below. Common interactive cloud workloads are highly bursty, making load consolidation [40] 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 can be effectively estimated using near-term history data (Insight 1).

Residual resource prediction: RHIC employs an online foreground workload CPU demand model using once-a-second CPU consumption samples from the interactive nodes. We considered four common prediction methods: moving average, auto-regression, auto-correlation, plus a hybrid of signature-based Fast Fourier Transform and Markov chains used in previous work [34].

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 a short lookahead but simultaneously do not expect a long history to be available due to interactive node transience. Fig. 8 shows the accuracy of these four prediction methods. 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.

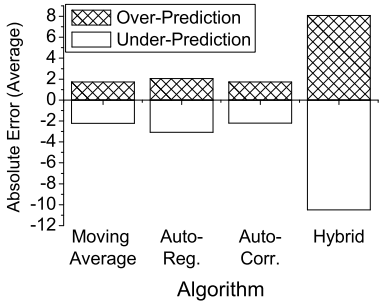


Figure 8: Accuracy of four different prediction algorithms for the foreground traces which we use. Absolute error is shown, with a value range of 0-100.

For memory, we assume that the foreground VMs have prespecified memory caps based on their workload, as in the case of Amazon EC2 and VCL instances. Background memory requirements, on the other hand, are estimated during the initial profiling phase. For MapReduce-like platforms, we adjust the number of simultaneous worker processes (such as Map slots N_{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.

Put together, the predicted foreground CPU consumption (CPU_{fg}) and CPU after considering memory restrictions ($100\% \times N_{Slots}$) indicate the volume of unused residual resources available for volunteer consumption on an interactive node. In effect, whichever of these two factors is most-restrictive dictates what CPU will be available for the volunteer's workload. We call this quantity *potential productivity* $P_{potential}$ (Eq. 1), because it is the estimated maximum productivity a volunteer, harvesting on this interactive node, could contribute to the background job. We distinguish this quantity as *potential* because I/O bottlenecks may result in a lower *actual* productivity, as we discuss in §4.3. Here

CPU_{max} represents the maximum CPU available on the interactive node, such as 400% for four cores.

$$P_{potential} = min(CPU_{max} - CPU_{fg}, (100\% \times N_{Slots})) \quad (1)$$

Note that we do not consider time-of-day in our predictions as idle cloud sessions are likely to be terminated by either the user or the system for cost/energy saving, as does the VCL. There will likely be daily or weekly interactive pool size fluctuations, which can be handled by RHIC as a global constraint when selecting volunteer cluster sizes for multiple concurrently running background workloads.

Node selection: In selecting specific volunteers from the interactive node pool, we adopt a greedy algorithm for better scalability. Candidate nodes are sorted according to their potential productivity level. Then 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 discussed in §4.4. 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. Further, this limits the search 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 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 swap the two. This process is repeated until the CPU availability difference falls under the threshold. Interactive node churn presents an issue for our search-driven cluster sizing scheme, because 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 immediately 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 session, there could be significant thrashing caused by the search process as it tries to react to a series of arrival/departure events. To avoid this, RHIC takes a *deferment* strategy: upon an interactive pool change, it enforces the decision made at the end of the last evaluation interval, deferring new decisions to the end of the current interval.

In our shared-nothing cluster, we disable migration because it is costly and ill-suited (§3.1). However, if foreground migration is enabled, RHIC can seamlessly adapt to the post-migration volunteer with its constant monitoring, periodic volunteer pool assessment and node selection.

4.3 Modeling Workload I/O Behavior

As verified in §3.2, our proposed method is based on the observation that, for typical distributed batch workloads, there is available I/O/network bandwidth for dedicated nodes to support additional volatile, diskless volunteer nodes. This model applies to background workloads with non-trivial compute demand, but this category is fairly broad - we find that significant cost/energy gains can be achieved for Grep, which is substantially I/O-intensive. However, as the number of volunteers grows, eventually I/O bandwidth on dedicated nodes is likely to become the chief limiting factor for performance/scalability (Insight 2), which has not been considered in prior work [9, 21]. Despite this scalability limitation, we show in our evaluation (§5.2) that a pool of volunteers can greatly *accelerate* a dedicated cluster's performance, making this portion of RHIC's modeling especially valuable.

RHIC builds an I/O model at runtime for the target batch job to identify the existence of I/O bottlenecks. Fig. 9 illustrates the interaction between the volunteer productivity and the I/O contention at the dedicated nodes for two sample MapReduce workloads. It shows the aggregate *actual productivity* from the volunteers at each level of aggregate *potential productivity*, averaged over the Map phase. The actual productivity is measured from the volunteer VM usage, while the potential is calculated with Eq. 1. 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 aggregate actual productivity. Fig. 9b plots the actual to potential productivity ratio over different volunteer counts. It illustrates that the MapReduce job consumes a constantly declining portion of the aggregate potential productivity. As a result, we base our I/O model on $\{P_{potential}, P_{actual}\}$ pairs for the given workload and hardware, derived at runtime.

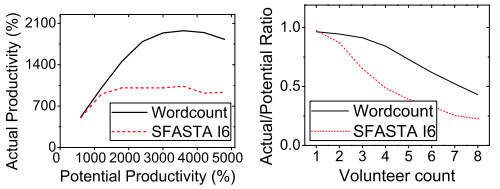
Saturation Point Estimation: For each background job, RHIC builds an I/O curve that tracks potential productivity on the X-axis and actual productivity on the Y-axis, in order to ultimately predict the actual productivity for a given volunteer set. RHIC uses data from the initial profiling, as well as continuous sampling, and applies regression to build this I/O curve. To avoid 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. 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 saturation point estimate on I/O bandwidth consumption data collected in the initial profiling phase. Assuming a linear relationship between actual productivity and I/O demands (limitations discussed below in §4.5), it estimates the excess volunteer productivity each dedicated node can support using Eq. 2. Here $BWUtil_{avg}$ is the average disk bandwidth utilization measured on the dedicated nodes during the initial profiling phase, and P_{max} is the maximum productivity potential on a node. $Vol\ P_{supp}$ is the volunteer productivity *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 $Vol\ P_{supp}$, where N_d is the number of dedicated nodes:

$$Vol\ P_{supp} = (\frac{100\%}{BWUtil_{Avg}} - 1) \times P_{max} \quad (2)$$

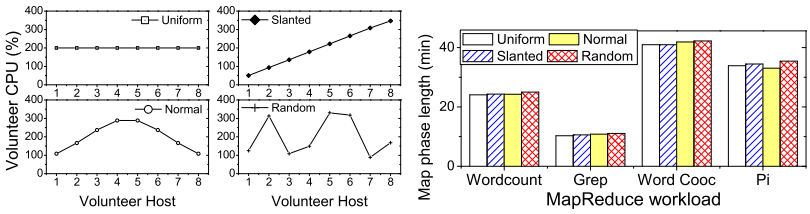
$$S_{best} = Vol\ P_{supp} \times N_d \quad (3) \qquad S_{worst} = Vol\ P_{supp} \quad (4)$$

Using the best and worst case estimates, RHIC 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.



(a) Actual **(b) Relative**

Figure 9: Impact of I/O bottlenecks on actual volunteer productivity, using 2 dedicated and 1-8 volunteers, with a max of 800% CPU on each.



(a) Resource distributions **(b) Completion times**

Figure 10: Impact of residual resource distribution on MapReduce job completion time for 2 dedicated and 8 volunteer nodes. All distributions have the same total residual CPU.

Algorithm 1 Initial volunteer pool scaling algorithm. The cluster is pushed to the saturation point using a combination of Eq. 3-4 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 if the price of volunteers is relatively high.

```
1: // Perform one profiling period with dedicated nodes alone
2: ProfileData ← RunPeriod(∅) , Period ← 1, Vols ← ∅
3: // Calculate saturation range using Eq. 3-4
4: (S_worst, S_best) ← ComputeSaturationPoints(ProfileData)
5: // Predict the max from DMR, ignoring I/O saturation
6: Max_DMR ← PredictIdealIgnoringIO(ProfileData)
7: // Push to the I/O saturation point, accounting for DMR
8: while (IOSaturation() ≠ True) AND (|Vols| < Max_DMR) do
9:   if Period == 1 then
10:    NextSize ← min(S_worst, Max_DMR)
11:  else if Period == 2 then
12:    NextSize ← min((S_best+S_worst)/2, Max_DMR)
13:  else if Period == 3 then
14:    if Max_DMR < S_best then
15:      NextSize ← Max_DMR
16:    else
17:      // Linear extrapolation
18:      Vols ← ExtrapolateSaturation()
19:    end if
20:  else
21:    NextSize ← |Vols| + S_worst
22:  end if
23:  // Find volunteers to satisfy the next size, and run the period
24:  Vols ← FindVolunteers(NextSize)
25:  RunPeriod(Vols) , Period += 1
26: end while
```

Improving I/O Balance: To increase the chance that I/O load is balanced across dedicated nodes, therefore yielding a saturation point closer to S_{best}, RHIC can leverage background framework-specific cues to assign volunteers to dedicated nodes in a round-robin fashion. In Hadoop, topology locality cues are used to assign sets of volunteers to the same logical rack as dedicated nodes, increasing the probability (but not guaranteeing) that mid-job, I/O demand is balanced across dedicated nodes. Since Hadoop allows for arbitrary rack hierarchy depths, this technique can be used to incorporate real topology data as well, by assigning dedicated and volunteer nodes to the same physical rack, and then subdividing them into logical racks per dedicated node. Hadoop will prefer DataNodes which are "closer" to the given TaskTracker, resulting in a volunteer trying its assigned dedicated node, then other rack-local dedicated nodes, before proceeding to other more-remote dedicated nodes. Below in §4.6 we discuss how heterogeneous I/O subsystems on dedicated nodes can be factored into this scheme.

I/O Curve Building with Clustering and Curve-fitting: Next, we complete the I/O curve that maps aggregate potential volunteer productivity to aggregate actual volunteer productivity. RHIC uses Mean-Shift clustering [8] to pre-process raw $\{P_{potential}, P_{actual}\}$ data points. This allows us to avoid a critical flaw in using curve-fitting for decision making, where incorrect decisions reinforce themselves by repeated sampling in the same area, 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, which is important near the saturation point, because minimization decisions hinge on marginal cost/gains. Prediction of the aggregate actual productivity on a given set of volunteers can then be performed with interpolation, based on the projected aggregate potential productivity on these volunteers. This approach assumes that the network bandwidth is either static or is not a limiting factor, which we believe is reasonable (RDP sessions consume a low 384Kbps on average [10]) but will be relaxed in future work, to incorporate hotspot detection, topology awareness and bandwidth availability prediction.

4.4 Background Job Performance Modeling

Background job performance modeling is the core of RHIC's sizing intelligence. As mentioned earlier, RHIC's performance modeling is based on the observation that the aggregate productivity from the selected volunteers, largely independent of the distribution of residual resources on individual volunteer nodes, is the chief factor determining a job's completion time on a hybrid cluster (Insight 3 - limitations discussed below in §4.5). Fig. 10 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, simulating different productivity distributions. According to each distribution, a volunteer is allocated 4 cores with a CPU cap between 50%-350% (with one core = 100%), while the total CPU allocations from all 8 volunteers are fixed at 1600%. Fig. 10a illustrates the shape of the distributions used.

Fig. 10b 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 speculative execution with the well-proven LATE algorithm [42]. In particular, the fact that dedicated nodes are robust, stable and 100% available results in aggressive, reliable speculative execution, effectively taking over the job from straggler volunteers.

This 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 potential productivity from candidate volunteer node sets, filtered through the I/O model. Although Fig. 10 only demonstrates static CPU allocation heterogeneity, we show in our evaluation that this technique can be successfully applied to dynamic heterogeneity as well.

Completion Time Estimation and Damping: More specifically, RHIC predicts that "a background job will complete at time y if it receives a sustained total volunteer productivity of x". This simplification is aided by both RHIC's preference for most-productive volunteers (§4.2) and speculative execution. For this, we developed a simple model based on the processing rate R_{proc}, shown in Eq. 5. Here $J_{completed}$ is the current fraction of the job completed, $T_{elapsed}$ is the time elapsed, and AP_{actual} is the aggregate actual productivity (dedicated + volunteer) over $T_{elapsed}$. R_{proc} is re-evaluated periodically during the background job.

By calculating the fraction of remaining work $J_{remaining} = J_{total} - J_{completed}$, we can then invert Eq. 5 and produce a Map completion time estimate $T_{remaining}$, given a predicted aggregate actual productivity $AP_{predicted}$, as shown in Eq. 6. $AP_{predicted}$ is calculated by applying RHIC's I/O model to the volunteers' predicted aggregate potential productivity, which together estimates the aggregate productivity that 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.

$$R_{proc} = \frac{J_{completed}}{T_{elapsed} \times AP_{actual}} \quad (5)$$

$$T_{remaining} = \frac{J_{remaining}}{AP_{predicted} \times R_{proc}} \quad (6)$$

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 (discussed in §4.8). 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.

Overall, with our experimentation we found this runtime modeling approach simple but effective. We view such simplicity as an asset, in contrast to alternative approaches which rely on highly-detailed whitebox techniques [19] and therefore cannot be applied to a broad range of parallel batch frameworks.

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 are likely to be charged at a lower rate. Given a certain pricing policy, RHIC calculates the performance score as the 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 [13]. 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 [13]. We then use multiple regression to derive a power model that estimates power consumption at an arbitrary utilization and frequency level. This 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. We made the simplification of using CPU alone, as energy modeling is not a major focus of our work and this naïve approach proved accurate for our testbed. If needed, it could be replaced with a more sophisticated model without modifications to other parts of RHIC.

Recall that our hybrid cluster design is partially motivated by the energy savings enabled 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.

4.5 Limitations of Linearity Assumptions

The aforementioned performance modeling is dependent on R_{proc} remaining somewhat static over the lifetime of the job. While our scheme tolerates noise in R_{proc} calculation resulting from uneven job progress reporting (shown in §5.3), workloads that have inherently heterogeneous progress can reduce RHIC's accuracy. For example, biological sequence search algorithms can skip over large portions of input sequences depending on their similarity. In cases like these, RHIC will only track the average R_{proc} of the workload, which makes the accuracy of runtime predictions dependent on the variance of this metric. The same issue applies to our assumption of linearity between I/O demand and CPU consumption: heterogeneous job progress per unit CPU is typically indicative of heterogeneous I/O demand per unit CPU. This assumption has also been made in prior works [41], but does hinder RHIC's ability to handle certain workload classes.

We believe that this shortcoming can be addressed with offline profiling, or hints provided by the administrator, which would allow us to distill how much R_{proc} varies in the given workload. Since R_{proc} is input-dependent, we would need to observe a number of runs across different inputs in order to form a degree of confidence in our measurements. These statistics will allow us to formulate best and worst-case estimates of the R_{proc} of a novel input at the job's outset, which can in turn influence how aggressive or conservative RHIC behaves, based on the performance goal. For example, to meet deadlines, RHIC would hew closer to the worst-case estimate early in the job, while for minimization goals, a midpoint between the best and worst-case may be more appropriate.We leave examination of this sub-problem as future work.

4.6 Handling Hardware Heterogeneity

Our I/O and runtime modeling techniques center on our synthetic productivity metric $P_{potential}$, described in previous sections. Heterogeneous hardware presents a problem with this metric because a background workload is likely to generate different progress rates at the same CPU utilization across different hardware, and exert different I/O pressure as a result. To cope with this disparity, we use a two-fold scheme which provides a translation layer to calculate equivalencies between different machine classes. This general approach has been applied in prior works [14, 19] under the assumption of a rather limited set of machine classes or generations, valid for today's clouds (like EC2 and VCL).

For different CPU and/or memory bus speeds, we use a translation metric which we call efficiency E, which allows us to equate

the processing power of different machine classes to a base class 0 ($E_0 = 1$), which we merely set as the most-popular class at cluster launch. E_K for a new machine class K is calculated by comparing the job progress $J_{completed_K}$ per unit of actual productivity P_{actual_K} on the new class, to that of the base class, as shown in Eq. 7. This technique is then applied to calculate the potential productivity in common units, as shown in Eq. 8, which can be directly incorporated with productivity from other classes when calculating P_{actual}, R_{proc}, runtime etc.

$$E_K = \frac{J_{completed_K}}{P_{actual_K}} \times \frac{P_{actual_0}}{J_{completed_0}} \quad (7)$$

$$P_{potential_{common}} = P_{potential_K} \times E_K \quad (8)$$

For different disk I/O subsystems on dedicated nodes, producing a translation metric based purely on online profiling is somewhat trickier because it must be disintangled from CPU differences. Because of this, we perform basic offline profiling of disk bandwidth for each dedicated machine class, deriving a disk bandwidth equivalency metric B_K for each machine class K, again relative to a chosen base class with $B_0 = 1$. B_K is then incorporated into our scheme in two ways: first during saturation point estimation and second in allocation of volunteers to dedicated nodes using topology cues, as discussed above in §4.3. For saturation point estimation, we multiply the best and worst case estimates by the average of B_K in the dedicated cluster. For volunteer allocation to dedicated nodes, we perform a similar scaling, assigning proportionally more volunteers to dedicated nodes with higher B values.

4.7 RHIC Scalability

In our evaluation, we run RHIC on a single management node with minimal overhead (30% single-core CPU utilization, 2% multi-core) and quick decision turnaround (250ms) using exhaustive searches on a hybrid cluster of 42 nodes. To handle much larger clusters of hundreds or thousands of nodes, we believe that significant portions of RHIC can be parallelized and pushed out onto the volunteers themselves if necessary. Each interactive node could use residual cycles to predict its own near-future CPU availability, and communicate this to RHIC. Further, each iteration of the search algorithm evaluates a different volunteer pool independently, and therefore can be parallelized and computed by volunteers. A more sophisticated search algorithm could further reduce decision overhead. Building the I/O model and calculating R_{proc} can also be accelerated in a distributed fashion, by aggregating volunteer productivity information before sending it to the management node using a tree-shaped reduction overlay network [5].

4.8 Integrating RHIC into MapReduce

RHIC uses a generic modeling approach and can manage a wide class of embarassingly-parallel batch frameworks. At present, MapReduce is easily the most-popular paradigm within this workload class, and below we discuss several issues specific to using RHIC to manage MapReduce background jobs.

Multi-tenancy: MapReduce clusters are traditionally multi-tenant with several jobs vying for available slots. Because RHIC tightly couples cluster size and the performance characteristics of a single job, we believe greater performance and efficiency can be gained by running multiple RHIC-guided hybrid clusters, side-by-side within the same cloud. With this proposed execution model, each job would be anchored on a set of dedicated nodes and managed by an independent instance of RHIC, harvesting from a shared pool of interactive nodes, each used by at most one job at a time. The RHIC instances would have no knowledge of each other and therefore could be co-located using a trivial arbitration layer, requiring no modification to RHIC. For example, interactive nodes could be offered to RHIC instances in a round-robin fashion, and rotated if unused after several periods, similar to Mesos [20].

Volunteer termination: The lifetime of a volunteer is equal to the lifetime of the interactive node which it resides on. As a result, volunteers can be terminated with little warning, which poses a problem for Hadoop because the JobTracker assumes that Map tasks completed by the terminated node are lost. Several fixes for this issue have been proposed, including task checkpoint-restart [33], pushing intermediate data to Reducers [28] and placing intermediate data on a distributed filesystem. Amazon's Elastic MapReduce (EMR) takes this third approach, allowing EMR clusters to use unstable SPOT instances. Because these features are not implemented in our version of Hadoop (0.21), we emulate them using a modified scheduler, which allows us to stop assigning new Map tasks to a volunteer while preserving its intermediate data.

Reducer placement: The loss of Reduce tasks is particularly damaging to MapReduce job runtimes because intermediate data must be re-shuffled [9, 38]. As a result, we do not run Reducers on volunteers and focus on the runtime of the Map phase, which for our workloads dominates the total execution time. This is backed by findings [7] that Map-only jobs are common, the Map phase dominates MapReduce jobs, and input data is the majority of stored bytes. We believe that this is a reasonable simplification, given that hybrid clusters have an abundance of CPU but limited I/O resources.

5. EXPERIMENTAL EVALUATION

In this section, we evaluate RHIC in six key areas, after giving an overview of our test platform in §5.1. First, in §5.2, we establish that RHIC can accurately discover near-ideal cluster sizing decisions, in comparison to an exhaustive search. In §5.3, we 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 §5.4, and demonstrate RHIC's general applicability to parallel batch frameworks in §5.5. Finally, we evaluate RHIC's hardware heterogeneity tolerance in §5.6, and briefly discuss RHIC's overhead in §5.7.

Unless otherwise noted, we run each test three times and report the average, with the goal of evaluating RHIC under a wide range of scenarios. To this end, we have conducted over 400 real-world (non-simulated) experiments, each with over 600 worker processes and in general found the variance to be quite small. Error bars denoting standard deviation are omitted unless we have at least 5 runs for a given test and the deviation is $\geq 2\%$.

5.1 Test Workloads, Platform, and Settings

Background Workloads: For evaluating RHIC, we use Hadoop [16] and a thin compute layer running over HBase [17] as the background job execution frameworks. Hadoop and HBase are widely used open-source implementations of the Google MapReduce and BigTable systems, respectively.

We used four representative MapReduce workloads: Wordcount (70GB of input), Grep (70GB of input), Word Co-occurrence (11GB of input), and Pi (trivial input). Map phase execution times are typically between 20 and 40 minutes, and are the target for our optimization as mentioned above (§4.8).

We used two representative workloads on top of HBase: Compress (offline LZO compression of text cells on 70GB of input data), which is I/O-intensive, and Raytrace (image generator on 100MB of input data), 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-

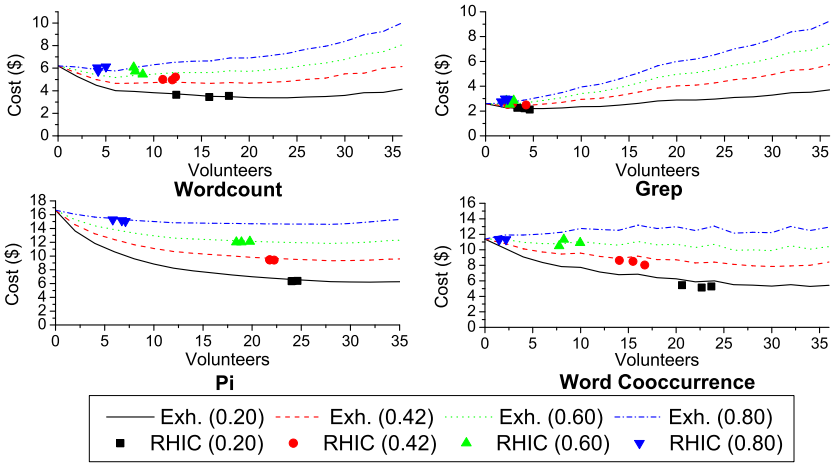


Figure 11: RHIC vs exhaustive: Cost minimization. Dedicated nodes are fixed at C_d = $\$1.00/hr$, with four different volunteer rates: C_v = $\{\$0.20, \$0.42, \$0.60, \$0.80\}/hr$, represented as Exh.($C_v$) and RHIC($C_v$). Volunteer count is the time-weighted average over the job.

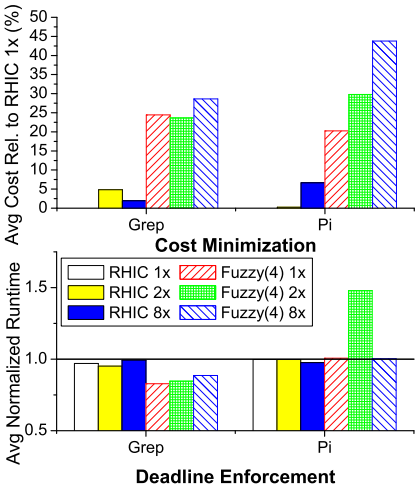


Figure 12: Impact of increased interactive churn on RHIC and Fuzzy(4)

oriented volunteer harvesting. Compress could conceivably be used on user messages, profile data etc., motivated by 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.

Foreground Workloads: NCSU's VCL is an excellent model of an interactive-user IaaS cloud, and we drew on it for ideas about "typical" clouds of this nature. To determine what were the most popular applications used in the VCL, we analyzed a log of 750,000 reservations from 2004-2010 and selected four representative workloads: Matlab, Photoshop, OpenOffice, and C Development. We then instrumented images of these types and collected over 600 resource consumption traces of real users, ranging in length from 20 minutes to 4 hours. Finally, we built a replay framework that can generate CPU and memory load using microbenchmarks to match the consumption in a recorded trace. In addition to trace replay on individual nodes, we also built support infrastructure to distribute traces, randomize their start points, and repeatedly replay the same set of randomized traces across a fleet of foreground VMs. We use different randomized foreground "mixes" for each group of experiments. The length and *churn rate* of the foreground VM sessions are generated using a normal distribution with the parameters derived from the 2004-2010 VCL log data. Finally, we set static memory allocations for foreground VMs using per-workload-type normal distributions extracted from the VCL logs.

Test Platform: Our main test platform is NCSU's ARC cluster, which 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 IP over 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. ARC is a scientific computing cluster and therefore is *top-heavy* in terms of compute to disk I/O resources, which actually makes this environment *more challenging* for RHIC. A more-robust I/O subsystem relative to compute and memory would yield greater scalability, higher I/O saturation points, and less-flat cost and energy curves (§5.2).

We chose to use ARC because it has both reasonable size and node-attached power meters. To calculate background power consumption we replay the foreground workload by itself and calculate the difference. For monetary cost evaluation, unless otherwise noted, we adopt a sample pricing policy following the 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 volunteers, although we calculate cluster costs to the nearest second due to the short duration of our test jobs.

5.2 Exhaustive Evaluation

First, we performed an exhaustive evaluation over the volunteer cluster size range, for each MapReduce test workload. 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, with over 600 worker processes. We collected exhaustive datapoints every 2 volunteers, from $\{0, 2, \ldots, 36\}$, 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 four foreground workloads described in §5.1 (25% each). This seeded mix allowed us to collect foreground-only energy consumption and subtract it from the total, calculating the background energy curves shown in Fig. 16. 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 mechanism (§4.2) as RHIC. 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 set.

Fig. 11 and 16 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 yield a large range in execution costs, generating 72% monetary savings 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 size automatically. On average, RHIC achieves within 5% of the minimum cost and 3% of the minimum energy. 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

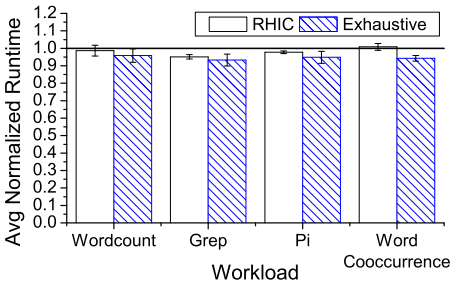


Figure 13: RHIC vs exhaustive: Deadline enforcement. Values just under 1.0 are ideal, but above 1.0 are missed deadlines.

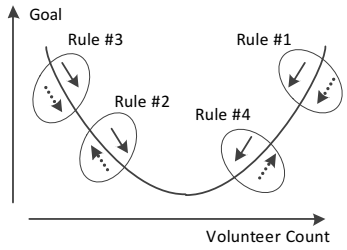


Figure 14: Fuzzy rules. Original figure credit to Liu et al. [29]. Dotted lines represent the observed change and solid lines represent the resulting action.

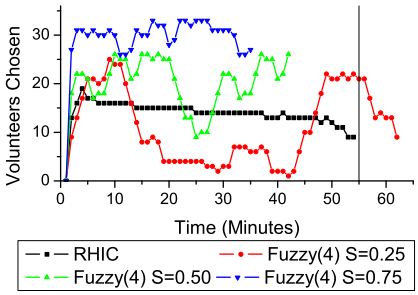


Figure 15: FUZZY's example choices for Pi. The vertical black bar shows the deadline..

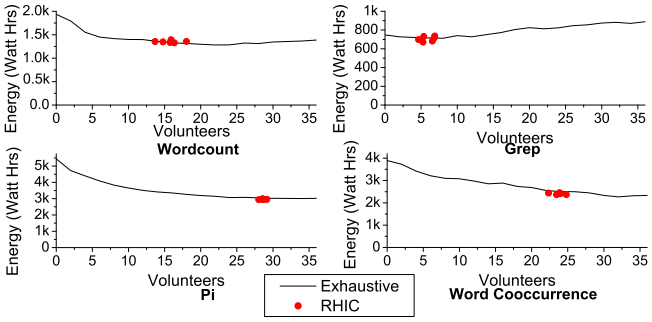


Figure 16: RHIC vs exhaustive: Energy minimization. Volunteer count is the time-weighted average over the job.

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 bottlenecks do not occur. This is exacerbated by the long straggler phase in Co-occurrence, during which most volunteers sit idle.

Fig. 13 shows soft deadline enforcement results. Three deadlines were chosen for each background workload, across the range of achievable completion times, each tested twice for 6 total datapoints per workload. In Fig. 13, the horizontal black bar marks the normalized deadline (@1.0). The exhaustive bar represents the closest setting, identified by the exhaustive tests, which achieves the deadline. Again, RHIC achieves near-ideal performance in most cases, enforcing runtimes 2% under the deadline on average. It misses 5 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 need for continual adjustment, a control theory approach could be a valid alternative. In this section, we compare RHIC with 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 struggle 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 by Liu et al. [29], which is used as the basis for our fuzzy controller (FUZZY) design. Its 2-period historical comparison is similar to hill-climbing.

For Liu et al.'s Rules #1,3 we increment/decrement by a parameter $p > 1$ (shown in Fig. 17 as Fuzzy(p)), which determines the magnitude of cluster size changes when the controller believes it is moving in the "correct" direction, while for Rules #2,4 we only increment/decrement by a single volunteer, since FUZZY is near the minimum. To adapt the minimizing fuzzy controller design to our scenario, we use the interactive node selection and cluster management (§4.2) modules from RHIC. 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. 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 Fig. 15.

In addition, we included a naïve "threshold" algorithm, which chooses interactive nodes with residual resource availability above a percentage - i.e., Threshold(0.5) selects all volunteers with $\geq 50\%$ predicted available resources. 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. One deadline was chosen for each background workload, in the middle of its achievable completion time range. For FUZZY we varied $p = \{2, 4, 8\}$, while for the naïve threshold algorithm we used thresholds of 25%, 50% and 75%, but 75% is omitted due to its universally poor performance.

From Fig. 17, we see that alternative schemes yield worse cost and energy minimization performance relative to RHIC, and RHIC enforces deadlines much more tightly. While some alternative schemes deliver near-RHIC minimization results ($< 5\%$ additional cost/energy) for some workloads, none consistently do so across all background workloads and goals. For example, Fuzzy(8) performs well on Grep and Word Co-occurrence deadlines and most energy minimization, but delivers poor results on Grep and Pi cost minimization, Grep energy minimization, and Wordcount and Pi deadline enforcement. RHIC's *adaptability to both the workload and the desired performance goal* clearly offers a broad advantage. The only place where RHIC underperforms any of the alternative schemes (by 2% at most) is in energy minimization for Word Co-occurrence, for the same reason as discussed in §5.2.

Several insights about these results are worth mentioning:
FUZZY's poor decision-making: this stems from two root causes.

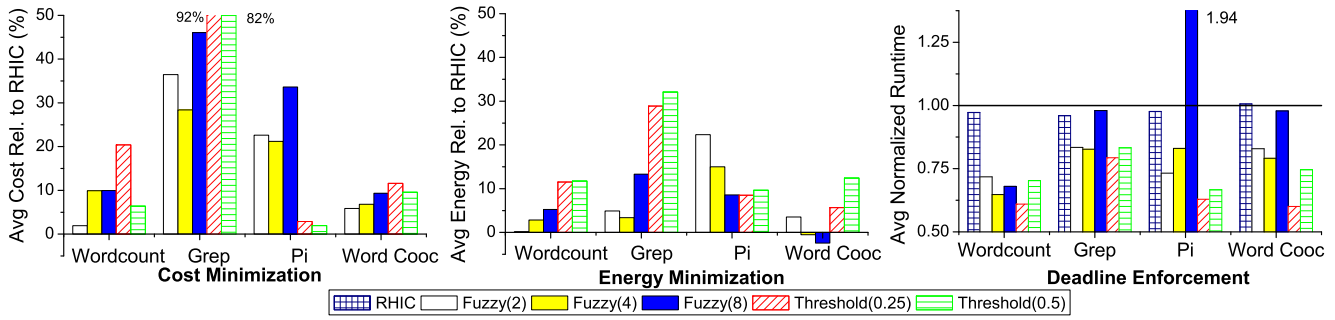


Figure 17: Performance of alternative schemes to RHIC. For cost and energy, lower values are better, and 0% is RHIC's performance. For deadlines, values just below 1.0 are best, and values above 1.0 indicate missed deadlines.

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.

Threshold is goal-oblivious: this yields arbitrary performance, solely dependent on cost, energy and runtime curves (as seen in Fig. 11). Due to this property, we ran only one batch of Threshold runs and used the performance for all goal settings, because changing the goal would make no difference in management behavior. While this algorithm is much simpler in implementation than RHIC, it is entirely inflexible and will suffer greatly from unfriendly performance landscapes, as shown in other work [19].

Alternate schemes finish far before deadlines: simply put, the administrator has requested a deadline of X, and runtimes which are much earlier than this deadline ($< X \times 90\%$) allocate too many volunteers and thus waste resources which could be used for other background jobs. Avoiding wasting residual resources a second time (after the foreground workload already neglected to use them) is a key motivation of our work, which is why RHIC tightly hugs the deadline whenever possible. Of all alternative schemes, none deliver tightly-coupled deadline enforcement except Fuzzy(8), which suffers from large deadline overruns with Pi and minimization inefficiency elsewhere.

5.4 Impact of Environment Stability

In all preceding experiments, we used the VCL's natural churn rate, described in §5.1. In this section, we attempt to quantify the impact of *increased churn* on RHIC's ability to conduct cluster sizing optimization. In Fig. 12 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 FUZZY parameter found in §5.3. Due to length limits, we show only the most I/O-intensive (Grep) and the most CPU-intensive (Pi) background workloads.

Overall RHIC clearly outperforms FUZZY in both minimization performance and deadlines, largely due to the decision-deferment technique discussed in §4.2. More specifically, RHIC is resilient to the high interactive node turn-over rates and achieves near-identical

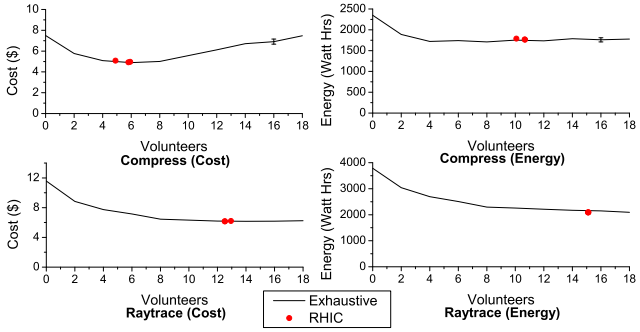


Figure 18: Minimization performance for HBasePCF. Volunteer count is the time-weighted average over the job.

cost minimization performance in all cases but Pi with $8\times$ churn. This exception is because the volunteer pool is smaller on average at such high churn rate: we begin all experiments with 100% of interactive nodes available, and many nodes become absent by the end of the job. For the CPU-intensive Pi, it is impossible to reach the cluster size producing the minimum cost.

5.5 Other Background Frameworks

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 runs on top of HDFS hosted on the dedicated nodes. We used HBasePCF to perform one I/O-intensive and one compute-intensive job (§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 $\$0.42/hr$. Fig. 18 shows the cost and energy minimization performance of RHIC alongside an exhaustive search with volunteer counts of $\{0, 2, \ldots, 18\}$. As with Hadoop, RHIC achieves near-minimum performance for both I/O and compute-intensive workloads, with 1% average error for cost and 2% for energy.

5.6 Hardware Heterogeneity

To evaluate RHIC's approach to handling hardware heterogeneity, we ran Pi and Grep on three different clusters $C_0 \ldots C_2$ with identical inputs. Each cluster has a different processor generation, number of cores, foreground CPU consumption and I/O subsystem. We set C_0 as the base class and determined the accuracy of our equivalency layer as follows: after the first period, we predicted the runtime T_{direct} of each cluster independently using RHIC's base methods described in §4.4. Simultaneously, we predicted the runtime T_{equiv} of each cluster using its equivalency metric E_K from Eq. 7, C_0's runtime and the ratio between actual productivity

P_{actual} on the two clusters. If E_K is representative of the difference between the clusters in processing power per unit of P_{actual}, and the difference in P_{actual} is accounted for, we should find that T_{direct} and T_{equiv} are very close. Between these two predictions we achieved 0.9% error for Pi and 2.0% error for Grep, demonstrating that our equivalency calculation is sufficient to translate performance between multiple machine classes.

5.7 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 1 of 16 cores, or 2% overall) 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 (§4.1). Since cluster sizing decisions are made once per evaluation interval (1 minute), all the periodic resource and job progress monitoring incurs less than 0.5% overhead on volunteer or candidate interactive nodes.

6. CONCLUSION AND FUTURE WORK

In conclusion, we have outlined 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 across a range of workloads and goals, with 5% average error for cost minimization and 3% for energy, relative to exhaustive searches, and delivers runtimes 2% under deadlines. With RHIC, we have found that it is possible to tolerate the high degree of instability in interactive clouds and run jobs with no *a priori* knowledge of either the foreground or background workloads. Finally, RHIC requires only system-level metrics and a progress score, yielding 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 ideal hybrid cluster compositions for a given workload and performance goal, scaling both the dedicated and volunteer nodes with topology awareness. Further, we plan to extend our system to flexibly harvest more resource types, including memory and network bandwidth.

7. REFERENCES

[1] S. Agarwal, S. Kandula, N. Bruno, et al. Re-optimizing data-parallel computing. In *NSDI '12*.

[2] Amazon. Elastic Block Store. `aws.amazon.com/ebs/`.

[3] Amazon. Elastic Compute Cloud. `aws.amazon.com/ec2/`.

[4] D. Anderson. Boinc: A system for public-resource computing and storage. In *Grid '04*.

[5] D. Arnold, G. Pack, and B. Miller. Tree-based overlay networks for scalable applications. In *IPDPS '06*.

[6] A. Chandra and J. Weissman. Nebulas: Using distributed voluntary resources to build clouds. In *HotCloud '09*.

[7] Y. Chen, S. Alspaugh, and R. H. Katz. Design insights for mapreduce from diverse production workloads. T. R. EECS-2012-17, UCB.

[8] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Trans. Pattern Anal. Mach. Intell.*, 1995.

[9] N. Chohan, C. Castillo, M. Spreitzer, et al. See spot run: using spot instances for mapreduce workflows. In *HotCloud'10*.

[10] Cisco. Enterprise Virtual Desktop Infrastructure: Design for Performance and Reliability. `http://cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns377/white_paper_c11-541004_R2_v7.pdf`.

[11] Cisco. Solution for Citrix VDI-in-a-Box. `http://cisco.com/en/US/solutions/collateral/ns340/ns517/ns224/ns836/ns978/solution_overview_c22-716452.pdf`.

[12] T. Deshane, Z. Shepherd, J. N. Matthews, et al. Quantitative comparison of xen and kvm. In *Xen Summit '08*.

[13] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07*.

[14] B. Farley, A. Juels, V. Varadarajan, et al. More for your money: exploiting performance heterogeneity in public clouds. In *SOCC '12*.

[15] A. D. Ferguson, P. Bodik, S. Kandula, et al. Jockey: Guaranteed job latency in data parallel clusters. In *EuroSys '12*.

[16] A. S. Foundation. Hadoop. `http://hadoop.apache.org/`.

[17] A. S. Foundation. HBase. `http://hbase.apache.org/`.

[18] A. Gupta, B. Lin, and P. Dinda. Measuring and understanding user comfort with resource borrowing. In *HPDC '04*.

[19] H. Herodotou, F. Dong, and S. Babu. No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *SOCC '11*.

[20] B. Hindman, A. Konwinski, M. Zaharia, et al. Mesos: a platform for fine-grained resource sharing in the data center. In *NSDI'11*.

[21] G. Lee, B.-G. Chun, and R. H. Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *HotCloud '11*.

[22] J. Li, A. Deshpande, J. Srinivasan, et al. Energy and performance impact of aggressive volunteer computing with multi-core computers. In *MASCOTS '09*.

[23] B. Lin and P. Dinda. Towards scheduling virtual machines based on direct user input. In *VTDC '06*.

[24] B. Lin and P. Dinda. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In *SC '05*.

[25] H. Lin, X. Ma, J. Archuleta, et al. Moon: Mapreduce on opportunistic environments. In *HPDC '10*.

[26] M. Litzkow, M. Livny, and M. Mutka. Condor-a hunter of idle workstations. In *DCS '88*.

[27] H. Liu. Cutting mapreduce cost with spot market. In *HotCloud '11*.

[28] H. Liu and D. Orban. Cloud MapReduce: A MapReduce Implementation on Top of a Cloud Operating System. In *CCGrid'11*.

[29] X. Liu, L. Sha, Y. Diao, et al. Online response time optimization of apache web server. In *IWQoS'03*.

[30] A. Mashtizadeh, E. Celebi, T. Garfinkel, et al. The design and evolution of live storage migration in VMware ESX. In *USENIX '11*.

[31] NCSU. NCSU Virtual Computing Lab. `vcl.ncsu.edu/`.

[32] J. Polo, C. Castillo, D. Carrera, et al. Resource-aware adaptive scheduling for mapreduce clusters. In *Middleware '11*.

[33] J.-A. Quiane-Ruiz, C. Pinkel, J. Schad, et al. Rafting mapreduce: Fast recovery on the raft. In *ICDE '11*.

[34] Z. Shen, S. Subbiah, X. Gu, et al. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *SOCC '11*.

[35] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[36] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *OSDI '02*.

[37] VMware. VDI Server Sizing and Scaling. `http://vmware.com/pdf/vdi_sizing_vi3.pdf`.

[38] G. Wang, A. Butt, P. Pandey, et al. A simulation approach to evaluating design decisions in mapreduce setups. In *MASCOTS '09*.

[39] A. Wieder, P. Bhatotia, A. Post, et al. Orchestrating the deployment of computations in the cloud with conductor. In *NSDI '12*.

[40] T. Wood, P. Shenoy, A. Venkataramani, et al. Black-box and gray-box strategies for virtual machine migration. In *NSDI '07*.

[41] L. Yu and D. Thain. Resource management for elastic cloud workflows. In *CCGrid '12*.

[42] M. Zaharia, A. Konwinski, A. D. Joseph, et al. Improving mapreduce performance in heterogeneous environments. In *OSDI '08*.