Cost-aware Cooperative Resource Provisioning for Heterogeneous Workloads in Data Centers

Jianfeng Zhan, Lei Wang, Xiaona Li, Weisong Shi, *Senior Member, IEEE*, Chuliang Weng, Wenyao Zhang, and Xiutao Zang

Abstract—Recent cost analysis shows that the server cost still dominates the total cost of high-scale data centers or cloud systems. In this paper, we argue for a new twist on the classical resource provisioning problem: heterogeneous workloads are a fact of life in large-scale data centers, and current resource provisioning solutions do not act upon this heterogeneity. Our contributions are threefold: first, we propose a cooperative resource provisioning solution, and take advantage of differences of heterogeneous workloads so as to decrease their peak resources consumption under competitive conditions; second, for four typical heterogeneous workloads: parallel batch jobs, Web servers, search engines, and MapReduce jobs, we build an agile system *PhoenixCloud* that enables cooperative resource provisioning; and third, we perform a comprehensive evaluation for both real and synthetic workload traces. Our experiments show that our solution could save the server cost aggressively with respect to the non-cooperative solutions that are widely used in state-of-the-practice hosting data centers or cloud systems: e.g., *EC2*, which leverages the statistical multiplexing technique, or *RightScale*, which roughly implements the elastic resource provisioning technique proposed in related state-of-the-art work.

Index Terms—Data Centers, Cloud, Cooperative Resource Provisioning, Statistical Multiplexing, Cost, and Heterogeneous Workloads.

1 Introduction

More and more computing and storage are moving from PC-like clients to data centers [32] or (public) clouds [21], which are exemplified by typical services like EC2 and Google Apps. The shift toward server-side computing is driven primarily not only by user needs, such as ease of management (no need of configuration or backups) and ubiquity of access supported by browsers [32], but also by the economies of scale provided by high-scale data centers [32], which is five to ten over small-scale deployments [20] [29]. However, high-scale data center cost is very high, e.g., it was reported in Amazon [29] that the cost of a hosting data center with 15 megawatt (MW) power facility is high as $5.6 *M* per month. High data center cost puts a big burden on *resource providers* that provide both data center resources like power and cooling infrastructures and server or storage resources to hosted *service providers*, which directly provides services to end users, and hence *how to lower data center cost is a very important and urgent issue*.

Previous efforts [32] [44] [46] [51] studied the cost models of data centers in Amazon, and Google etc, and concluded with two observations: first, different from enterprise systems, the personnel cost of hosting data center shifts from top to nearly irrelevant [29]; second, the server cost—the cost of server hardware [1] contributes the largest share (more than a half), and the power & cooling infrastructure cost, the power cost and other infrastructure cost follow, respectively. In this context, *the focus of this paper is how to lower the server cost, which is the largest share of data center cost.*

Current solutions to lowering data center cost can be classified into two categories: first, virtualization-based consolidation is proposed: a) to combat server sprawl caused by isolating applications or operating system heterogeneity [50], which refers to a situation that under-utilized servers take up more space and consume more data center, server or storage resources than that are required by by their workloads; b) to provision elastic resources for either multi-tier services [48] [49] or scientific computing workloads [20] in hosting data centers. Second, *statistical multiplexing* techniques [27] are leveraged to decrease the server cost in state-of-the-practice EC2 systems; an overbooking resource approach [39], routinely used to maximize yield in airline reservation systems, is also proposed to maximize the revenue—the number of applications with *light loads* that can be housed on a given hardware configuration. Multiplexing means that a resource is shared among a number of users [58]. Different from peak rate allocation multiplexing

• *J. Zhan and L. Wang, X. Li, X. Zang are with the Institute of Computing Technology, Chinese Academy of Sciences. W. Shi is with Wayne State University. C. Weng is with Shanghai Jiaotong University. As the corresponding author, W. Zhang is with Beijing Institute of Technology.*

1. For a data center with 15 MW power facility in Amazon, the share of server cost, power & cooling infrastructure cost, power cost and other infrastructure cost are 53, 23, 19 and 5 percents, [29] respectively. Internet-scale services typically employ direct attached disks rather than storage-area networks common in enterprise deployments [29], so the cost of storages is included into the server cost. For a real deployment of 50k serves with 40 servers per rack, the network cost is under 3% about the server cost [29].

Digital Object Indentifier 10.1109/TC.2012.103

that guarantees the peak resource demands for each user [58], statistical multiplexing allows the sum of the peak resource demands of each user exceeding the capacity of a data center.

On one hand, as more and more computing moves to data centers, resource providers have to confront with the server sprawl challenge caused by increasing *heterogeneous workloads in terms of different classes of workloads*, e.g., Web server, data analysis jobs, and parallel batch jobs. This observation is supported by more and more case reports. For example, search engine systems, like Nutch (http://nutch.apache.org/), include two major heterogeneous workloads: MapReduce-like parallel data analysis and search engine services The Boeing company also reported this trend of consolidating parallel batch jobs and Web service applications on one data center in the recent IDC HPC user forum, held in October 30, 2010 at Beijing, China. Heterogeneous workloads often have significantly different resource consumption characteristics and performance goals, of which we defer the discussion to Section 3, and hence the server sprawl challenge caused by them can not be simply resolved on the extension of the previous virtualization-based consolidation work. On the other hand, simply leveraging statistical multiplexing or overbooking resources can not effectively cope with the server sprawl challenge caused by increasing heterogeneous workloads. *Constrained by the power supply and other issues, the scale of a data center can not be limitless*. Even many users and services can be deployed or undeployed, it is impossible that the resource curves of the systems that leverage the statistical multiplexing technique, like EC2, will remain smooth. Besides, the overbooking solution to resource provisioning is probabilistic, and hence it is unsuitable for critical services.

In this paper, we gain an insight from the previous data cost analysis [32] [44] [46] [51] to decrease data center costs. From the perspective of a resource provider, the highest fraction of the cost—the server cost (53%) [29] is decided by *the system capacity*, and the power & infrastructure cost is also closely related with the system capacity. Since the system capacity at least must be greater than *the peak resource consumption* of workloads, which is *the minimum system capacity allowed*, and hence we can decrease the peak resources demands of workloads so as to save both the server cost and the power & infrastructure cost.

We leverage differences of heterogeneous workloads in terms of resource consumption characteristics and performance goals, and propose *a cooperative resource provisioning solution* to decreasing the peak resource consumption of workloads on data centers. Our solution coordinates the resource provider's resource provisioning actions for service providers running heterogeneous workloads under competitive conditions, and hence we can lower the server cost. On a basis of our previous work [20] [21], we design and implement an innovative system *PhoenixCloud* that enables cooperative resource provisioning for four typical heterogeneous workloads: parallel batch jobs, Web servers, search engines, and MapReduce jobs. The contributions of our paper are threefold:

First, leveraging differences of heterogeneous workloads, we propose a cooperative resource provisioning solution, and take advantage of statistical multiplexing at a higher granularity—a group of two heterogeneous workloads to save the server cost.

Second, we build an innovative system *PhoenixCloud* to enable cooperative resource provisioning for four representative heterogeneous workloads: parallel batch jobs, Web servers, search engines, and MapReduce jobs.

Third, we perform a comprehensive evaluation of PhoenixCloud and the non-cooperative EC2+RightScale-like systems.

The rest of the paper is organized as follows. Section 2 formulates the problem. Section 3 presents the cooperative resource provisioning solution, followed by the description of our cooperative resource provisioning solution in Section 4. A comprehensive evaluation and comparison of the systems are depicted in Section 5. Finally, related work and concluding remarks are listed in Section 6 and 7, respectively.

2 Problem Formulation and Motivation

According to [29] [32] [46], the server cost contributes the largest proportion of the total cost of data centers. From the cost model analysis [29] [32] [46], we can observe that the system capacity is a key factor in lowering data center costs: on one hand, the highest fraction of the cost—the server cost (53%) [29] is decided by *the system capacity in terms of servers*; on the other hand, the second highest fraction of data center cost—the power & cooling infrastructure cost depends on the maximum design (rated) power consumption that is decided by the system capacity in addition to the degree of fault tolerance and functionality desired of data centers [46].

Since the system capacity at least must be greater than *the peak resource consumption* of workloads, and hence we must decrease the peak resource demands of workloads so as to lower the server cost and the power & cooling infrastructure cost. At the same time, a resource provider (in short, *RP*) also concerns the total resource consumption of workloads. Since each event of requesting, releasing or provisioning resources will trigger a setup action, for example wiping off the operating system or data, elastic resource provisioning in EC2-like systems will result in the management overhead, which can be measured in terms of *(the accumulated counts of adjusting resources)* $\times$ *(the average setup duration)*. So for a RP, the total resource consumption is the sum of *the effective resource consumption*, directly consumed by workloads, and the management overhead.

We gives the accurate definitions of the peak resource consumption and the effective resource consumption as follows:

For a service provider (in short, *SP*), the workloads are $w_{i,i=1...N}$. For w_i, the whole running duration is T_i. At the j_{th} time unit of leasing resources (Δt), $j = 1...M$, the size of the resources provisioned to w_i is $c_{i,j}$ servers, and the peak resource consumption is $\max(\sum_{i=1}^{N} c_{i,j,j=1...M})$, while the effective resource consumption is $\sum_{i=1}^{N} \sum_{j=1}^{\lceil \frac{T_i}{\Delta t} \rceil} c_{i,j} \Delta t$.

Lowering the server cost can not sacrifice performance concerns of both SPs and end users. For workloads, when a user leases resources, resources are charged at a granularity of *a time unit of leasing resources*—Δt. For example, in EC2, the time unit of leasing resources is one hour. So we can use *the effective resource consumption of workloads* that is *the sizes of leased resources times their respective leasing terms* to indirectly reflect the SP cost. The other performance metrics are closely related with different workloads. For parallel batch jobs or MapReduce jobs, the major concern of an SP is *the throughput in terms of the number of completed jobs* [3] [8]; while the main concern of end users is *the average turnaround time per job, which is the average time from submitting jobs till completing them [8] [10]*. For Web servers and search engines, the major concern of an SP is *the throughput in terms of requests per second* [5] [7], while the quality of service in terms of *the average response time per request* is the major concern of end users [5] [7].

In this context, this paper focuses on *how to lower the server cost while not severely degrading the performance metrics of SPs and end users*. As explained above, the minimum system capacity allowed in terms of the number of nodes at least must be greater than *the peak resource consumption* of workloads, so we measure *the minimum server cost* in terms of the peak resource demands of workloads, and our resource provisioning solution focuses on how to decrease the peak resource demands while not severely degrading the performance metrics of SPs and end users.

2.1 Why previous work fails to propose a cooperative solution?

First, in the past, users choose dedicated systems for parallel batch jobs or Web services [20]. The idea that interactive/web workloads and batch workloads should be scheduled differently is classics. Moreover, the trace data of *individual* HPC or Web service workloads are publicly available in http://www.cs.huji.ac.il/labs/parallel/workload/ or http://ita.ee.lbl.gov/html/traces.html, respectively. In this context, most of researchers focus on how to optimize for homogeneous workloads.

Second, cloud is an emerging platform in recent years. Only after Amazon' EC2 becomes a public utility, people begin to realize that more and more heterogeneous workloads appeared on the same platform.

Third, the trace data of consolidating heterogeneous workloads *on the same platform* are not publicly available, and hence researchers have no publicly available benchmarks and traces to evaluation their solutions, which prevents them from deep investigations into the issue of cooperative resources provisioning for heterogeneous workloads. Recently, we release a benchmark suite for cloud computing [61].

Lastly, EC2 uses a statistical multiplexing technique, which indeed leverages the differences of heterogenous workloads. However, our work takes a further action and takes advantage of statistical multiplexing at a higher granularity—a group of two heterogeneous workloads to save the server cost.

3 The Cooperative Resource Provisioning Model

As more and more computing moves to data centers, a RP needs to provision resources for *increasing* heterogeneous workloads. Different from the server sprawl caused by isolating applications or operating system heterogeneity [50] (server consolidation), increasing heterogeneous workloads in terms of both types and intensities raise new challenges in the system capacity planning, since they have significantly different resource consumption characteristics.

In this paper, for four typical workloads: batch jobs, Web servers, search engines, and MapReduce jobs, we leverage the workload differences to save the server cost in two ways.

First, their resource demands in terms of usage mode, timing, intensity, size and duration are significantly different. Web server workloads are often composed of a series of requests with short durations like seconds; the ratios of peak loads to normal loads are high; requests can be serviced simultaneously and interleavedly through resource multiplexing. Batch job workloads are composed of a series of submitted jobs with varying resource demand sizes, and each job is a parallel or serial application, whose runtime is varying but much longer, e.g., hours; running a parallel application needs a group of exclusive resources. Though MapReduce jobs share several similarities with batch jobs written in MPI, they have two distinguished differences: (a) MapReduce jobs are often composed of different jobs with various sizes of data inputs, which decide scales of their resource demands. (b) In running, MapReduce tasks are independent of each other so killing one task will not impact another [28]. This independence between tasks is in contrast to batch jobs of programming models like MPI in which tasks execute concurrently and communicate during their execution [28].

Second, their performance goals are different, and we can coordinate resource provisioning actions of service providers running heterogeneous workloads to decrease peak resource consumption. In general, from perspectives of end users, submitted jobs can be queued when resources are not available. However, for Web services like Web servers or search engines, each individual

request needs an immediate response. So when urgent resources are needed by web service spikes, we can firstly satisfy the resource requests of Web services while delaying execution of parallel batch jobs or MapReduce jobs.

For cooperative resource provisioning, we propose two guiding principles as follows: first, if an SP does not allow cooperative resource provisioning, e.g., for security or privacy, or can not find a *coordinated SP*, the RP will independently provision resources for its loads. We call two SPs adopting the cooperative resource provisioning solution *two coordinated SPs*. Second, if allowed by SPs, the RP supports cooperative resource provisioning at *a granularity of a group, which consists of two heterogeneous workloads*.

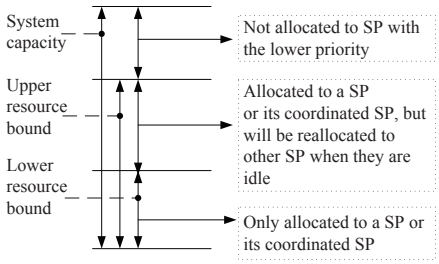


Fig. 1: The system capacity and two resource bounds.

For each group of heterogeneous workloads, we propose the following cooperative resource provisioning model.

First, when an SP resorts to a hosting data center, it needs to specify two resource bounds: *the lower resource bound* and *the upper resource bound*. The RP will guarantee that the resources within the lower resource bound can only be allocated to the specified SP or its coordinated SP. Regarding the resources between the lower and the upper resource bounds, the RP firstly satisfies resource requests of the specified SP or its coordinated SP, however the resources between two bounds can be reallocated to another SP when they are idle. Fig.1 shows the relationship among the system capacity and two resource bounds.

Second, the RP decides whether a group of two SP candidates can join or not according to the following strategy.

The RP will set and maintain *a resource booking threshold—RBT that determines whether two SP candidates are allowed to join or not*. Through setting a reasonable value of *RBT*, the RP guarantees that the sum of the upper bounds of all SPs joined in the system and the two SPs to be joined (candidates) is lower than the system capacity times *RBT*. When *RBT* is larger than one, it means resources are overbooked. In addition, the sum of the lower bounds of the two SP candidates must be lower than the size of the current idle resources in the system. Only when the two requirements are met at the same time, the two SP candidates could join in the system.

Third, if a group is allowed to join, the RP allocates resources within the lower resource bounds to the group, which consists of the two SPs running heterogeneous workloads, at their startup, respectively.

Fourth, for each group of two heterogenous workloads, one heterogenous workload whose resource requests need an immediate response has a higher priority than another one whose resource requests can be queued. For example, for two typical heterogenous workloads: Web servers and parallel batch jobs, Web servers have a higher priority than batch jobs because submitted jobs can be queued when resources are not available, while for Web services each individual request needs an immediate response.

We propose the following runtime algorithm as shown in Algorithm 1 of Appendix A.

1) If it is an SP with a higher priority to request resources, the RP will allocate resources with the requested size immediately. However, if it is an SP with a lower priority to request resources, the extra conditions need to be checked as follows. Since the resources are not unlimited, the RP sets *a usage rate reference —URR* and *a proportional sharing factor—PSF* to coordinate resource sharing under competitive conditions. For an SP with a lower priority, if the size of its allocated resources plus its requested resources is over its upper bound, no resources will be allocated. Otherwise the SP will be granted with only a part of the requested resources, which is decided by the RP through comparing *the usage rate that is the ratio of the resources allocated to all SPs to the system capacity*, with URR.
2) If the usage rate is larger than *URR*, the RP will grant resources to the SP with the size defined as follows:
   AllocatedSize = min{the size of the requested resources, the size of the unallocated resources} $*PSF$.
   We define PSF as the ratio of LB of the SP with a lower priority requesting for resources over the sum of LBs of all joined SPs with lower priorities.
3) When the usage rate is not larger than *URR*, if the size of the requested resources is less than the size of the unallocated resources, the RP will grant resources to the SP with the requested size, otherwise the RP will grant resources to the SP with the size defined as follows:
   AllocatedSize = (the size of the unallocated resources) $*PSF$.

Fifth, it is the SP that proactively decides to request or release resources according to their own resource demands.

Lastly, if an SP does not allow cooperative resource provisioning or can not find a coordinated SP, we can treat the coordinated SP as a null and simplify the above model as an independent resource provisioning solution for either a workload with a higher priority or a lower one.

When we presume the RP has unlimited resources, *both the resource booking threshold and the usage rate reference can be set as one*. This case indicates that the resources are enough with respect to SPs' resources requests, so resources overbooking is not necessary and resources demanded in each request $\ll$ the system capacity.

4 Design and Implementation of PhoenixCloud

In this section, we give out the design and implementation of PhoenixCloud, which is on the basis of our previous system—-DawningCloud [20] [21]. Different from DawningCloud, the unique difference of PhoenixCloud is that it supports *cooperative resource provisioning* for heterogeneous workloads through creating coordinated *runtime environments that are responsible for cooperatively managing resources and workloads.*

Presently, PhoenixCloud only supports cooperative resource provisioning between Web servers, parallel batch jobs, search engine, and MapReduce jobs. However, it can be extended for other data-intensive programming models, e.g., Dryad-like data flow and All-Pairs, which are supported by our previous Transformer system [33].

PhoenixCloud follows a two-layered architecture: one is *the common service framework* (in short CSF) for a RP, and another is *thin runtime environment software* (in short, TRE) for an SP, which is responsible for managing resources and workloads for each SP. The two-layered architecture indicates that there lies a separation between the CSF and a TRE: the CSF is provided and managed by a RP, independent of any TRE; with the support of the CSF, a TRE or two coordinated TREs can be created on demand. The concept of TRE implies that for heterogeneous workloads, the common sets of functions of runtime environments are delegated to the CSF, while a TRE only implements the core functions for a specific workload.

Fig. 2 describe the system architecture view of PhoenixCloud, of which a TRE for parallel batch jobs or MapReduce jobs (in short PBJ TRE) and a TRE for Web servers (in short WS TRE) reuse the CSF. Hereby, we just simply gives out the major components of CSF and TRE, and the detail of the other components can be found at our publicly available technical report [35].

Among the CSF, *the resource provision service* is responsible for coordinating resources provisioning; there are two types of monitors: *the resource monitor* and *the application monitor*. The resource monitor on each node monitors usages of physical resources, e.g. CPU, memory, swap, disk I/O and network I/O. The application monitor detects application status.

There are three components in a TRE: *the manager*, *the scheduler*, and *the Web portal*. The manager is responsible for dealing with user requests, managing resources, and interacting with the CSF. The scheduler is responsible for scheduling jobs or distributing requests. The Web portal is a graphical user interface, through which end users

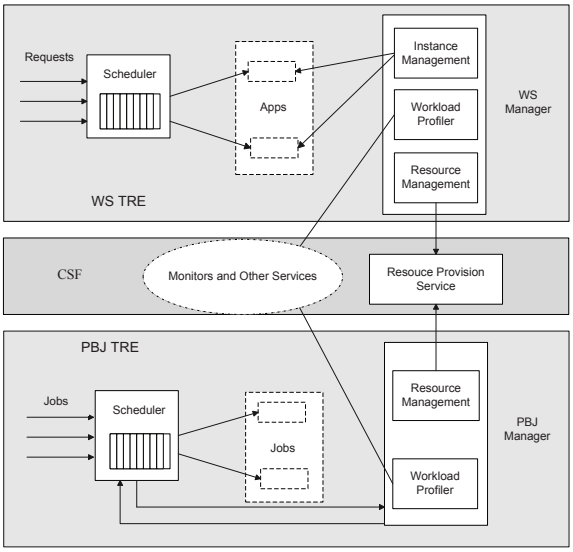


Fig. 2: Interactions of a PBJ TRE and a WS TRE with the CSF.

submit and monitor jobs or applications. When a TRE is created, a configuration file is used to describe their dependencies.

Fig.2 shows the interactions among two TREs and the CSF. The coordination of two service providers running heterogeneous workloads and the RP is as follows:

Specified for the resource provision service, *a resource provision policy* determines when the resource provision service provisions how many resources to a TRE or how to coordinate resources between two coordinated runtime environments. In Section 3, we introduce our resource provision policy.

Specified for the manager—the management entity of each TRE, *a resource management policy* determines when the manager requests or releases how many resources from or to the resource provision service according to what policy.

For different workloads, *the scheduling policy* has different implications. For parallel batch jobs or MapReduce jobs, the scheduling policy determines when and how the scheduler chooses jobs or tasks (which constitute a job) for running. For Web servers, the scheduling policy includes two specific policies: *the instance adjustment policy* and *the request distribution policy*. The instance adjustment policy decides when the number of Web server instances is adjusted to what an extent, and the request distribution policy decides how to distribute requests according to what criteria.

As an example, the interaction of a WS TRE and a PBJ TRE with the CSF is as follows, respectively:

1) The WS manager obtains initial resources within the lower resource bound from the resource provision service, and runs Web server instances with a matching scale.
2) The WS manager interacts with *the load balancer—a type of the scheduler* that is responsible for as-

signing workloads to Web server instances to set its request distribution policy. The WS manager registers the IP and port information of Web server instances to the load balancer, and *the load balancer* distributes requests to Web server instances according to the request distribution policy. We integrate LVS (http://www.linuxvirtualserver.org/) as the IP-level load balancer.

3) The monitor on each node periodically checks resources utilization rates and reports them to the WS manager. If the threshold performance value is exceeded, e.g., the average of utilization rates of CPUs consumed by instances exceeds 80 percent, the WS manager adjusts the number of Web server instances according to the instance adjustment policy.
4) According to the current Web server instances, the WS manager requests or releases resources from or to the resource provision service.

For parallel batch jobs or MapReduce jobs, PhoenixCloud adopts the resource management policy as follows:

There is *a time unit of leasing resources, which is represented as* L. We presume that *the lease term of a resource is a time unit of leasing resource times a positive integer*. In EC2-like systems, for parallel batch jobs, each end user is responsible for manually managing resources, and we presume that a user only releases resources at the end of each time unit of leasing resources if a job runs over. This is because: first, EC2-like systems charge the resource usage in terms of a time unit of leasing resources (an hour); second, it is difficult for end users to predict the completed time of jobs, and hence releasing resources to RP on time is almost impossible.

We define *the ratio of adjusting resource* as *the ratio of the accumulated resource demands of all jobs in queue* to *the current resources owned by a TRE*. When the ratio of adjusting resource is greater than one, it indicates that for immediate running, some jobs in the queue need more resources than that currently owned by a TRE.

We set two threshold values of adjusting resources, and call them *the threshold ratio of requesting resource*, which is represented as U, and *the threshold ratio of releasing resource*, which is represented as V, respectively. The processes of requesting and releasing resources are as follows:

First, the PBJ manager registers a periodical timer (a time unit of leasing resources) for adjusting resources per time unit of leasing resources. Driven by the periodical timer, the PBJ manager scans jobs in queue.

Second, if the ratio of adjusting resources exceeds the threshold ratio of requesting resource, the PBJ manager will request resources with the size of $DR1$ as follows:

for parallel batch jobs:

$DR1$=((the accumulated resources needed by all jobs in the queue)$-$(the current resources owned by a PBJ TRE))$/2$)

for MapReduce jobs:

$DR1$=((the resources needed by the present biggest job in queue))

Third, for parallel batch jobs if the ratio of adjusting resource does not exceed the threshold ratio of requesting resources, but *the ratio of the resource demand of the present biggest job in queue* to *the current resources owned by a TRE* is greater than one, the PBJ manager will request resources with the size of $DR2$:

$DR2$=(the resources needed by the present biggest job in queue)-(the current idle resources owned by a TRE)

For parallel batch jobs when the ratio of *the resource demand of the present biggest job in the queue* to *the current resources owned by a TRE* is greater than one, it implies that the largest job will not run without available resources. Please note that significantly different from parallel batch jobs, MapReduce tasks (that constitutes a job) are independent of each other [28], so the case mentioned above will not happen.

Fourth, if the ratio of adjusting resources is lower than the threshold ratio of releasing resources, the PBJ manager will releases idle resources with the size of RSS (ReleaSing Size).

RSS=(idle resources owned by the PBJ TRE)/2.

Fifth, if the resource provision service proactively provisions resources to the PBJ manager, the latter will receive resources.

Zhang *et al.* [22] argue that in managing web services of data centers, actual experiments are cheaper, simpler, and more accurate than *models* for many management tasks. We also hold the same position. In this paper, we deploy the real systems to decide the resource management policy for two Web servers and one search engine workload traces.

5 PERFORMANCE EVALUATIONS

In this section, for four typical workloads: Web servers, parallel batch jobs, search engine, and MapReduce jobs, we compare the performance of the non-cooperative system: EC2+RightScale-like systems and PhoenixCloud for the same workload traces: first, on a hosting data center, the RP deploys an EC2-like system for a large amount of end users that submit parallel batch jobs or MapReduce jobs, and RightScale for service providers running Web server or search engine; second, the RP deploys PhoneixCloud, providing a cooperative resource provisioning solution for heterogeneous workloads. We choose EC2+RightScale-like systems because EC2 leverages the statistical multiplexing technique to decrease server sprawl while RightScale roughly implements the elastic resource provisioning technique proposed in state-of-the-art work [47] [48] [49] [50].

5.1 Real workload traces

For parallel batch jobs, we choose three typical workload traces: NASA iPSC, SDSC BLUE and LLNL Thunder from http://www.cs.huji.ac.il/labs/parallel/workload/. NASA iPSC is a real trace segment of two

weeks from Oct 01 00:00:03 PDT 1993. For the NASA iPSC trace, the configuration of the cluster system is 128 nodes. SDSC BLUE is a real trace segment of two weeks from Apr 25 15:00:03 PDT 2000. For the SDSC BLUE trace, the cluster configuration is 144 nodes. LLNL Thunder is a real trace segment of two weeks from Feb 1 18:10:25 PDT 2007. For the LLNL Thunder trace, the configuration of the cluster system is 1002 nodes (excluding the management nodes and so on).

For Web servers, we use two real workload traces from http://ita.ee.lbl.gov/html/traces.html: one is the World Cup workload trace [2] from June 7 to June 20 in 1998, and the other is the HTTP workload trace from a busy Internet service provider—ClarkNet from August 28 to September 10 in 1995. Meanwhile, we choose a publicly available anonymous search engine workload trace from March 28 00:00:00 to 23:59:59 in 2011 [56], which we call SEARCH in the rest of this paper.

5.2 Synthetic Workload traces

To the best of our knowledge, the real traces of parallel batch jobs, Web servers, search engines, and MapReduce jobs on the same platform are not available. So in our experiments, on a basis of the real workload traces introduced in Section 5.1, we create synthetic workload traces . We propose a tuple of (PRC_A, PRC_B) to represent two heterogeneous workload traces: A that has a lower priority, e.g., parallel batch jobs or MapReduce jobs, and B that has a higher priority, e.g., Web servers or search engines; PRC_A is the maximum resource demand of the largest job in A, and PRC_B is the peak resource consumption of B. For example, a tuple of $(100, 60)$ that is scaled on a basis of the SDSC BLUE and World Cup traces has two-fold implications: (a) we scale the SDSC BLUE and World Cup traces with two different constant factors, respectively; (b) on the same simulated cluster system, the maximum resource demand of the largest job in SDSC BLUE and the peak resource consumption of World Cup is 100 and 60 nodes, respectively.

5.3 Experiment methods

To evaluate and compare the PhoenixCloud and EC2+RightScale-like systems, we adopt the following experiments methods.

5.3.1 The experiments of deploying real systems

For Web servers and search engines, we obtain *resource consumption traces* through deploying and running the real systems for the three real workload traces in Section 5.1. For MapReduce jobs, we also perform experiments through deploying and running real systems on physical hardware as shown in Section 5.5.2.

5.3.2 The simulated experiments for heterogeneous workloads

The period of a typical workload trace is often weeks, or even months. To evaluate a system, many key factors have effects on experiment results, and we need to frequently perform time-consuming experiments. Moreover, when we perform experiments for several hundreds of workload traces, their resource consumption is up to more than ten thousand nodes, which are not affordable. So we use the simulation method to speedup experiments. We speed up the submission and completion of jobs by a factor of 100. This speedup allows two weeks' trace to complete in about three hours. In addition, in Section 5.7, we deploy *the real systems on physical hardware* to validate the accuracies of our simulated systems.

5.4 The testbed

Shown in Fig.3, the testbed includes three types of nodes, nodes with the name starting with *glnode*, nodes with the name starting with *ganode*, and nodes with the name starting with *gdnode*. The nodes of *glnode* have the same configuration, and each node has 2 GB memory and two quad-core Intel(R) Xeon(R) (2.00 GHz) processors. The OS is a 64-bit Linux with the kernel of 2.6.18-xen. The nodes of ganode have the same configuration, and each node has 1 GB memory and 2 AMD Optero242 (1.6 GHz) processors. The OS is an 64-bit Linux with the kernel version of 2.6.5-7.97-smp. The nodes of gdnode have the same configuration, and each node has 4 GB memory and one quad-core Intel(R) Xeon(R) (1.60 GHz) processor. The OS is an 64-bit Linux with the kernel version of 2.6.18-194.8.1.el5. All nodes are connected with a 1 Gb/s switch.

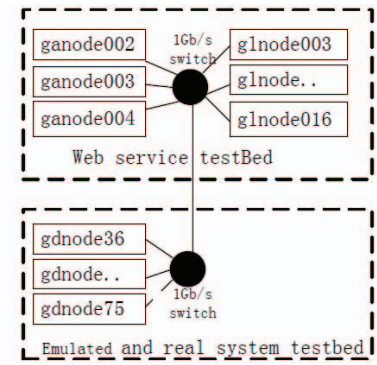


Fig. 3: The testbed.

5.5 The experiments of deploying real systems

5.5.1 Running real web servers and search engines systems

In this section, we deploy the real systems to obtain the resource consumption traces for two Web servers and one search engine workload traces.

We deploy eight XEN virtual machines (http://www.xensource.com/) on each node of starting with *glnode*. For each XEN virtual machine, one core and 256 MB memory is allocated, and the guest operating system

is a 64-bit CentOS with the kernel version of 2.6.18-XEN. we deploy the real PhoenixCloud system: the load balancer is LVS with the direct route mode http://kb.linuxvirtualserver.org/wiki/LVS/DR; each agent and each monitor is deployed on each virtual machine, respectively; LVS and the other services are deployed on *ganode004*, since all of them have light loads.

We choose the least-connection scheduling policy (http://kb.linuxvirtualserver.org/wiki/Least-Connection_Scheduling) to distribute requests. We choose *httperf* (http://www.hpl.hp.com/research/linux/httperf/) as the load generator and an open source application—*ZAP!* (http://www.indexdata.dk/) as the target Web server. The versions of httperf, LVS and ZAP! are $0.9.0$, 1.24 and $1.4.5$, respectively. Two httperf instances are deployed on *ganode002* and *ganode003*.

The Web server workload trace is obtained from the World Cup workload trace [2] with a scaling factor of 2.22. The experiments include two steps. First, we decide the instance adjustment policy; secondly, we obtain the resource consumption trace.

In the first step, we deploy PhoenixCloud with the instance adjustment policy disabled. For this configuration, the WS manager will not adjust the number of the Web service instances. On the testbed of 16 virtual machines, 16 ZAP! instances are deployed with each instance deployed on each virtual machine. When httperf generates different scales of loads, we record the actual throughput, the average response time, and the average utilization rate of CPU cores. Since we guarantee that one CPU core is allocated to one virtual machine, for virtual machine, the number of VCPUs is number of CPU cores. So the average utilization rate of each CPU core is also the average utilization rate of VCPUs. We observed that when the average utilization rate of VCPUs is below 80 percent, the average response time of requests is less than 50 milliseconds. However, when the average utilization rate of VCPUs increases to 97%, the average response time of requests dramatically increase to 1528 milliseconds. Based on the above observation, we choose the average utilization rate of VCPUs as the criterion for adjusting the number of instances of ZAP!, and set 80 percent as the threshold value. For ZAP!, we specify the instance adjustment policy as follows: the initial service instances are two. If the average utilization rate of VCPUs consumed by all instances of Web service exceeds 80% in the past 20 $seconds$, the WS manager will add one instance. If the average utilization rate of VCPUs, consumed by the current instances of Web service, is lower than $(0.80(\frac{n-1}{n}))$ in the past 20 $seconds$ where n is the number of current instances, the WS manager will decrease one instance.

In the second step, we deploy PhoenixCloud with the above instance adjustment policy enabled. The WS manager adjusts the number of Web service instances according to the instance adjustment policy. In the experiments, we also record the relationship between the actual throughput, the average response time, and the number of virtual machine. We observe that for different number of VMs, the average response time is below 700 milliseconds and the throughput increases linearly when the number of VM increases. This indicates that the instance adjust policy is appropriate, may not optimal.

With the above policies, we obtain the resource consumption trace of two weeks for the World Cup workload trace. Using the same approach, we also obtain the resource consumption trace of two weeks for ClarkNet and an anonymous search engine trace—SEARCH [56] with a scaling factor of 772.03 and 171.29, respectively. *The resource consumption traces can be found at Appendix B.1.*

5.5.2 Experiments of running MapReduce jobs on physical hardware

This section evaluates PhoenixCloud for MapReduce jobs through deploying the real systems.

The testbed is composed of 20 X86-64 nodes, starting from *gdnode36* to *gdnode55* as shown in Fig.3. we select a series of widely used MapReduce jobs as our MapReduce workloads, the details of which can be found in Appdendix A.5. Meanwhile, we replay the SEARCH resource consumption trace introduced in Section 5.5.1. We use (PRC_A, PRC_B) to present the peak resource consumption of MapReduce jobs and search engine workloads. In this experiment, (PRC_A, PRC_B) is $(19, 64)$.

We use Hadoop with the version 0.21.0 to run MapReduce jobs, and its JDK version is 1.6.0.20. We configure the namenode and jobtracker on one node while datanodes and tasktrackers on the other nodes. In Hadoop, the block size is 64 MB, and there are 4 map slots and 2 reduce slots on every node, and the scheduling policy is *first come first serve(FCFS)*.

We adopt the resource management policy in Section 4. In PhoenixCloud, for MapReduce jobs, the baseline parameters are $[R0.28/E19/U5/V0.05/L60]$. we also set RBT and URR as one, respectively. In Hadoop, the resources requirement of a MapReduce job is determined by its map task number, and hence the requested resources are the map task number divided the the map slots of each node. So as to prevent the peak resource demand of MapReduce jobs from exceeding the system capacity of the testbed, we divide each resource request by a scaling number. For the current 20-node testbed, the scaling number is 16. For an EC2-like system, we run each MapReduce job one by one according to its timestamp. Finally, we add up the resource consumption of all MapReduce jobs to get its peak and total resource consumption. For the same reason, we also use a scaling factor to decrease the peak resource demand of MapReduce jobs.

From the above results, we can observe that our algorithm works well for MapReduce jobs. With respect to the non-cooperative systems: EC2+RightScale-like systems, PhoenixCloud can significantly decrease

TABLE 1: The RP's metrics for MapReduce jobs and the search engine workload—SEARCH.

System	peak resource consumption	resource consumption $(node * hour)$	saved peak resource consumption	saved resource consumption
Phoenix Cloud	78	960	63.21%	57.33%
Non-cooperative	212	2250	0	0

TABLE 2: The service provider's metrics for MapReduce jobs.

System	number of completed jobs	average turn around time $(seconds)$	resource consumption $(node * hour)$
Phoenix Cloud	477	568	960
Non-cooperative	477	310	2250

the peak resource consumption by 63.21 percent and total resource consumption by 57.33 percent. Meanwhile, the throughput of PhoenixCloud is the same as that of the EC2-like system; the average turnaround time per job is 568, 310 seconds for PhoenixCloud and the EC2-like system, respectively, and the average delay is 258 seconds per job.

5.6 Simulation Experiments

In this section, we compare the performance of EC2+RightScale-like systems and PhoenixCloud for *Web server and parallel batch jobs* workload traces, and the experiment setups are as follows:

5.6.1 The simulated systems

(a) **The simulated clusters**. The workload traces mentioned above are obtained from the platforms with different configurations. For example, NASA iPSC is obtained from the cluster system with each node composed of one processor; SDSC BLUE is obtained from the cluster system with each node composed of eight processors; LLNL Thunder is obtained from the cluster system with each node composed of four processors. In the rest of experiments, *our simulated cluster system is modeled after the NASA iPSC cluster, comprising only single-processor nodes*.

(b) **The simulated EC2+RightScale-like systems**. Based on the framework of PhoenixCloud, we implement and deploy the EC2+rightScale-like systems as shown in Fig.4 on the testbed. The simulated system includes two simulation modules: the job simulator and the resource simulator. With respect to the real PhoenixCloud system, we only keep the resource provision service, the WS manager and the PBJ manager. The job simulator reads the number of nodes which each job requests in the trace file and sends the resource requests to the resource provision service, which assigns corresponding resources for each job. When each job runs over, the job simulator will release resources to the resource provision service. The resource simulator simulates the varying resources consumption and drives the WS manager to request or release resources from or to the resource provision service. Because RightScale provides the same scalable management for Web service as PhoenixCloud, *we just use the same resource consumption traces of Web service in Section 5.5.1 in two systems, respectively*.

(c) **The simulated PhoenixCloud**. With respect to the real PhoenixCloud system in Fig.2, our simulated PhoenixCloud keeps the resource provision service, the PBJ manager, the WS manager, and the scheduler, while other services are removed. The resource provision service enforces the cooperative resource provisioning model defined in Section 3.

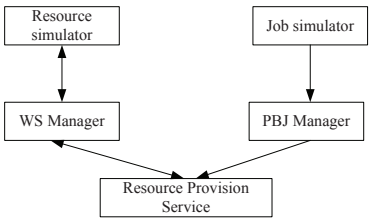


Fig. 4: The simulated system of EC2+RightScale-like systems.

5.6.2 Experiment configurations

(a) **The scheduling policy of parallel batch jobs**. For parallel batch jobs, PhoenixCloud adopts the first-fit scheduling policy. The first-fit scheduling policy scans all the queued jobs in the order of job arrivals and chooses the first job, whose resources requirement can be met by the system, to execute. The EC2-like system needs no scheduling policy, since it is each end user that is responsible for manually requesting or releasing resources for running parallel batch jobs.

(b) **The resource management policy**. For parallel batch jobs, we adopt the resource management policy stated in Section 4.

5.6.3 Experiments with limited system capacities

In Appendix B.2, we present experiments results when we presume the RP has limitless resources. In this section, we report results when we presume the RP has limited resources of different system capacities.

For the six workload traces introduced in Section 5.1, there are three combinations. We use the combinations of heterogeneous workloads:((IPSC, WorldCup), (SDSC, Clark), (LLNL, SEARCH)), and their numbers are $((128, 128), (144, 128), (1002, 896))$, respectively.

Through comparisons with a large amount of experiments, we set the baseline parameters in PhoenixCloud: $[R0.5/E400/U1.2/V0.1/L60]$ for (IPSC,WorldCup) and (SDSC,Clark); and

$[R0.5/E3000/U1.2/V0.1/L60]$ for (LLNL,SEARCH). $[Ri/Ej/Uk/Vl/Lm]$ indicates that *the ratio of the sum of the two lower resource bounds to the sum of* PRC_A *and* PRC_B, which is represented as R, is i; *the sum of the two upper resource bounds, which is represented as* E, *is* j *nodes*; the threshold ratio of requesting resources U is k; the threshold ratio of releasing resources V is l; the time unit of leasing resources L is m minutes. In Appendix B.4, we investigate the effects of different parameters in PhoenixCloud.

Each experiment is performed six times, and we report the mean values across six times experiments. In Table 3, for different system capacities, i.e., 4000, 5000, 6000, in addition to the baseline parameters for SPs, we set the usage rate reference—*URR*, and the resource booking threshold—*RBT* for the RP as 0.5 and 2, respectively. For *E* (the sum of the two upper resource bounds), $[A/B/C]$ means that for (IPSC, WorldCup), the sum of two upper bounds is *A*; for (SDSC, Clark), the sum of two upper bounds is *B*; for(LLNL, SEARCH), the sum of two upper bounds is *C*.

TABLE 3: The configurations v.s. different system capacities.

system capacity $(nodes)$	system	E $[A/B/C]$	URR	RBT
4000	Phoenix Cloud	[400/400/3000]	0.5	2
5000	Phoenix Cloud	[400/400/3000]	0.5	2
6000	Phoenix Cloud	[400/400/3000]	0.5	2

TABLE 4: The RP's metrics of six real workload traces in PhoenixCloud v.s. system capacities.

system capacity $(nodes)$	system	peak resource consumption $(nodes)$	effective resource consumption $(node * hour)$	total resource consumption $(node * hour)$
six traces	Non-cooperative	6126	677190	687441
4000	Phoenix Cloud	3518	545450	551845
5000	Phoenix Cloud	3283	546395	552846
6000	Phoenix Cloud	2899	543569	549832

From Table 4, Table 5, Table 6, Table 7 and Table 8, we can say that our algorithm works quite well on different system capacities:

First, from the perspective of the RP, PhoenixCloud can save a significant amount of peak resource consumption. When the system capacity in PhoenixCloud decreases to 4000 nodes, only 65.3 percent of *the minimum system capacity allowed* in EC2+RightScale-like systems, which is at least largest than the peak resource consumption, PhoenixCloud can save the peak resource consumption and the total resource consumption with respect to EC2+RightScale-like systems maximally by 43 and 20 percents, respectively. The experiment results have twofold reasons: (a) parallel batch jobs can be queued in PhoenixCloud, while the required resources will be provisioned immediately for each batch job in the EC2-like system, hence the resource provider using PhoenixCloud can decrease the peak resource consumption and total resource consumption with respect to EC2+RightScale-like systems; (b) in PhoenixCloud, setting an usage rate reference and a proportional sharing factor can help decrease the peak resource consumption effectively under competitive conditions.

TABLE 5: The service provider' metrics of NASA iPSC v.s. system capacities.

system capacity $(nodes)$	system	number of completed jobs	average turn around time $(seconds)$	resource consumption $(node * hour)$
six traces	Non-cooperative	2603	573	54118
4000	Phoenix Cloud	2603	766	35786
5000	Phoenix Cloud	2603	738	35621
6000	Phoenix Cloud	2603	733	35146

TABLE 6: The service provider' metrics of SDSC Blue v.s. system capacities.

system capacity $(nodes)$	system	number of completed jobs	average turn around time $(seconds)$	resource consumption $(node * hour)$
six traces	Non-cooperative	2657	1975	35838
4000	Phoenix Cloud	2656	2642	31231
5000	Phoenix Cloud	2652	2546	31654
6000	Phoenix Cloud	2654	2535	31573

Second, from the perspectives of the service providers, the throughput of PhoenixCloud in terms of the number of completed jobs has little change with respect to EC2+RightScale-like systems (the maximal decrease is only 0.2 percent); the average turn around time is delayed with small amounts (the maximal delay is 34, 34 and 21 percents for NASA, SDSC and LLNL, respectively). On the other hand, the resource consumption of each SP is significantly saved, and the maximal saving is 35 ,13, 21 percents for NASA, SDSC and LLNL with respect to EC2+RightScale-like systems, respectively. The reason is as follows: parallel batch jobs may be queued in PhoenixCloud, while the required resources will be provisioned immediately for each batch job in the EC2-like system, hence the number of completed jobs and the average turn around time would be affected, but the resource management policy in PhoenixCloud makes these influences slightly and lets the SPs save the resource

TABLE 7: The service provider' metrics of LLNL Thunder v.s. system capacities.

system capacity ($nodes$)	system	number of completed jobs	average turn around time ($seconds$)	resource consumption ($node * hour$)
six traces	Non-cooperative	7273	2465	339416
4000	Phoenix Cloud	7261	2941	270847
5000	Phoenix Cloud	7262	2882	271233
6000	Phoenix Cloud	7262	2978	268298

TABLE 8: With respect to EC2+RightScale-like systems, the peak and total resource consumption saved by PhoenixCloud for different system capacities.

system capacity ($nodes$)	system	saved peak resource consumption	saved total resource consumption
4000	Phoenix Cloud	42.57%	19.72%
5000	Phoenix Cloud	46.41%	19.58%
6000	Phoenix Cloud	52.68%	20.02%

consumption significantly.

5.6.4 Experiments with different amounts of synthetics heterogeneous workloads

This section evaluate the efficiency of our approach under different amounts of synthetics heterogeneous workloads from 6 to 120.

The synthetic workloads are generated based on the ones introduced in Section 5.1, which are (IPSC, WorldCup),(SDSC, Clark), and (LLNL, SEARCH). When we perform experiments for a group of workload traces of an amount of $(6N + 6)$, we generate N new workload combinations as follows:

We record the i_{th} workload combination as $((iPSC_i, WorldCup_i), (SDSC_i, Clark_i), (LLNL_i, SEARCH_i)$,$_{i=1...N}$). We divide each workload trace in ((IPSC,WorldCup),(SDSC,Clark) and (LLNL,SEARCH)) into $(N + 1)$ parts. So as to prevent from the duplication of workload traces in the same period, in the same combination of workload traces, for iPSC, SDSC or LLNL, we swap the first i parts with the rest of the following parts to obtain $((iPSC_i), (SDSC_i), (LLNL_i))$; for Clark, WorldCup or SEARCH, we swap the last i parts with the rest of the preceding parts to obtain $((WorldCup_i), (Clark_i), (SEARCH_i))$.

For parallel batch job workload traces, we take $IPSC_i$ as an example: first, we define $Sub = ((336*3600*i)/(N+1))$ Second, for a submitted job, if its time stamp t is lower than Sub, we reset its submitting time as $(t+(336*3600 - Sub))$, else $t - Sub$.

For Web service workload traces, we take $Clark_i$ as an example. We define $MP = ((336 * i)/(N + 1))$. $RC_{j,j=1...336}$ is the resource consumption of the Clark workload trace at the j_{th} hour. For synthetic workloads $Clark_{i,i=1...N}$, if $j < MP$, $RC_{i,j} = RC_{336-MP+i}$, else $RC_{i,j} = RC_{i-MP}$.

Since we consider a large amount of workload traces, we presume the RP has unlimited resources, and set the resource booking threshold and the usage rate reference as one, respectively (See explanation in Section 3). We set the baseline parameters in PhoenixCloud: $[R0.5/E100000/U1.2/V0.1/L60]$. *E100000* means that each service provider has a large upper bound and will get enough resources. Table 9 summarizes the RP's metrics of EC2+RightScale-like systems and PhoenixCloud running different amounts of workloads.

From Table 9, we can observe that for larger-scale deployments, our solution could save the server cost more aggressively. For small-scale (6126 nodes), medium-scale (28206 nodes), and large-scale deployments (116030 nodes), our solution saves the server cost by 18.01, 57.60, and 65.54 percents with respect to the EC2+RightScale-like solutions, respectively.

Please note that in Table 9, for the six workload traces, the configurations of PhoenixCloud are sightly different from that in Table 8 and hence they have different results. First, in Table 9, we presume that each service provider has a larger upper bound; second, in Table 9, we presume that the resource provider has unlimited resources, and set the resource booking threshold and the usage rate reference as one, respectively.

TABLE 9: With respect to EC2+RightScale-like systems, the peak and total resource consumption saved by PhoenixCloud for different amounts of workloads.

Workload number	peak resource consumption of EC2+RightScale-like systems	saved peak resource consumption	saved total resource consumption
6	6126	18.01%	17.22%
30	28206	57.60%	15.32%
60	56920	61.14%	16.52%
120	116030	65.54%	16.56%

5.7 The accuracies of the simulated systems

In order to verify the accuracies of our simulated system, we deploy the real PhoenixCloud system on the testbed. The testbed is composed of 40 X86-64 nodes, starting from *gdnode36* to *gdnode75* as shown in Fig. 3.

Since we have run the real web service workloads in Section 5.5.1, we only perform the real experiment of running parallel batch workload on physical hardware. We synthesize a parallel batch job workload trace, which includes 100 jobs with the size from 8 to 64 cores. 100 jobs are submitted to PhoenixCloud within 10 hours, and the average interval of submitting jobs is 300 seconds. Our experiments include two steps: first, we submit the synthetic parallel batch workloads to the real PhoenixCloud

system, and then collect the workload trace. Second, after we obtain the workload trace through the real system experiments, we submit the same workload trace to the simulated system again. We compare the metrics of the real and simulated systems to evaluate the accuracy of the simulated system. The baseline parameters in PhoenixCloud are $[R0.5/E40/U1.2/V0.1/L60]$, and we set RBT and URR as one, respectively. Through the experiments, we find that the ratios of the peak resource consumption and the total resource consumption of the real PhoenixCloud system to that of the simulated system are about 1.14, respectively, which validates that our emulated systems are enough accurate.

6 Related Work

In this section, we summarize the related work from four perspectives.

6.1 Approaches and Systems Leveraged to Save Data Center Costs

Virtualization-based consolidation. Vogels *et al.* [50] summarizes both state-of-the-practice and state-of-the-art work on leveraging virtualization-based consolidation solutions to combat server sprawl caused by isolating applications or operating system heterogeneity. Several previous efforts utilize virtual machines to provision elastic resources for either multi-tier services [49] [48] or HPC jobs [20] in hosting data centers. Chase *et al.* [43] propose to provision server resources for co-hosted services in hosting data centers in a way that automatically adapts to offered load so as to improve the energy efficiency of server clusters.

Statistical multiplexing. As pointed by Zaharia *et al.* [27], state-of-the-practice EC2 systems leverage on statistical multiplexing techniques to decrease the server cost (the system capacity). Urgaonkar *et al.* [39] present the overbooking resource approach to maximize the number of applications with light loads that can be housed on a given hardware configuration, however they only validated their results for large amount of *light service loads with fixed intensities*. Based on insights gained from detailed profiling of several applicationsłboth individual and consolidated, Choi *et al.* [53] developed models for predicting average and sustained power consumption of consolidated applications.

6.2 Data Center Software Infrastructure

Two open source projects, OpenNebula (http://www.opennebula.org/) and Haizea (http://haizea.cs.uchicago.edu/), are complementary and can be used to manage virtual infrastructures in private/hybrid clouds [25]. Steinder *et al.* [18] show that a virtual machine allows heterogeneous workloads to be collocated on any server machine. Our previous DawningCloud system [20] [21] aims to provide an enabling platform for answering the question: can scientific communities benefit from the economies of scale. Hindman *et al.* [26] present Mesos, a platform for sharing commodity clusters between multiple diverse cluster computing frameworks, such as Hadoop and MPI. The systems proposed in [5] [18] [25] [20] [26] do not support cooperative resource provisioning for heterogeneous workloads.

6.3 Resource provisioning and Scheduling

Steinder *et al.* [18] only exploits a range of new automation mechanisms that will benefit a system with a homogeneous, particularly non-interactive workload by allowing more effective scheduling of jobs. Silberstein *et al.* [16] devise a scheduling algorithm for massively parallel tasks with different resource demands.

Lin *et al.* [12] provide an OS-level scheduling mechanism, *VSched*. VSched enforces compute rate and interactivity goals for interactive workloads, and provides soft real-time guarantees for VMs hosted *on a single server machine*. Margo *et al.* [13] are interested in metascheduling capabilities (co-scheduling for Grid applications) in the TeraGrid system, including user-settable reservations among distributed cluster sites. Sotomayor *et al.* [17] present the design of a lease management architecture that only consider homogeneous workloads—parallel batch jobs mixed with best-effort lease requests and advanced reservation requests.

Isard *et al.* [28] and Zaharia *et al.* [27] focus on resolving the conflicts between the scheduling fairness and the data locality for MapReduce likes data-intensive jobs on low-end system with directly attached storages. Benoit *et al.* [54] dealt with the problem of scheduling multiple applications, made of collections of independent and identical tasks, on a heterogeneous master-worker platform. Sadhasivam *et al.* [57] enhanced Hadoops fair scheduler that queues the jobs for execution in a fine grained manner using task scheduling.

In non-cooperative themes, Waldspurger *et al.* [59] present lottery scheduling, a novel randomized resource allocation mechanism, which provides efficient, responsive control over the relative execution rates of computations; Jin *et al.* [60] propose an approach to proportional share resource control in shared services by interposed request scheduling.

6.4 Datacenter and cloud benchmarks

Benchmarking is the foundation of evaluating computer systems. Zhan *et al.* [41] systematically identifies three categories of throughput oriented workloads in data centers, whose targets are to increase the volume of throughput in terms of processed requests or data, or supported maximum number of simultaneous subscribers, respectively, and coin a new term *high volume throughput computing* (in short *HVC*) to describe those workloads and data center systems designed for them. Luo *et al.* [61] propose a new benchmark suite, CloudRank, to benchmark and rank private cloud systems that are

shared for running big data applications. Jia *et al.* [40] propose twenty-one representative applications in different domains as a benchmark suite—*BigDataBench* for big data applications.

7 CONCLUSIONS

In this paper, we leveraged the differences of heterogeneous workloads and proposed a cooperative resource provisioning solution to save the server cost, which is the largest share of hosting data center costs. To that end, we built an innovative system *PhoenixCloud* to enable cooperative resource provisioning for four representative heterogeneous workloads: parallel batch jobs, Web server, search engine, and MapReduce jobs. We proposed an innovative experimental methodology that combines real and emulated system experiments, and performed a comprehensive evaluation. Our experiments show taking advantage of statistical multiplexing at a higher granularity—a group of two heterogeneous workloads gains more benefit with respect to the non-cooperative EC2+RightScale-like systems, which simply leverage statical multiplexing technical; for small-scale (6126 nodes), medium-scale (28206 nodes), and large-scale deployments (116030 nodes), our solution saves the minimum server cost allowed (at least larger than the peak resource consumption) by 18, 58, and 66 percents with respect to that of the non-cooperative systems, respectively.

ACKNOWLEDGMENT

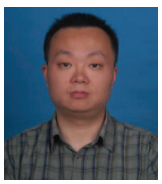
We are very grateful to anonymous reviewers. This work is supported by the Chinese 973 project (Grant No.2011CB302500) and the NSFC project (Grant No.60933003).

REFERENCES

[1] K. Appleby *et al.* 2001. Océano–SLA Based Management of a Computing Utility. In Proc. of IM 2001, pp. 855-868.
[2] M. Arlitt *et al.* 1999. Workload Characterization of the 1998 World Cup Web Site, Hewlett-Packard Company, 1999
[3] A. AuYoung *et al.* 2006. Service contracts and aggregate utility functions. In Proc.of 15th HPDC, pp.119-131.
[4] A. Bavier *et al.* 2004. Operating system support for planetary-scale network services. In Proceedings of NSDI 04, pp. 19-19.
[5] J. S. Chase *et al.* 2001. Managing energy and server resources in hosting centers. In Proc. of SOSP '01, pp.103-116.
[6] J. S. Chase *et al.* 2003. Dynamic Virtual Clusters in a Grid Site Manager. In Proc. of the 12th HPDC, pp.90-103.
[7] A. Fox *et al.* 1997. Cluster-based scalable network services. SIGOPS Oper. Syst. Rev. 31, 5 (Dec. 1997), pp. 78-91.
[8] K. Gaj *et al.* 2002. Performance Evaluation of Selected Job Management Systems. In Proc. of 16th IPDPS, pp.260-260.
[9] L. Grit *et al.* 2008. Weighted fair sharing for dynamic virtual clusters. In Proceedings of SIGMETRICS 2008, pp. 461-462.
[10] K. Hwang *et al.* Scalable Parallel Computing: Technology, Architecture, Programming, and McGraw-Hill 1998.
[11] D. Irwin *et al.* 2006. Sharing networked resources with brokered leases. In Proc. of USENIX '06, pp.18-18.
[12] B. Lin *et al.* 2005. Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling. In Proc. of SC 05, pp.8-8.
[13] M. W. Margo *et al.* 2007. Impact of Reservations on Production Job Scheduling. In Proc. of JSSPP 07, pp.116-131.
[14] B. Rochwerger *et al.* 2009. The Reservoir model and architecture for open federated cloud computing IBM J. Res. Dev., Vol. 53, No.4, 2009.
[15] P. Ruth et al. *2005.* VioCluster: Virtualization for dynamic computational domains. In Proceedings of Cluster 05, pp. 1-10.
[16] M. Silberstein *et al.* 2006. Scheduling Mixed Workloads in Multigrids: The Grid Execution Hierarchy. In Proc. of 15th HPDC, pp. 291-302.
[17] B. Sotomayor *et al.* 2008. Combining Batch Execution and Leasing Using Virtual Machines. In Proc. of HPDC 2008, pp.87-96.
[18] M. Steinder *et al.* 2008. Server virtualization in autonomic management of heterogeneous workloads. SIGOPS Oper. Syst. Rev. 42, 1 (Jan. 2008), pp.94-95.
[19] J. Zhan *et al.* 2005. Fire Phoenix Cluster Operating System Kernel and its Evaluation. In Proc. of Cluster 05, pp.1-9.
[20] L. Wang *et al.* 2009. In cloud, do MTC or HTC service providers benefit from the economies of scale?. In Proc. of SC-MTAGS '09. ACM, New York, NY, 1-10.
[21] L. Wang *et al.* 2011. In Cloud, Can Scientific Communities Benefit from the Economies of Scale?. IEEE TPDS. vol.23, no.2, pp.296-303, Feb. 2012
[22] W. Zheng *et al.* 2009. JustRunIt: Experiment-based management of virtualized data centers, in Proc. of USENIX 09.
[23] Z. Zhang *et al.* 2006. Easy and reliable cluster management: the self-management experience of Fire Phoenix, In Proc. of IPDPS 2006.
[24] J. Zhan *et al.* 2008. Phoenix Cloud: Consolidating Different Computing Loads on Shared Cluster System for Large Organization. In Proc. of CCA' 08. http://arxiv.org/abs/0906.1346.
[25] B. Sotomayor *et al.* 2009. Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing, pp.14-22, 2009.
[26] B. Hindman *et al.* 2010. Mesos: A platform for fine-grained resource sharing in the data center, In Proc. NSDI 2011.
[27] M. Zaharia *et al.* 2010. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In Proc. of EuroSys '10. ACM, New York, NY, 265-278.
[28] M. Isard *et al.* 2009. Quincy: fair scheduling for distributed computing clusters. In Proc. of SOSP '09. ACM, New York, NY, 261-276.
[29] J. Hamilton. 2009. Internet-scale service infrastructure efficiency. SIGARCH Comput. Archit. News 37, 3 (Jun. 2009), 232-232.
[30] D. Thain *et al.* Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, 17(2):323-356, February 2005.
[31] U. Hoelzle *et al.* 2009. The Datacenter as a Computer: an Introduction to the Design of Warehouse-Scale Machines. 1st. Morgan and Claypool Publishers.
[32] P. Wang *et al.* 2010. Transformer: A New Paradigm for Building Data-Parallel Programming Models. IEEE Micro 30, 4 (Jul. 2010), 55-64.
[33] L. Rodero-Merino *et al.* 2010. From infrastructure delivery to service management in clouds. Future Gener. Comput. Syst. 26, 8 (Oct. 2010), 1226-1240.
[34] J. Zhan *et al.* 2010. PhoenixCloud: Provisioning Runtime Environments for Heterogeneous Cloud Workloads. Technical Report. http://arxiv.org/abs/1003.0958v1.
[35] S. Govindan, et al. 2009. Statistical profiling-based techniques for effective power provisioning in data centers. In Proc. of EuroSys '09. ACM, New York, NY, 317-330.
[36] P. Padala *et al.* Automated control of multiple virtualized resources. In Proc. EuroSys '09.
[37] B. Urgaonkar *et al.* 2002. Resource overbooking and application profiling in shared hosting platforms, in Proc. OSDI'02, Boston, MA.
[38] Z. Jia *et al.* BigDataBench: a Benchmark Suite for Big Data Applications in Data Centers. ICT Technical Report.
[39] J. Zhan *et al.* High Volume Throughput Computing: Identifying and Characterizing Throughput Oriented Workloads in Data Centers. In Proc. LSPP 2012 in conjunction with IPDSP 2012.
[40] A. Verma *et al.* 2008. Power-aware dynamic placement of HPC applications. In Proc. ICS '08.
[41] J. Chase *et al.* Managing energy and server resources in hosting centers. In Proc. of SOSP'01.
[42] C. D Patel *et al.* Cost Model for Planning, Development and Operation of a Data Center, Hewlett-Packard Laboratories Technical Report, HPL-2005-107(R.1), June 9, 2005

[43] J. Moreira *et al.* The Case for Full-Throttle Computing: An Alternative Datacenter Design Strategy, IEEE Micro, vol. 30, no. 4, pp. 25-28, July/Aug. 2010
[44] J. Karidis *et al.* 2009. True value: assessing and optimizing the cost of computing at the data center level. In Proc. of CF '09. ACM, New York, NY, 185-192.
[45] W. Iqbal *et al.* 2010. SLA-Driven Dynamic Resource Management for Multi-tier Web Applications in a Cloud. In Proc. of CCGrid' 10, pp.832-837.
[46] P. Campegiani *et al.* 2009. A General Model for Virtual Machines Resources Allocation in Multi-tier Distributed Systems. In Proc. of ICAS' 09.
[47] I. Goiri *et al.* 2010. Characterizing Cloud Federation for Enhancing Providers' Profit. In Proc. of Cloud' 10, pp. 123-130
[48] W. Vogels, Beyond Server Consolidation, ACM Queue, vol. 6, no. 1, 2008, pp. 20-26.
[49] X. Fan *et al.* 2007. Power provisioning for a warehouse-size computer. In Proc. of ISCA' 07.
[50] L. Rodero-Merino *et al.* 2010. From infrastructure delivery to service management in clouds. Future Gener. Comput. Syst. 26, 8 (Oct. 2010), 1226-1240.
[51] J. Choi *et al.* 2010. Power Consumption Prediction and Power-Aware Packing in Consolidated Environments. Computers, IEEE Transactions on , vol.59, no.12, pp.1640-1654, Dec. 2010
[52] A. Benoit *et al.* Scheduling Concurrent Bag-of-Tasks Applications on Heterogeneous Platforms. Computers, IEEE Transactions on , vol.59, no.2, pp.202-217, Feb. 2010
[53] P. Martí *et al.* 2009. Draco: Efficient Resource Management for Resource-Constrained Control Tasks. IEEE Trans. Comput. 58, 1 (January 2009), 90-105.
[54] H. Xi *et al.* Characterization of Real Workloads of Web Search Engines. In Proc. IISWC 2011.
[55] G. S. Sadhasivam *et al.* Design and Implementation of a Two Level Scheduler for HADOOP Data Grids. Int. J. of Advanced Networking and Applications 295 Volume: 01, Issue: 05, Pages: 295-300 (2010)
[56] http://www.sics.se/ aeg/report/node6.html
[57] C. A. Waldspurger *et al.* Lottery scheduling: Flexible proportional-share resource management.In Proc. OSDI 1994.
[58] W. Jin *et al.* Interposed proportional sharing for a storage service utility. In Proc. SIGMETRICS 2004.
[59] C. Luo *et al.* CloudRank: Benchmarking and Ranking Private Cloud Systems. Frontier of Computer Sciences.

Jianfeng Zhan received the Ph.D degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2002. He is currently an Associate Professor of computer science with Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include distributed and parallel systems. He was a recipient of the Second-class Chinese National Technology Promotion Prize in 2006, and the Distinguished Achievement Award of the Chinese Academy of Sciences in 2005.



Lei Wang received the master degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2006. He is currently a senior engineer with Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include resource management of cloud systems. He was a recipient of the Distinguished Achievement Award of the Chinese Academy of Sciences in 2005.

Xiaona Li received the master degree in software engineering from Beihang University, Beijing, China, in 2012. From 2010 to 2012, she was a visiting graduate student at Institute of Computing Technology, Chinese Academy of Sciences, China. She is currently a software engineer in Project Management Department in Baidu.

Weisong Shi received the Ph.D degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2000. He is currently an Associate Professor of computer science with Wayne State University. His current research interests include computer systems and mobile computing. He is a recipient of the NSF CAREER award.

Chuliang Weng received the PhD degree in computer software and theory from Shanghai Jiao Tong University (SJTU), China, in 2004. He is currently an associate professor with the Department of Computer Science and Engineering at SJTU. His research interests include parallel and distributed systems, system virtualization, and system security.

Wenyao Zhang received the PhD degree in computer science from Chinese Academy of Science, China, in 2003. He is currently an assistant professor with School of Computer Science, Beijing Institute of Technology. He is also a member of Beijing Lab of Intelligent Information Technology. His current research interests include scientific visualization, media computing, and parallel processing.

Xiutao Zang received the master degree in computer engineering from Chinese Academy of Sciences, Beijing, China, in 2010. He is currently a software engineer with Institute of Computing Technology, Chinese Academy of Sciences. His current research interests include testbeds for datacenter and cloud computing.