2013 IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems

Accelerating Batch Analytics 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 savings in our testbed. For a given background job, RHIC intelligently discovers and 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 with 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 and compositions across a wide range of workload/goal combinations.**

***Keywords*—Distributed computing; Performance analysis; Adaptive systems;**

I. 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) [1] is a production cloud system hosting virtual desktops 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. They also yield significant *residual*, or unused, resources, due to overprovisioning and the bursty, unpredictable nature of interactive workloads.

By aggressively harnessing such residual resources, cloud providers can 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]. However, traditional techniques such as virtual machine (VM) packing [3] cannot be performed aggressively here, 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 §III-A.

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 for background jobs will suffer from performance and stability issues. Prior studies [4], [5], [6] have proposed a hybrid batch cluster design where *volunteer* (harvesting) nodes supplement a core set of stable *dedicated* nodes, in some cases using EC2 SPOT instances in the volunteer role. 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 orthogonal temporal characteristics (as we show later), and has been described previously [2], [7]. In our preliminary experiments (§III-B), we demonstrate 20-29% energy and 20-40% cost gains over normal dedicated clusters with only 1% average slowdown of interactive workloads.

We propose 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 accounts for interactive users' bursty resource consumption, which easily leads to high migration cosst, as shared nothing clouds such as VCL lack robust shared storage (like Amazon's Elastic Block Store). 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. For example, in contrast to passive volunteer MapReduce computing environments described by prior work [6], interactive nodes are much shorter-lived and unlikely to return in the near future. Also, the hybrid cluster design provides a performance baseline to mitigate stragglers, caused by bursty and unreliable volunteers, via speculative execution of delayed volunteer tasks on 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.

1526-7539/13 $26.00 © 2013 IEEE
DOI 10.1109/MASCOTS.2013.63

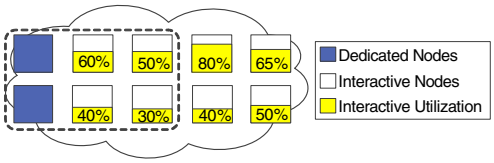


Fig. 1: Sample hybrid cloud computing system, with 8 interactive nodes running interactive services. A background job runs on 2 dedicated and 4 volunteer nodes.

	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

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

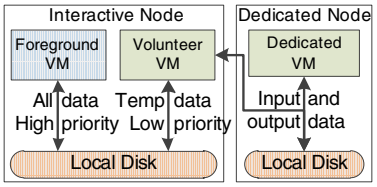


Fig. 3: Disk layout in the hybrid cluster design. The hypervisor is used to prioritize foreground disk access.

For example, Fig. 2 shows the diverse range of monetary costs and energy consumption among different batch workloads. These results are highly dependent on the specific batch inputs, foreground workloads and pricing structure chosen, as well as cluster hardware, network and energy characteristics.

Existing work has addressed several related problems, including MapReduce cluster sizing [8], [9], [10], [11], volunteerism/hybrid clusters for MapReduce [4], [5], [6] and workload consolidation [3]. 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.

In this paper, we present 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 delivers accurate performance estimates and quickly discovers the best cluster size for novel workloads. Our major contributions:

- 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 job performance models which map aggregate residual resources to 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.

II. Related Work

Our work is related to contributions from several other areas:

Volunteer computing. Volunteer computing (VC), known widely through projects such as Condor [12] and BOINC [13], 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 [2], [14], 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 [14] and delivering consistent background performance using unreliable residual resources [15].

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 recent works perform cluster sizing for parallel batch workloads [8], [9], [10], [11], [16]. Of these, our efforts are most-closely related to those which combine modeling with online adjustment and feedback [8], [10], [11]. Jockey [8] 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 [11] also combines modeling and online adjustment to meet deadlines and minimize cost for MapReduce, taking into account data upload and migration overheads. RAS [10] 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 [9] 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. [16] describe a system for modeling batch workload performance 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 [8], [9], [10], [11], [16] will not adequately capture the performance of a given cluster. Our insight regarding aggregate residual CPU availability and its direct effect on cluster performance (§IV-D) led to RHIC's CPU-centric modeling approach. Further, hybrid clusters have significant I/O restrictions because dedicated nodes provide the only persistent storage. We take a unique approach to discovering and modeling I/O bottlenecks (§IV-C) in response. Wieder et al. [11] 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. [16] 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 [8], [17], [18]) 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 [8], [9], [10], [16] or key performance characteristics [11]. While those with online adjustment [8], [10], [11] could adapt to some deviation from the profile performance (as Wieder et al. [11] demonstrate), the dynamic nature of volunteer heterogeneity directly inspired RHIC's online learning and reactive approaches to CPU (§IV-C) and I/O (§IV-D).

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 [10], [11] to specific MR frameworks [8], [9]. Because we envision RHIC as a harvesting platform which manages throughput-oriented parallel batch jobs, we built it to be workload-independent and evaluate this capability (§V-D). Further, because volunteers are lightweight and transient, we believe RHIC could be applied to multi-stage jobs [8] by managing each stage independently.

Hybrid MapReduce, Volunteerism and Cluster Sharing. Prior works use Amazon EC2 Spot Instances to perform MapReduce jobs [4], [5], [19], whose transience is similar to interactive cloud nodes. Two approaches have been 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 [19]. 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. [5] highlight a similar problem space to our work, they have not proposed any concrete solution for automatically determining ideal cluster size.

MOON [6] enhanced Hadoop to operate under passive volunteerism, where a foreground workload and MapReduce are interleaved temporally but not spatially. Mesos [18] is a framework for batch framework co-location above a shared distributed filesystem. Both works are orthogonal to ours: they do not consider our target scenario, with two workloads asymmetrically sharing resources, or perform cluster sizing.

Workload Consolidation. Co-locating workloads on the same physical host is a well-established technique [3] that is 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".

III. Background

As mentioned earlier, we leverage a *hybrid* cluster design [4], [5], [6] to harvest residual resources. In §III-A, we justify our cluster design choice by showing that it is appropriate for our environment. Then, in §III-B we validate assumptions regarding the feasibility and profitability of adopting this approach.

A. Hybrid Cluster Design Rationale

Our hybrid design is motivated by an analysis of interactive cloud workloads observed in the VCL: 600 real user traces (described in §V-A) and reservation metrics from 750,000 sessions during 2004-2010. We found that user reservations are both fairly long and have very high variances ($CoV = 1.04$) indicating unpredictable session lengths. Further, average CPU utilization is low (3-22% on average) and bursts are quite short-lived (2-47s on average), even for computate-intensive workloads like Matlab. Further analysis of these traces is given in our tech report [20].

Such highly dynamic behavior renders traditional approaches such as workload consolidation [3] less appealing. Conservative consolidation can maintain interactive users' QoS requirements but will inevitably waste resources. Aggressive approaches, on the other hand, may face severe performance penalties in the case of resource conflicts. Although live migration is possible both with shared and non-shared storage, 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

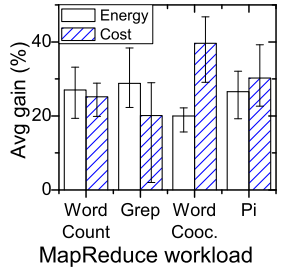


Fig. 4: Energy and cost savings by using a hybrid cluster design, over a regular dedicated-only cluster. Error bars represent the range of savings.

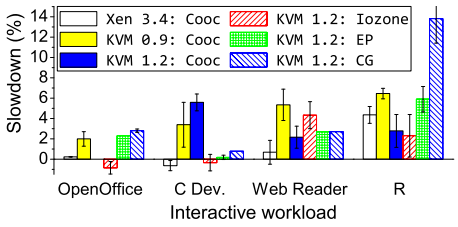


Fig. 5: Slowdown of interactive foreground workloads padded with volunteers. Foreground workloads include members of `bltk` and AT&T's R benchmark. Background workloads include several resource-intensive benchmarks: Word Cooccurrence (Cooc), Iozone, and NAS PB (EP, CG).

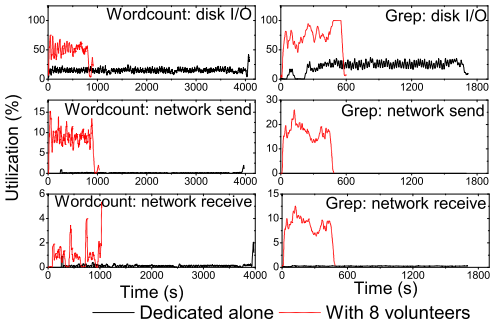


Fig. 6: Disk and network bandwidth utilization on 2 dedicated nodes, with and without 8 volunteers. Disk utilization is measured by the % of time the CPU spent blocked on I/O.

design keeps volunteer nodes lightweight and agile, making it much easier to use/discard a node due to foreground demand shifts. Further, through mechanisms such as task replication and reliable dedicated nodes, this hybrid design can aggressively harvest residual resources while mitigating stragglers.

B. 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 §V-A. 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). The hybrid cluster design delivers 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 [21], partially demonstrated by today's high VDI densities [22]. We verified this by testing co-located foreground and background VMs under work-conserving schedulers in the Xen and KVM hypervisors, with the foreground given the maximum CPU, disk and network priority, and the background VM minimum. We ran 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. Fig. 5 indicates that the performance impact is low despite virtual desktop applications' sensitivity to I/O latency. Xen yields an average slowdown of 1%, while KVM 1.2 delivers $< 6\%$ slowdown with all combinations except R paired with CG, due to CG's high memory bandwidth demand. To our knowledge, no hypervisor currently arbitrates memory bandwidth.

3. Dedicated nodes have sufficient residual disk bandwidth to offload computation to volunteers. Figure 6 plots the disk and network utilization (collected with the `iostat` and `dstat` tools respectively) of 2 dedicated nodes, with and without 8 volunteers, for the two most I/O-intensive workloads in our MapReduce test set. It illustrates that (1) substantial disk and network bandwidth is available on dedicated nodes, (2) using volunteers significantly speeds up the job execution, and

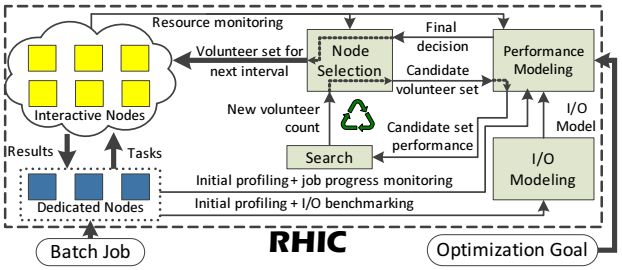


Fig. 7: RHIC components and data flow

(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.*

IV. Framework Design

A. 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. 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 profiling phase, defaulted to 1 minute, 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*, defaulted to 1 minute in length. 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 (§IV-B). Their predicted resource availability is supplied as input to the I/O model, generated by the *I/O modeling* component (§IV-C), which 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 (§IV-D). The best candidate volunteer pool is used until the end of the interval, when the process repeats. 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 which typically finds near-optimum cluster sizes within 3-4 periods.

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 §IV-B and §IV-C, and how RHIC uses it to model workload performance in §IV-D.

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 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 in the near future 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 discussion is based on homogeneous hardware across the node pool. However, in our tech report [20], we describe and evaluate a thin translation layer that allows RHIC manage and model different physical node types with very low error ($\leq 2\%$).

B. 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 [3] 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 near-future CPU utilization can be effectively estimated using short-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 several common prediction methods under a range of history / prediction window lengths simulating our environment, and evaluated them on the foreground traces we use (§V-A), with extended details given in our tech report [20]. On the basis of this evaluation, we 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.

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 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 discard any nodes that lack the minimum memory required.

Put together, the predicted foreground CPU consumption (CPU_{fg}) and 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), and distinguish this quantity as *potential* because I/O bottlenecks may result in a lower *actual* productivity, as we discuss in §IV-C. 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 savings, 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 §IV-D. 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 and limits the search to a linear rather than exponential space, in regard to the candidate interactive node pool size. Interactive node churn presents an issue for our search-driven cluster sizing scheme, because nodes can arrive/leave unexpectedly, i.e. at the end of a class lab session. 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 (§III-A). 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.

C. Modeling Workload I/O Behavior

As verified in §III-B, 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, eventually I/O bandwidth on dedicated nodes is likely to become the chief limiting factor for scalability (Insight 2), which has not been considered in prior work [4], [5].

Fig. 8 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. 8b plots the actual to potential productivity ratio, showing 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. It is critical to estimate the I/O saturation point, beyond which more volunteers will not yield additional performance, to avoid extrapolation and needlessly over-subscribing the dedicated nodes' I/O subsystems. 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 §IV-E), 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 potential productivity on a node. $Vol\ P_{supp}$ is the volunteer productivity *each* dedicated node could support, in addition to its own demand.

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

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:

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

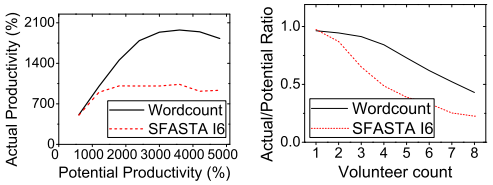
Using the best and worst case estimates, RHIC intelligently increases the size of the volunteer pool using the following approach (algorithm given in our tech report [20]): it samples the worst case estimate, halfway between the best and worst case, and then uses linear regression to guess the actual saturation onset point. RHIC then verifies the occurrence of I/O saturation using the disk sensors on the dedicated nodes. 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.

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 subsets 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.

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 a combination of clustering and spline fitting 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 only 384Kbps on average [22]) but will be relaxed in future work, to incorporate hotspot detection, topology awareness and bandwidth availability prediction.

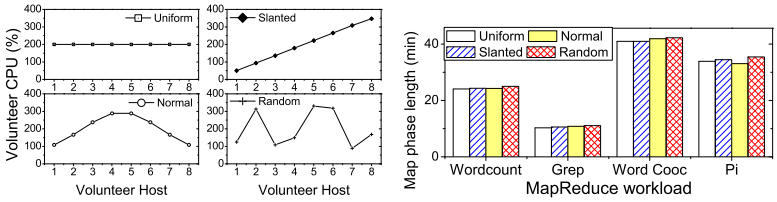
D. Background Job Performance Modeling

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 §IV-E). Fig. 9 shows 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%-



(a) Actual (b) Relative

Fig. 8: Impact of I/O bottlenecks on productivity, using 2 dedicated and 1-8 volunteers.



(a) Volunteer resource distributions (b) Job completion times

Fig. 9: Impact of residual resource distribution on job completion time for a hybrid cluster. All distributions have the same total residual CPU.

350% (with one core = 100%), while the total CPU allocations for all 8 volunteers are fixed at 1600%.

Fig. 9b 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 [23]. 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 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. 9 only demonstrates static CPU allocation heterogeneity, we show in our evaluation that this technique can be successfully applied to dynamic heterogeneity.

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 (§IV-B) 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_{act} is the aggregate actual productivity (dedicated + volunteer) over $T_{elapsed}$. R_{proc} is re-evaluated periodically during the background job.

$$R_{proc} = \frac{J_{completed}}{AP_{act} \times T_{elapsed}} \quad (5) \qquad T_{rem} = \frac{J_{rem}}{AP_{pred} \times R_{proc}} \quad (6)$$

By calculating the fraction of remaining work $J_{rem} = J_{total} - J_{completed}$, we can then invert Eq. 5 and produce a completion time estimate T_{rem}, given a predicted aggregate actual productivity AP_{pred}, as shown in Eq. 6. AP_{pred} 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, we add a small padding value to our runtime estimate to improve straggler tolerance and incorporate transition times by profiling volunteer launches/shutdowns. Overall, in our experimentation we found this runtime modeling approach simple but effective.

Goal Estimation: Based on the completion time estimate, RHIC generates performance scores (to be minimized) for candidate volunteer sets, given a goal:

(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:* 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 a hardware-specific energy model. In this paper, we take the well-established approach of running a micro-benchmark to enumerate the relationship between CPU utilization, frequency and power consumption, and then apply multiple regression to derive a power model. 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. While all power consumption on dedicated nodes is billed to the background user, he/she is only charged for the *incremental* energy consumption incurred by the background job on the volunteer nodes, because these nodes would be powered on anyway to host interactive users.

E. Limitations of Linearity Assumptions

The aforementioned performance modeling is dependent on R_{proc} (Eq.5) 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 §V-C), workloads that have inherently heterogeneous progress can reduce RHIC's accuracy. We believe that this shortcoming can be addressed with offline profiling, which would allow us to distill how much R_{proc} varies in the given workload.

F. Integrating RHIC into MapReduce

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

Multi-tenancy: MR 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 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. Each copy of

RHIC harvests from a shared pool of interactive nodes, which are traded between jobs as demand changes.

Volunteer termination: The lifetime of a volunteer is equal to the lifetime of the interactive node which it resides on, and abrupt termination poses a problem for Hadoop because the JobTracker assumes that tasks completed by the terminated node are lost. Several fixes for this issue have been proposed, including checkpoint-restart, pre-emptively pushing intermediate data to Reducers or placing it on a distributed filesystem. We emulate these features using a modified scheduler.

Reducer placement: The loss of Reduce tasks is particularly damaging to MR job runtimes because intermediate data must be re-shuffled [4]. 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 [24] that Map-only jobs are common, the Map phase dominates MapReduce jobs, and input data is the majority of stored bytes.

V. Experimental Evaluation

In this section we evaluate RHIC after giving an overview of our test platform in §V-A. First, in §V-B, we establish that RHIC can accurately discover near-ideal cluster sizes. In §V-C, we compare the performance, stability, and adaptability of RHIC to an alternative algorithm based on fuzzy control theory. Finally, we outline additional evaluations in §V-D.

A. Test Workloads, Platform, and Settings

Background Workloads: For evaluating RHIC we use Hadoop with four representative 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 20-40 minutes.

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 the `stress` microbenchmark to match the consumption in a recorded trace. We use different randomized foreground "mixes" for each group of experiments. The length and *churn rate* of the foreground VM sessions, along with static memory allocations, are generated using normal distributions with the parameters derived from VCL log data.

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 and the KVM hypervisor. We use IP over Infiniband for our experiments, but due to virtualization overhead can only achieve approximately 500 MBit/sec speeds (VM to VM). ARC has a high of ratio of compute to I/O resources, which makes this environment *more challenging* for RHIC: a more-robust I/O subsystem would yield greater scalability and less-flat cost and energy curves (§V-B). 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. We adopt a monetary pricing policy following the costs of EC2 `m2.xlarge` On-Demand and SPOT Instances at the time of writing: $1.00/hour for dedicated nodes and $0.42/hour for volunteers, calculated to the nearest second.

Evaluation: 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 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 $\mu \geq 2\%$.

B. Exhaustive Evaluation

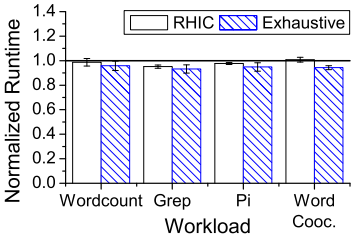


Fig. 10: RHIC vs exhaustive: Deadline enforcement. Values just under 1.0 are ideal, but above 1.0 are missed deadlines.

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 §V-A (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. 12. 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 (§IV-B) as RHIC. This ensures that if RHIC and the targeted framework choose X interactive nodes at the same point in the background job, they receive the same set.

Fig. 11 and 12 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, because Co-occurrence has non-

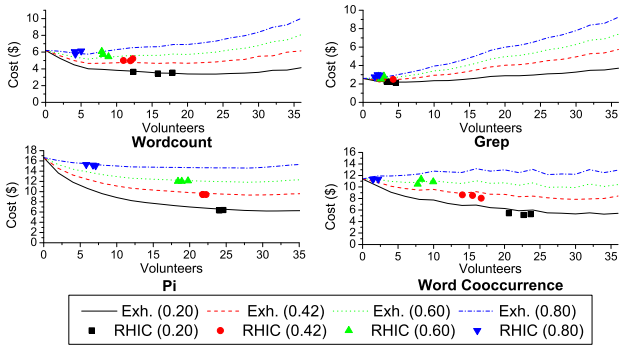


Fig. 11: RHIC vs exhaustive: Cost minimization. Dedicated nodes are fixed at C_d = $\$1.00/hr$, with four different volunteer rates: $C_v = \{\$0.20, \$0.42, \$0.60, \$0.80\}/hr$, represented as Exh.($C_v$) and RHIC($C_v$). Volunteer count is the time-weighted average over the job.

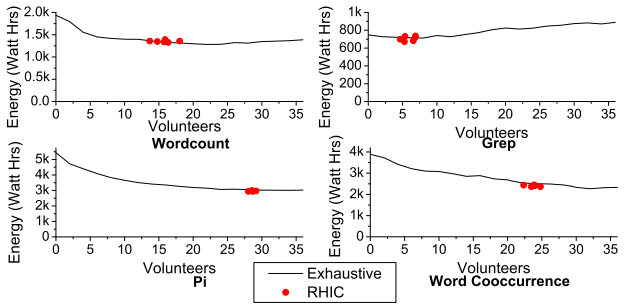


Fig. 12: RHIC vs exhaustive: Energy minimization. Volunteer count is the time-weighted average over the job.

trivial I/O demand which RHIC cautiously explores, and a long stragger phase during which most volunteers sit idle.

Fig. 10 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. 10, 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.

C. 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. [25], which we use as the basis for our *FUZZY* controller design (full design details in our tech report [20]). Its 2-period historical comparison is similar to hill-climbing.

When FUZZY believes it is moving in the "correct" direction (Liu's rules #1,3), we increment/decrement cluster size by a parameter Fuzzy(p>1), otherwise incrementing/decrementing by a single volunteer when the minimum is nearby (rules #2,4). 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 Threshold we used thresholds of 25%, 50% and 75%, but 75% is omitted due to its universally poor performance.

From Fig. 13, 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 a subset of workload / goal combinations, but delivers inconsistent and poor results elsewhere. 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 §V-B. Several insights about these results are worth mentioning:

FUZZY's poor decision-making: this 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, which is also very noisy. We opted against adding this capability to FUZZY, under the reasoning that it would simply shift the unreliability issue elsewhere.

Threshold is goal-oblivious: this yields arbitrary performance, solely dependent on cost, energy and runtime curves (as seen in Fig. 11). While this algorithm is much simpler in implementation than RHIC, it is inflexible and will suffer greatly from unfriendly performance landscapes.

Alternate schemes finish far before deadlines: runtimes which are much earlier than the deadline allocate too many volunteers and thus waste resources which could be used for other background jobs. RHIC tightly hugs the deadline whenever possible to avoid wasting residual resources.

D. Extended Evaluation

Here we give a brief overview of additional experiments, with detailed descriptions given in our tech report [20]:

Overhead: RHIC's control node consumes less than 2% CPU on average and takes less than $250ms$ to make an exhaustive

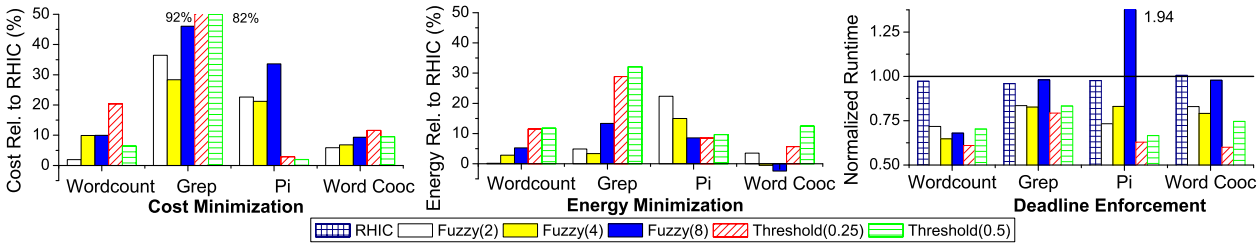


Fig. 13: Performance of alternative schemes to RHIC. For cost and energy, lower values are better, and 0% is RHIC's performance. For soft deadlines, values just below 1.0 are best, and values above 1.0 indicate missed deadlines.

cluster sizing decision for 36 volunteers.

Impact of Environment Stability: we increased the interactive node churn rate to 2× and 8× normal and evaluated RHIC against FUZZY. FUZZY delivered cost overruns of 25%-45% relative to RHIC and inconsistent deadline performance.

Other Background Frameworks: we used RHIC to manage a lightweight compute framework on top of HBase, exporting only a progress score, and achieved near-minimum performance for both I/O and compute-intensive workloads: 1% average error for cost and 2% for energy.

Hardware Heterogeneity: we built a thin translation layer to aid RHIC in managing heterogeneous hardware, based on capturing R_{proc} and I/O bandwidth equivalency metrics between different node classes, which yielded $\leq 2\%$ modeling error for both I/O and compute-intensive workloads.

VI. 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 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.

Acknowledgements

We appreciate the helpful input from the anonymous reviewers. This work has been supported in part by NSF grants 0546301 (CAREER), 0915861, and 0958311, two IBM Faculty Awards, a Graduate Merit Award from the College of Engineering at NC State University, as well as a joint faculty appointment between Oak Ridge National Laboratory and NCSU. Any opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF or U.S. Government.

References

[1] NCSU, "NCSU Virtual Computing Lab," vcl.ncsu.edu/.
[2] J. Li, A. Deshpande, J. Srinivasan *et al.*, "Energy and performance impact of aggressive volunteer computing with multi-core computers," in *MASCOTS '09*.
[3] T. Wood, P. Shenoy, A. Venkataramani *et al.*, "Black-box and gray-box strategies for virtual machine migration," in *NSDI '07*.
[4] N. Chohan, C. Castillo, M. Spreitzer *et al.*, "See spot run: using spot instances for mapreduce workflows," in *HotCloud'10*.
[5] G. Lee, B.-G. Chun, and R. H. Katz, "Heterogeneity-aware resource allocation and scheduling in the cloud," in *HotCloud'11*.
[6] H. Lin, X. Ma, J. Archuleta *et al.*, "Moon: Mapreduce on opportunistic environments," in *HPDC '10*.
[7] B. Lin and P. Dinda, "Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling," in *SC '05*.
[8] A. D. Ferguson, P. Bodik, S. Kandula *et al.*, "Jockey: Guaranteed job latency in data parallel clusters," in *EuroSys'12*.
[9] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics," in *SOCC '11*.
[10] J. Polo, C. Castillo, D. Carrera *et al.*, "Resource-aware adaptive scheduling for mapreduce clusters," in *Middleware'11*, 2011.
[11] A. Wieder, P. Bhatotia, A. Post *et al.*, "Orchestrating the deployment of computations in the cloud with conductor," in *NSDI '12*.
[12] M. Litzkow, M. Livny, and M. Mutka, "Condor-a hunter of idle workstations," in *DCS '88*.
[13] D. Anderson, "Boinc: A system for public-resource computing and storage," in *Grid '04*.
[14] A. Gupta, B. Lin, and P. Dinda, "Measuring and understanding user comfort with resource borrowing," in *HPDC '04*.
[15] A. Chandra and J. Weissman, "Nebulas: Using distributed voluntary resources to build clouds," in *HotCloud'09*.
[16] L. Yu and D. Thain, "Resource management for elastic cloud workflows," in *CCGrid '12*.
[17] S. Agarwal, S. Kandula, N. Bruno *et al.*, "Re-optimizing data-parallel computing," in *NSDI '12*.
[18] B. Hindman, A. Konwinski, M. Zaharia *et al.*, "Mesos: a platform for fine-grained resource sharing in the data center," in *NSDI'11*.
[19] H. Liu, "Cutting mapreduce cost with spot market," in *HotCloud'11*.
[20] R. B. Clay, Z. Shen, and X. Ma, "Building and scaling virtual clusters with residual resources from interactive clouds," NCSU, Tech. Rep. TR-2013-4 (Department of Computer Science).
[21] T. Deshane, Z. Shepherd, J. N. Matthews *et al.*, "Quantitative comparison of xen and kvm," in *Xen Summit '08*.
[22] 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.
[23] M. Zaharia, A. Konwinski, A. D. Joseph *et al.*, "Improving mapreduce performance in heterogeneous environments," in *OSDI '08*.
[24] Y. Chen, S. Alspaugh, and R. H. Katz, "Design insights for mapreduce from diverse production workloads," UCB, Tech. Rep. EECS-2012-17.
[25] X. Liu, L. Sha, Y. Diao *et al.*, "Online response time optimization of apache web server," in *IWQoS'03*.