CloudBay: Enabling an Online Resource Market Place for Open Clouds

Han Zhao*, Ze Yu†, Shivam Tiwari*, Xing Mao†, Kyungyong Lee†, David Wolinsky‡, Xiaolin Li†, and Renato Figueiredo†
*Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL 32611
†Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611
‡Department of Computer Science, Yale University, New Haven, CT 06520

Abstract—**This paper presents CloudBay, an online resource trading and leasing platform for multi-party resource sharing. Following a market-oriented design principle, CloudBay provides an abstraction of a shared virtual resource space across multiple administration domains, and features enhanced functionalities for scalable and automatic resource management and efficient service provisioning. CloudBay distinguishes itself from existing research and contributes in a number of aspects. First, it leverages scalable network virtualization and self-configurable virtual appliances to facilitate resource federation and parallel application deployment. Second, CloudBay adopts an eBay-style transaction model that supports differentiated services with different levels of job priorities. For cost-sensitive users, CloudBay implements an efficient matchmaking algorithm based on auction theory and enables opportunistic resource access through preemptive service scheduling. The proposed CloudBay platform stands between HPC service sellers and buyers, and offers a comprehensive solution for resource advertising and stitching, transaction management, and application-to-infrastructure mapping. In this paper, we present the design details of CloudBay, and discuss lessons and challenges encountered in the implementation process. The proof-of-concept prototype of CloudBay is justified through experiments across multiple sites and extensive simulations.**

I. Introduction

The emerging cloud computing paradigm reshapes the way IT services are delivered with its ability to elastically grow and shrink the resource provisioning capacity on demand. It launches a new chapter for e-science and e-engineering applications that offers High Performance Computing (HPC) at scale. For example, the recent published top500 list includes Amazon's EC2 virtual cluster composed of over one thousand *cc2.8xlarge* instances [1]. In order to realize HPC-as-a-service with the full potential of cloud computing, it is best to take advantage of resources in an open marketplace across multiple clouds [2]. However, two major challenges still remain to be addressed. First, although end users are liberated from the arduous task of resource configuration, this burden is transferred to computational resource providers. Existing work either limits service to local area connectivity [3], or requires nontrivial resource and networking setup among all resource contributors [4]. Second, there lacks a flexible application-to-infrastructure mapping mechanism that accommodates differentiated service requirements and at the same time, maintains high efficiency for resource allocation across multiple clouds. Finally, it is critical to implement a fair pricing scheme in a multi-party cloud environment for both resource sellers and customers.

The work presented in this paper is supported in part by National Science Foundation (grants CCF-1128805, OCI-0904938, and CNS-0709329).

To overcome these hurdles, we propose CloudBay as a full-fledged solution for computational resource sharing and trading in an open cloud environment. CloudBay addresses the first challenge by incorporating decentralized self-configurable networking and self-packaging cloud toolsets. This design breaks the barrier of proprietary clouds and reduces efforts for resource joining, maintenance and query. It also helps cloud resource customers to deploy and maintain their applications using the shared cloud infrastructure. To address the second challenge, CloudBay implements an eBay-style trading mechanism. Specifically, user requests are classified as quality-sensitive and cost-sensitive depending on the offers the users are willing to make. The service scheduler in CloudBay assigns higher priorities to quality-sensitive service requests, and allows opportunistic provisioning of under-utilized resources through preemptive application execution. The competition among cost-sensitive service requests are resolved by an efficient auction mechanism that guarantees resource access for those users who value them the most. Service scheduling in CloudBay also supports distributed sites bid for jobs for optimal system-wide performance. Our proposed market-driven solution differs from Amazon's on-demand and spot IaaS in the following two aspects: (1) CloudBay is more flexible than Amazon's spot market because it allows for partially fulfilling user requests, whereas EC2 spot market only supports all-or-none resource acquisition. This feature is useful as HPC users often have fuzzy resource demand [5]. (2) Resource auction in CloudBay is based on a novel Ausubel auction model that encourages truthful bidding (i.e., bidders bid based on their true valuation), and achieves Vickrey efficiency compared with Amazon's spot market auction. We believe that the exploratory investigation presented in this study can open up significant perspectives of merging HPC and cloud computing in the long run.

In this study, we demonstrate that the following features render CloudBay a favorable implementation for HPC-as-a-service in an open cloud environment:

- **Scalable resource federation:** Leveraging P2P-based virtual networking, CloudBay achieves scalable resource bridging by disseminating routing information in a decentralized fashion.
- **Self-configurable resource provisioning:** We develop a number of programs to automate network configura-

tion and application deployment in CloudBay. Our work greatly simplifies the task of resource providers and provides timely services to end users.
- **Fair resource allocation:** A fair allocation of resources allows the service qualities received by end users to be proportional to the values they pay. In CloudBay, we implement an efficient market-driven matchmaking mechanism to achieve this goal.
- **Flexible resource usage:** CloudBay accommodates a variety of resource usage models and offers differentiated levels of services to end users. For example, it can support both rigid and flexible parallel application execution.

The rest of the paper is organized as follows. In Section II, we provide an overview of CloudBay and introduce the design and implementation of virtualization tools facilitating resource sharing. In Section III, we explain the details of the job scheduling algorithms in CloudBay. The evaluation results of our prototype CloudBay implementation are presented in Section IV. Section V describes related work to our proposed system. And finally, we conclude our paper and discuss possible future research directions in Section VI.

II. Design Overview

A. Architecture

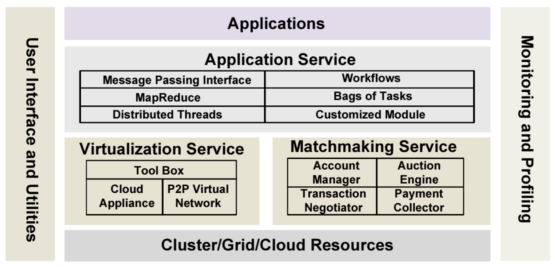


Fig. 1. CloudBay Architecture

Figure 1 depicts the architecture of CloudBay. The design goal of CloudBay is to provide a suite of tools that facilitate computational resource sharing and enhance application-to-infrastructure mapping. To fulfill this goal, CloudBay is designed as a service-oriented architecture that seamlessly bridges the gap between applications and resources. First, CloudBay provides resource virtualization services on top of the bare hardware, including: (1) a P2P virtual networking tool that supports scalable and cross-domain resource stitching; and (2) an application-aware VM image called Cloud Appliance that packages grid/cloud computing toolsets and self-configurable networking facilities. With the support of the virtualization service, computational resources residing on different domains can be easily connected together to form an ad-hoc cluster over wide-area networks.

Next, accompanied by the virtualization services, CloudBay offers market-oriented resource-request matchmaking services for both quality-sensitive and budget-sensitive users. The core functionalities include: (1) an account manager managing resource seller and buyer accounts; (2) a transaction negotiator that helps to arrange user requests based on the supply and demand level of the current resource market; (3) an auction engine that resolves resource competition when necessary; and (4) a payment collector that collects fees for resource rental. We will cover the details of the market-driven service scheduling scheme in Section III.

Finally, CloudBay offers a variety of popular programming models for deploying and running distributed applications. This is achieved by interfacing with pre-packaged softwares supporting application compilation, run-time configuration and job management. For example, the current implementation of Cloud Appliance image packages MPI library and MyHadoop [6] for HPC application tuning and running. Additional functionalities such as interfacing with users, monitoring and profiling are traversal to the entire CloudBay service stack.

B. Use Case Illustration

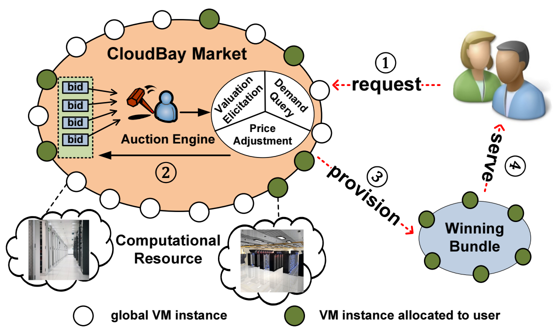


Fig. 2. A simple use case for CloudBay

Figure 2 illustrates a simple working scenario in CloudBay. A user submits a bid request (detailed in Section III-A) to the CloudBay server seeking to access resources within his budget constraint (step ①). The CloudBay server accepts the bid request and places it together with other incomplete bid requests in the system. If the request cannot be satisfied by the current resource supply, a dynamic ascending auction is launched by the centralized auction engine (step ②). Suppose this user wins 6 VM instances as a result of the auction, the CloudBay server will automatically provide connectivity that bundles the allocated instances into a cluster (step ③). The bundle now becomes invisible to other users and is isolated from other resources in the system. CloudBay employs Condor [7] to manage user submitted jobs, and randomly designates a node within the winning bundle as the head node. The job submitted by the user will be forwarded to a local client node running the *condor_schedd* daemon, and CloudBay will let Condor take over the rest of the work (step ④).

C. Resource Virtualization Tools

This section introduces our previous work on platform, resource and network virtualization. These techniques form the basis of scalable and self-configurable resource sharing in CloudBay. Since this paper mainly focuses on resource management and service scheduling issues, we only present an overview of these virtualization tools, and refer the readers to [8], [9], [10] for implementation details.

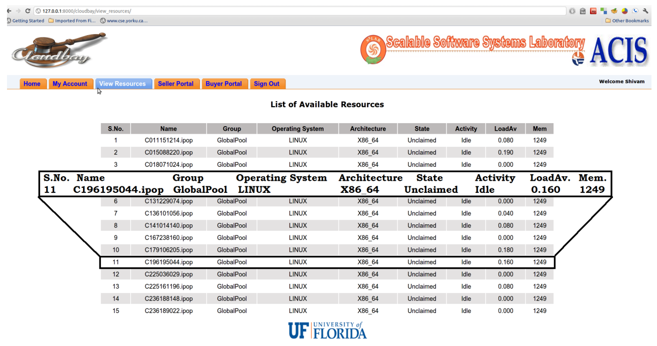
a) IP-over-P2P (IPOP) [8], [9]: CloudBay is designed to provide infrastructure support to scale up to large numbers of geographically distributed resources over wide-area networks. To address this requirement, we developed IPOP, a P2P-based self-configurable tool for network virtualization. CloudBay uses IPOP to enable elastic resource provision and relinquish, and attains the following benefits: (1) scalable network management, because routing information is self-configured and disseminated in a decentralized manner. (2) resilient to failure, because P2P networks offer more robustness against failure than a centralized approach. (3) easy accessibility, due to IPOP's ability to traverse NAT/firewalls.

b) Cloud Appliance: Cloud Appliance directly extends our previous work of Grid Appliance [10]. It packages cloud computing toolsets into an application-aware virtual machine image (available in VMware, Virtual Box and KVM), and supports on-demand resource clustering. A resource provider may choose to launch a Cloud Appliance on the physical host machine, which will automatically place the contributed resource slice into the global CloudBay resource pool. Alternatively, a resource provider may also choose to install separate CloudBay package on the fly (e.g., the package *grid-appliance-base* offers virtual networking functionality and can be installed from ubuntu). In essence, a Cloud Appliance is an integrated middleware that encapsulates a full job scheduling software stack. *It hides the heterogeneity of various cloud platforms and provides a uniform interface to different cloud resource providers.* Cloud Appliance also allows resource customers to run unmodified, binary software executables without imposing platform-specific APIs that applications must be bound to. Scheduling service in Cloud Appliance directly interfaces with the Condor scheduler for job management. Finally, Cloud Appliance offers sandboxing security such that undesirable behaviors are confined to an isolated VM instance.

D. Autonomic Resource Pooling

This section presents the implementation details of resource pooling in CloudBay. Resource pooling involves development of: (1) a centralized resource pool accessible to all users; and (2) an isolated resource pool allocated to a particular user. Our implementation uses a centralized approach to provide autonomic services for resource configuration and management. Specifically, a resource manager process, running side-by-side with the Condor central manager, is implemented on the CloudBay server that helps to monitor and manage active resources in the system. We automate the resource joining process by packing a booting script written in Python into the Cloud Appliance VM image. To contribute a VM instantiated by the Cloud Appliance, a resource provider first submits a resource join request from a web interface, and then downloads a certified configuration file bound to the VM. This process is termed as "*floppy insertion*" in CloudBay.

The front end of CloudBay is implemented using Django [11], allowing users to easily interact with the server. Figure 3 shows a resource summary page that returns the *condor_status* result using Condor SOAP API. When a resource



List of Available Resources

S.No.	Name	Group	Operating System	Architecture	State	Activity	LoadAv	Mem
1	C011151214.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.080	1249
2	C015088220.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.190	1249
3	C018071024.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.000	1249
6	C131229074.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.000	1249
7	C136101056.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.040	1249
8	C141014140.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.080	1249
9	C167238160.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.000	1249
10	C179106205.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.180	1249
11	C196195044.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.160	1249
12	C225036029.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.000	1249
13	C225161196.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.080	1249
14	C236188148.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.000	1249
15	C236189022.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.000	1249

S.No.	Name	Group	Operating System	Architecture	State	Activity	LoadAv.	Mem.
11	C196195044.ipop	GlobalPool	LINUX	X86_64	Unclaimed	Idle	0.160	1249

Fig. 3. User interface for viewing resources in the global pool

bundle is allocated to some request, the resource manager process will create a new configuration file (floppy) for each VM within the bundle. In our previous implementation [12], users have to manually configure the allocated resource bundle through the web interfaces. Whereas in CloudBay, the resource manager automatically locates the VMs based on their addresses on the IPOP virtual network and transfers the floppies to them via *scp*. This autonomic floppy insertion process enables CloudBay to form an isolated resource pool upon request and greatly simplifies resource allocation.

III. Market-driven Service Scheduling

This section presents the design details for user service scheduling in CloudBay. Due to space limitation, we focus on the eBay-style differentiated service provisioning, HPC job submission, and auction mechanism design. The details about CloudBay economy bootstrapping and interfaces for valuation expression are omitted. These details will be addressed in a future extended version of this paper.

A. Resource and Service Request Models

We consider the resource pool of CloudBay consisting of **dedicated** and **high-performance** computing and storage facilities (e.g., clusters and network shared file systems) that span across organizational and national boundaries. Leveraging techniques presented in Section II, these facilities are easy to confederate within a common resource namespace, forming what is referred to as a *science cloud*. Note that CloudBay does not target at non-dedicated and cheap resource in the volunteer computing model because it is hard to guarantee quality of service for quality-sensitive HPC users in a highly dynamic environment. On the resource market formed by CloudBay, resource providers partition their resources into standard sized resource slices that are instantiated using Cloud Appliances, and delegate the task of negotiation and selling to CloudBay. The resource rental model in CloudBay is similar to that used in Amazon EC2, where users purchase computing services in the unit of instance·hours. However, rather than providing IaaS where users have complete control over the allocated VMs and build their own software stacks, CloudBay is more PaaS-oriented that packs a computing platform and job management functionalities as a service.

Let $\mathcal{R}$ be the set of VM instances within the global resource pool. CloudBay allows for $\mathcal{V}$ classes of VM instances to be created by resource providers (e.g., small, medium and large VM instances). All instances within the same class, i.e., $R_v \in \mathcal{R}, v \in \mathcal{V}$, have homogeneous configurations. We denote the set of user requests by $\mathcal{U}$, each request $U \in \mathcal{U}$ is limited to a set of VM instances within the same class. If a user wishes to run a job on a set of heterogeneous resources, he can simply create a request group in CloudBay that bundles the VMs granted by all the requests sharing the same job configuration.

The need for differentiated service provisioning is imminent because it improves utilization of the infrastructure. Traditional HPC centers allow different job priority classes and use backfilling scheduling [13] to reduce fragmentation of system resources, while modern IaaS providers in cloud computing tend to jointly schedule on-demand and opportunistic resource requests, as is the case of Amazon's launch of spot market in addition to the on-demand service. As HPC merges with cloud computing, the question becomes, *how to implement the differentiated request model in modern HPC centers equipped with cloud infrastructure?* In CloudBay, we develop a service scheduling approach inspired by the transaction model used in eBay. Before we proceed to describe our approach, we clarify the assumptions and specifications of the user request model in the next few paragraphs.

We consider a heavily loaded system where user demand is greater than resource supply. Otherwise, all user requests are able to be accommodated and the question becomes *which site to choose for optimal job placement*. CloudBay plans to adopt an adaptive scheduling policy to solve both cases. When demand falls below the supply level, the auction engine in CloudBay automatically switch from the seller-initiated mode to bidder-initiated mode, i.e., all the eligible resource providers can bid for job execution according to their reputations, current site load and data locality. We plan to add the bidder-initiated auction in our future work.

We define two types of user requests in CloudBay.

- **"buy"-request** – submitted by quality-sensitive users and is analogous to the option of *buy-it-now* on eBay. The submitted job is likely to be associated with a deadline, and the interruption in service is generally undesirable (non-preemptive). Note that we cannot promise immediate access to the resource because the system might become so congested filling with non-preemptive jobs[1].
- **"bid"-request** – submitted by budget-sensitive users and is analogous to those who *bid* on goods on eBay wishing to find a deal. There is no deadline associated with bid-requests. The bidder may specify a expected duration of job execution, or simply let it run to completion. The jobs are characterized as failure resilient that interruption in service does not compromise the computation integrity.

[1]If it happens and some request is rather urgent, the user is given the option to overpay a large amount in order to squeeze in. Just as the way to deal with sellers canceling buy-it-now transaction on eBay, the transaction negotiator in CloudBay needs to compensate the preempted job owner for violating SLA.

Given a mixture of the two types of user requests, the goal of service scheduling is to achieve **fair** resource allocation while maintaining **high utilization** of the infrastructure. By fair we mean: (1) jobs associated with high bids[2] should take precedence over low-bid jobs; and (2) market price of resources is not over- or under-valuated. By high infrastructure utilization we mean that the matchmaking service should make resource allocation decisions in a timely manner, and grants resource access rights to end users whenever there is a chance. We will illustrate the design details of service scheduling in CloudBay in the later sections.

B. Job Submission

CloudBay directly interfaces with Condor for job management because of Condor's ability to support both dedicated and opportunistic job execution. We create a uniform web interface that allows users to upload executables and job configuration files to the CloudBay server. The job submission process in CloudBay is illustrated in Figure 4. First, when a resource bundle is allocated to serve a request, the CloudBay server will send the job to a gateway node within the winning bundle running the *condor_schedd* daemon. After that, the local Condor server will fetch the job information and schedule the job in the local pool (see the left of Figure 4). If this job get preempted some time later, the CloudBay server will store the computing state (through checkpointing) as well as the original job configuration. Suppose after a while, a new pool of resources become available again, the CloudBay server will redirect the job information to a gateway node in the new pool to resume the job execution (see the right part of Figure 4).

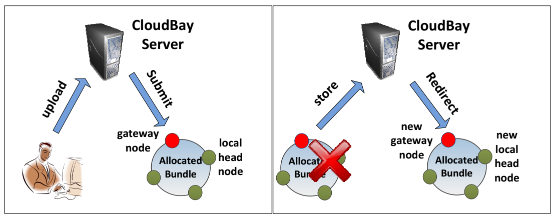


Fig. 4. Job submission in CloudBay. Left: a new job submit to an isolated resource bundle. Right: a job is preempted from the old allocated bundle and resumed at the new bundle.

C. Service Scheduling in CloudBay

The procedure for request scheduling in CloudBay is summarized in Algorithm 1. In order to eliminate request queueing, the transaction negotiator tries to make an allocation decision whenever a service request arrives. An incoming request issued by some quality-sensitive user takes precedence over all bid requests and gain access to the desired resource bundle whenever possible (line 3 to 12). On the other hand, if an incoming request is of bid type, it is scheduled to compete resources with other bid requests when current resource supply cannot meet its demand. The auction engine will trigger a two-stage Ausubel auction (line 17) to resolve the competition.

[2]the buy-request is viewed as a special case of bid-request where users are willing to pay a fixed predefined amount.

Algorithm 1: Request scheduling in CloudBay

```
1  begin
2      examine incoming request type
3      case buy-request
4          if supply ≥ demand then
5              allocate VMs as requested
6
7          else if ∃ unfinished bid jobs AND their aggregate resource occupation ≥ demand then
8              preempt jobs from low-bid to high until demand is satisfied
9          else
10             negotiate with the user with two options
11             • try at a later time
12             • pay large fine for immediate resource access
13     case bid-request
14         if supply ≥ demand then
15             allocate VMs as requested and collect payments accordingly
16         else
17             start a two-stage Ausubel auction, reconsider bid-requests for all incomplete jobs
18             allocate according to the auction result
```

The original Ausubel auction (also known as the efficient ascending auction) was proposed in [14], and possess two appealing properties that make it a good match for our design goal. First, it is *computationally tractable*. Second, it employs a non-linear payment method to *eliminate the incentives of strategic bid behaviors*. However, we cannot directly apply the original Ausubel auction to our scheduling context because of the following difficulties: (1) Ausubel auction uses iterative price adjustment to balance market demand and supply. In practical algorithmic design, the convergence to market equilibrium state might take long time due to price oscillating around the market clearing price. The reason behind this is that it's impossible to determine the step length for price adjustment unless we know the search stop point (the market clearing price) in advance. (2) Some bidders have all-or-none resource acquisition preference. They may suddenly drop out of the auction when price is adjusted. If that is the case, the market equilibrium state may not exist at all. In order to determine resource allocation, we have to extend the feasible region for the solution. Specifically, suppose n bidders bid for m VM instances of certain class v. Let each bidder's demand be d_i^t at auction round t (the auction is iterative). We relax the convergence condition of $\sum_{i=1}^{n} d_i^t = m$ to $\sum_{i=1}^{n} d_i^t \leq m$. Note that such relaxation will result in efficiency loss. However, as we use backtracking to find the closest point to equilibrium state, such loss is relatively small.

In the original Ausubel auction, the payment calculation is carried out along with the procedure to search for the market equilibrium price. We propose a *two-stage Ausubel auction* to overcome the first difficulty. In the first stage, the algorithm quickly locates a final market price. With this information, we can decide the price adjustment step and simulate the original Ausubel payment calculation procedure in the second stage. We assume user's valuation to resource bundle is monotonic and strictly concave, i.e., allocated resources exhibit diminishing rewards to users. For a given VM class v, let p_i^k (we omit v for brevity of notations) be user i's bid price for the kth allocated instance·unit time. To obtain the market clearing price, we perform a binary search on a sorted list of such bid prices. When two bids submitted by two different users tie with each other, the algorithm assigns higher priority to the bid submitted at an earlier wall clock time. If the algorithm fails to converge to a market clearing price, it will backtrack to find the best feasible allocation yielding $\sum_{i=1}^{n} d_i^t \leq m$. The final allocation for each user is determined by evaluating the marginal bid vector using the returned final market price.

D. Payment Accounting

In the second stage, the auction engine simulates the auctioneer-bidder communications as proposed in the original Ausubel auction [14] in an iterative manner. The payment collector interacts with the auction engine in order to calculate payment amounts for all bidders. We briefly summarize the payment accounting method as follows. First, at each round t, the auctioneer calculates the aggregate reserved bundle ρ_i^t for bidder i by comparing the market supply against the aggregate demand from i's opponents:

$$\rho_i^t = \max\{0, m - \sum_{j \neq i} d_j^t\} \tag{1}$$

Accordingly, the round reserved bundle μ is defined as the difference of the aggregate reserved bundle at adjacent rounds:

$$\begin{aligned} \mu_i^1 &= \rho_i^1 \\ \mu_i^t &= \rho_i^t - \rho_i^{t-1} (t > 1) \end{aligned} \tag{2}$$

Note that $\mu_i^t \geq 0$ because the aggregate demand from i's opponents is weakly diminishing. If $\mu_i^t > 0$, then this amount of allocation is referred to as "clinched" by bidder i at current round price p^t. Suppose i wins A_i at the final round T, the total payment of i is calculated as:

$$P_i(A_i) = \sum_{t=1}^{T} p^t \mu_i^t \tag{3}$$

One virtue of the Ausubel auction is that it replicates the outcome of the static Vickrey auction. This property is desirable because untruthful users experience degraded performance in computing markets [15]. The proposed auction is incentive compatible (proof detailed in [14]), and results in fair market pricing upon convergence.

E. Discussion

Our scheduling decision is made on request. This might cause constant thrashing of the low-bid requests. In fact, such an effect is the tradeoff to reduced resource utilization in periodic scheduling. To alleviate this problem, we can compensate the preempted low-bid jobs for a small amount. As the compensation accumulates, the job becomes more resilient to preemption. This is an interesting topic because doing so seems to violate our design goal of fairness. Due to the space limitation we will not discuss the solution further in this paper.

IV. Evaluation

A. Performance Evaluation for Resource Pooling

We develop and deploy an experimental CloudBay platform composed of 32 VM instances, with 20 of them setup on FutureGrid [16], 8 on Amazon EC2, and 4 on local lab machines at the University of Florida. Each instance is equipped with 1.5G memory and 1 virtual CPU core running at 2.66GHz, and is pre-configured with Condor supporting both dedicated and opportunistic scheduling. The CloudBay server process is implemented and run on a separate machine that also works as the head node for the global Condor resource pool.

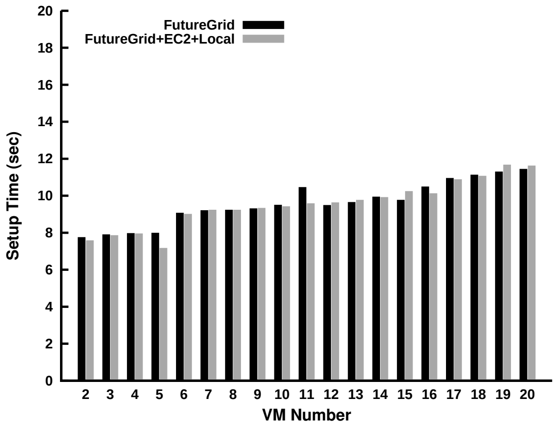


Fig. 5. Experiment for autonomic resource pooling

First, we examine the setup time for creating an isolated bundle of VM instances. In particular, the setup time is the time elapsed from the moment an allocation decision is made until all resources in the bundle are shown using the *condor_status* command. According to Section II-D, the setup time comprises: (1) generating floppy file for network configuration; and (2) notifying the VM instance within the winning bundle about the information of the new Condor head node by transferring the floppy file and modify the local Condor configuration. Figure 5 shows the measurement of bundle setup time for multiple VM instances cross three different sites and on FutureGrid only. We observe that the setup time displays an increasing trend as the number of VM instances increases for both experiments. Since cloud users typically request resources over hours, the experiment results indicate that automatic resource pooling in CloudBay imposes a *trivial overhead* to the total resource rental period.

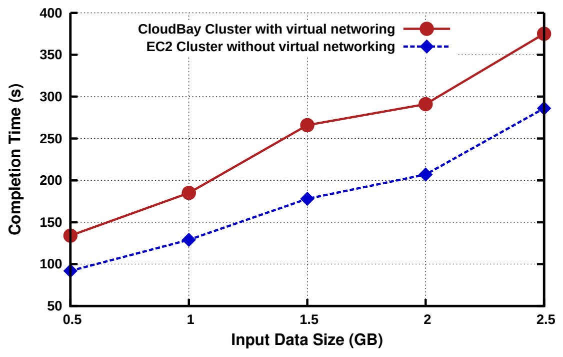
Next, we investigate the performance of CloudBay virtual networking by stress testing the Hadoop cluster with and without IPOP virtual network, respectively. Specifically, we deploy a CloudBay Hadoop cluster, with two VM instances hosted on the UF campus network, and the other two VM instances hosted on Amazon's EC2 platform. All instances have the same resource configuration with EC2's m1.large instance type. For the purpose of performance comparison, we also setup a EC2 homogeneous Hadoop cluster connected by EC2's internal network. Two MapReduce programs, *wordcount* and *terasort*, are selected as benchmark programs. For each program, we vary the input file size from 0.5G to 2.5G, and measure the completion time of all the map and reduce tasks. The results are shown in Figure 6(a) and Figure 6(b). From the figure, we observe that the heterogeneous networking environment in CloudBay virtual cluster achieves broader deployment scope at the cost of degraded execution time. Using the Hadoop monitoring tool, we observe that the two local nodes greatly straggle the program progress due to the intermediate data transfer from the EC2 site (the master node is located at EC2 side). In addition, the performance gap in terms of completion time difference is relatively consistent in the wordcount program, but increases significantly as the input data size grows. This phenomenon is primarily contributed to the difference of intermediate data transfer between the map and the reduce phase. For wordcount, the size of the word list generated from the map tasks is almost the same for all input[3], while for terasort, the size of the intermediate data for the reduce tasks is increasing all the time. As the data sharing problem becomes more serious in a virtual cloud environment [17], the research for location-aware scheduling mechanism for data-intensive applications is therefore imperative in the future development of CloudBay.

B. Performance Evaluation for Service Scheduling

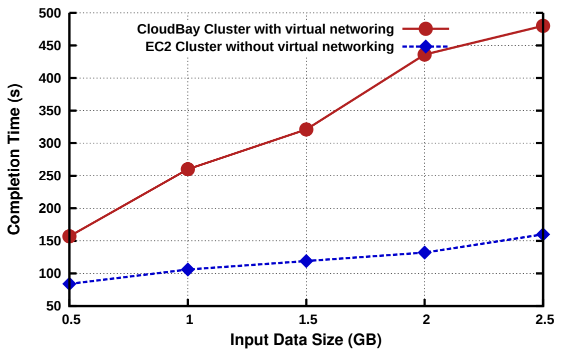
This section studies CloudBay's ability to schedule mixed-type service requests. Our investigation answers two questions from different perspectives. First, from the perspective of the resource providers, we are interested in understanding how much resource time is consumed by frequent preemptions of low-bid service requests (*preemption overhead*). Next, from the perspective of the end users, we are concerned about the perceived lag of service completion (*service delay*) against the willingness to pay for the service (*offered price*).

The CloudBay platform is in prototype testing stage and does not accumulate enough user base. Therefore, our evaluation is simulation-based. We implement a discrete-event simulator using the *Simpy* [18] simulation package based on Python. In the simulation, we create 512 single-core VM instances to serve incoming user requests. Each request can ask for up to 32 instances for running applications. The requested VM number per request is uniformly distributed in the range of $(0, 32]$. For buy-requests, the resource reservation price in a unit of time is set to 20. According to Amazon's spot price history [19], we set the offered prices for bid-requests to fall in the range of $(0, 20)$, and follows a normal distribution with $\mu = 8$ and $\sigma = 4$. The job arrival process is assumed to follow a Poisson distribution. By varying the rate parameter λ, we can simulate system behaviors under different workloads.

[3]We simply append the same text to generate larger size of input



(a) WordCount



(b) Terasort

Fig. 6. Performance evaluation for virtual netoworking

We generate synthetic user-requested resource usage times based on a realistic workload scenario described in [20]. The workload traces include a Condor workload from the University of Notre Dame, and an on-demand IaaS cloud workload from the University of Chicago Nimbus science cloud. Based on these traces, the requested times are set spanning a relatively long period of time (e.g., a typical request will ask for resource rental over several hours).

In the first set of simulations, we assume the preemption time is linearly proportional to the number of VM instances to relinquish and reset. The preemption process includes the time to save program state (checkpointing), restart the networking configuration process and reconfigure local Condor service. This process can take several minutes for repooling a large number of VM instances. Figure 7 shows the results calculated over a one-month period simulation run. We vary the percentage of bid requests to generate different flows of incoming requests. The labels of *high*, *medium*, and *low* workload correspond to the average system utilization of 83.3%, 66.5%, and 53.4%, respectively. Note that the presented results are *relative measurements*, e.g., the bid overhead is measured as the preemption loss with regard to the total resource time occupied by bid-requests, not to resource time occupied by all requests. Therefore, the overall overhead is approximately the weighted sum of bid-request and buy-request overhead. When less bid requests are present, they are subject to frequent preemption by the dominant buy requests. As a result, we observe spikes at the initial phase for bid requests. However, the overall overhead is relatively stable in all the tested scenarios, contributing around 1.8% to the total busy resource cycles, indicating that CloudBay is suitable for processing high throughput service requests in an open cloud environment.

In the second set of simulations, we create $2,000$ synthetic requests and investigate the average service delay with regard to different user bid prices. The *service delay factor* is defined as the ratio of the actual service completion time to the user requested time. A factor of 1.0 means there is no service delay. We conduct five simulation runs with varying percentages of bid request from 30% to 70%. For each run, the system utilization averages at around 83%, and the total simulated time is about 50 days. The results are shown in Figure 8. As

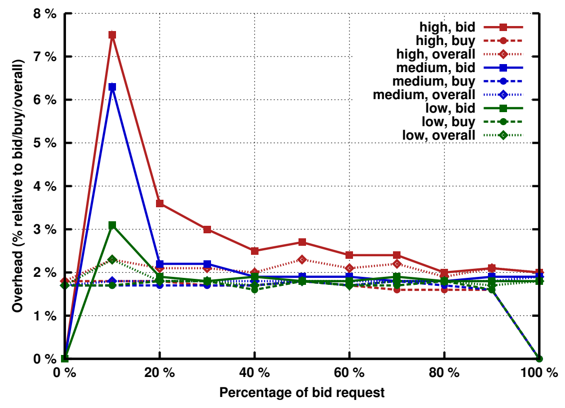


Fig. 7. Evaluation of the overhead due to preemption

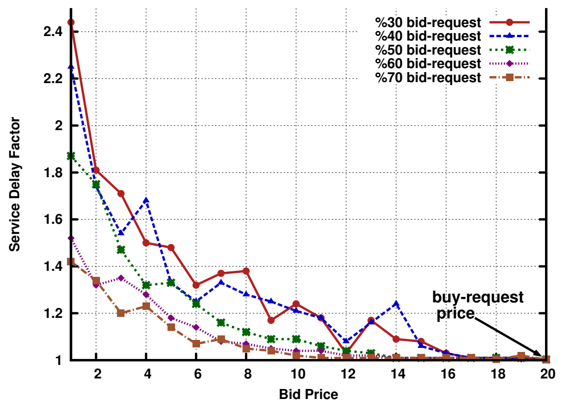


Fig. 8. Service delay factor VS. Offered price

we expected, higher bid price leads to less service delay in general. However, we also observe a few irregular points on the figure, and the less bid requests a curve gets, the more wrinkled a curve exhibits. This can be explained as follows: (1) a low-bid request might get scheduled without blocking simply because there are available slots in the system; (2) the bid price is randomly generated for each request such that the number of bid requests for a particular price is insufficient. In general, we can conclude that CloudBay achieves fair resource allocation for serving differentiated user requests.

V. Related Work

There has long been significant interest in investigating the application of economic approaches for resource management in distributed systems. According to Wolski et al. [21], two types of market strategy are commonly used in a computational economy, namely commodities markets and auctions. Auctions are simple to implement and are efficient to sell off computing cycles to contending users. Therefore, auctions achieved wide applications in early computational ecosystems such as Spawn [22], Popcorn [23], and Tycoon [24]. Another early work was Nimrod/G [25], where grid resources were allocated based on user-negotiated contracts with the resource sellers. Most systems were designed for early distributed computing infrastructure such as dedicated clusters and computational grids, and did not account for the latest technology advance in networking and hardware virtualization.

In the era of cloud computing, due to the service-oriented paradigm shift, market-driven distributed systems become commercialized in the next-generation data centers. Efforts were made to take advantage of cloud computing for efficient resource sharing [26]. However, the role of CloudBay is not to serve as yet another IaaS, PaaS, or SaaS provider, but rather to bridge the scattered scientific community in support of HPC application development and delivery.

VI. Conclusion and Future Work

We presented a novel CloudBay framework for large-scale computational resource sharing. Equipped with virtual networking and application-aware virtual appliances, CloudBay achieves ad-hoc self-organization, discovery and grouping of distributed resources without incurring extra deployment and management efforts from both resource providers and end users. Moreover, CloudBay implements a market-driven service scheduling policy that accommodates a mixture of user request models, and efficiently distributes idle resources to users in a cost-effective manner. The pricing and payment accounting policies boosts utilities for multiple trading parties, and guarantees incentive compatibility for bidders. Utilizing services provided by CloudBay, researchers with domain knowledge can comfortably deploy their parallel applications using popular parallel programming models on a resource bundle assembled from multiple organizations.

CloudBay is still in its infancy stage and is actively evolved towards a fully functional system. We have already deployed virtual appliances across a variety of open and private cloud platforms, including university clusters, FutureGrid, and Amazon EC2. Our experimental results demonstrated the effectiveness of deploying CloudBay in production HPC centers. The system performance due to market-driven scheduling was validated though simulations in Section IV. In the future, we plan to fulfill the design of CloudBay, and collect real-world data of user behaviors for investigations. Another possible research direction is to design and evaluate bidder-initiated auction in which placement of job execution is determined by efficient auction forms for optimal system performance. We expect that our experiences gained from the design and implementation of CloudBay would open a new research avenue for realizing HPC-as-a-service, and push the boundary for new cloud computing usage models.

References

[1] "HPC on AWS," http://aws.amazon.com/hpc-applications/.

[2] O. Krieger, P. McGachey, and A. Kanevsky, "Enabling a marketplace of clouds: VMware's vCloud director," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 103–114, 2010.

[3] H. Abbes, C. Cerin, and M. Jemni, "Bonjourgrid: Orchestration of multi-instances of grid middlewares on institutional desktop grids," in *IPDPS'09*, 2009, pp. 1–8.

[4] W. Cirne, F. Brasileiro, N. Andrade, L. Costa, A. Andrade, R. Novaes, and M. Mowbray, "Labs of the world, unite!!!" *Journal of Grid Computing*, vol. 4, no. 3, pp. 225–246, 2006.

[5] C. Bailey Lee, Y. Schwartzman, J. Hardy, and A. Snavely, "Are user runtime estimates inherently inaccurate?" in *JSSPP'04*, 2005, pp. 253–263.

[6] S. Krishnan, M. Tatineni, and C. Baru, "myhadoop - hadoop-on-demand on traditional hpc resources," San Diego Supercomputer Center, University of California San Diego, Tech. Rep. SDSC TR-2011-2, 2011.

[7] "Condor Project Home Page," http://research.cs.wisc.edu/condor/.

[8] A. Ganguly, A. Agrawal, O. P. Boykin, and R. Figueiredo, "IP over P2P: Enabling self-configuring virtual IP networks for grid computing," in *IPDPS'06*, 2006.

[9] D. I. Wolinsky, Y. Liu, P. S. Juste, G. Venkatasubramanian, and R. Figueiredo, "On the design of scalable, self-configuring virtual networks," in *SC'09*, 2009, pp. 13:1–13:12.

[10] D. Wolinsky and R. Figueiredo, "Experiences with self-organizaing, decentralized grids using the grid appliance," in *HPDC'11*, 2011.

[11] "Django," https://www.djangoproject.com/.

[12] D. I. Wolinsky and R. Figueiredo, "Grid appliance user interface," http://www.grid-appliance.org, September 2009.

[13] B. G. Lawson and E. Smirni, "Multiple-queue backfilling scheduling with priorities and reservations for parallel systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 29, pp. 40–47, 2002.

[14] L. M. Ausubel, "An efficient ascending-bid auction for multiple objects," *American Economic Review*, vol. 94, no. 5, pp. 1452–1475, 2004.

[15] S. Shudler, L. Amar, A. Barak, and A. Mu'alem, "The effects of untruthful bids on user utilities and stability in computing markets," in *CCGrid'10*, 2010, pp. 205–214.

[16] "FutureGrid," Available: http://futuregrid.org/.

[17] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving mapreduce performance in heterogeneous environments," in *OSDI'08*, 2008, pp. 29–42.

[18] "Simpy," Available: http://simpy.sourceforge.net/.

[19] "Cloud Exchange," Available: http://cloudexchange.org/.

[20] P. Marshall, K. Keahey, and T. Freeman, "Improving utilization of infrastructure clouds," in *CCGrid'11*, 2011, pp. 205–214.

[21] R. Wolski, J. Plank, T. Bryan, and J. Brevik, "G-commerce: market formulations controlling resource allocation on the computational grid," in *IPDPS'01*, 2001.

[22] C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and W. Stornetta, "Spawn: a distributed computational economy," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 103–117, 1992.

[23] O. Regev and N. Nisan, "The POPCORN marketan online market for computational resources," in *ICE'08*, 1998, pp. 148–157.

[24] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent Grid Syst.*, vol. 1, pp. 169–182, 2005.

[25] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: an architecture for a resource management and scheduling system in a global computational grid," in *Proceedings of the fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, 2000, pp. 283–289.

[26] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken, "Using a market economy to provision compute resources across planet-wide clusters," in *IPDPS'09*, 2009, pp. 1–8.