Cost-aware Cloud Bursting for Enterprise Applications[1]

Tian Guo, University of Massachusetts Amherst
Upendra Sharma, IBM Research
Prashant Shenoy, University of Massachusetts Amherst
Timothy Wood, The George Washington University
Sambit Sahu, IBM Research

The high cost of provisioning resources to meet peak application demands has led to the widespread adoption of pay-as-you-go cloud computing services to handle workload fluctuations. Some enterprises with existing IT infrastructure employ a hybrid cloud model where the enterprise uses its own private resources for the majority of its computing, but then "bursts" into the cloud when local resources are insufficient. However, current commercial tools rely heavily on the system administrator's knowledge to answer key questions such as when a cloud burst is needed and which applications must be moved to the cloud. In this paper we describe Seagull, a system designed to facilitate cloud bursting by determining which applications should be transitioned into the cloud and automating the movement process at the proper time. Seagull optimizes the bursting of applications using an optimization algorithm as well as a more efficient but approximate greedy heuristic. Seagull also optimizes the overhead of deploying applications into the cloud using an intelligent precopying mechanism that proactively replicates virtualized applications, lowering the bursting time from hours to minutes. Our evaluation shows over 100% improvement compared to naïve solutions but produces more expensive solutions compared to ILP. However, the scalability of our greedy algorithm is dramatically better as the number of VMs increase. Our evaluation illustrates scenarios where our prototype can reduce cloud costs by more than 45% when bursting to the cloud, and that the incremental cost added by precopying applications is offset by a burst time reduction of nearly 95%.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms: Design, Prototype, Algorithms, Performance

Additional Key Words and Phrases: Hybrid Clouds, Resource Management, Live Migration

1. INTRODUCTION

Many enterprise applications see dynamic workloads at multiple time scales. Since predicting peak workloads is frequently error-prone and often results in underutilized systems, cloud computing platforms have become popular with their ability to rapidly provision computing and storage capacity to handle workload fluctuations. At the same time, many medium and large enterprises have significant current investments in IT data centers that is often sufficient for the majority of their needs, while offering greater control and lower operating costs than the cloud. However, workload spikes, both planned and unexpected, can sometimes drive the resource needs of enterprise applications above the level of resources available locally. Rather than incurring capital expenditures for additional server capacity to solely handle such infrequent workload peaks, a hybrid model has emerged where an enterprise leverages its local IT infrastructure for the majority of its needs, and supplements with cloud resources whenever local resources are stressed.

[1]This paper is an expanded version of a short paper that appeared in USENIX 2012 Annual Technical Conference

This research was supported in part by NSF grants CNS-0855128, CNS-0916972, CNS-1117221, CNS-1253575, OCI-1032765 and an IBM OCR award and an Amazon AWS grant.

DOI: http://dx.doi.org/10.1145/0000000.0000000

This hybrid technique, which is referred to as "cloud bursting", allows the enterprise to expand its capacity as needed while making efficient use of its existing resources. Cloud bursting typically relies on virtualization technologies to encapsulate applications in virtual machines that can be transitioned between different hardware platforms with relative ease. While commercial and open-source virtualization tools are beginning to support basic cloud bursting functionalities [Openstack 2012] [Opennebula 2012] [VDataCenter 2012], the primary focus has been on the underlying mechanisms to enable the transition of VMs between locations or to seamlessly integrate the networks at different sites. These systems leave significant policy decisions in the hands of system administrators, who must manually determine when to invoke cloud bursting and which applications to "burst". Such decision processes require individual administrators to have profound knowledge of the running applications, and as a result often lead to poor choices in terms of minimizing cloud costs or reducing downtime during the transition. Further, unanticipated spikes that arrive at off hours may see a delayed response unless support staff are available at all times.

One of the *insights* of our work is that rather than naïvely moving an overloaded application to the cloud, it may sometimes be cheaper and faster to move one or more "smaller" applications and then assign the locally freed-up resources to the overloaded application. Judiciously making these choices manually is difficult especially when there are a large number of diverse applications in the data center and different cloud platform pricing models.

Bursting an application to the cloud involves copying its virtual disk image and any application data. Since this disk state may be large, consisting of tens or hundreds of gigabytes, a pure on demand migration to the cloud may require hours to copy this large amount of data. A second *insight* of our work is that occasional precopying of virtual disk snapshots of a few overload-prone applications can significantly reduce the cloud bursting latency, since only the incremental delta of the disk state needs to be transferred to reconstruct the image in the cloud. Our work examines the impact of judiciously choosing the candidate applications for such precopying.

We have developed a system called Seagull to address the above challenges; Seagull automatically detects when resources in a private cloud are overloaded, decides which applications can be moved to a public cloud at lowest cost, and then performs the migrations needed to dynamically expand capacity as efficiently as possible. Seagull supports both horizontally and vertically scalable applications for cloud busting. It also allows flexible policies to be specified in terms of which applications to periodically precopy into the cloud. By automating these processes, Seagull is able to respond quickly and efficiently to workload spikes, allowing the data center to be safely operated at a higher utilization level.

Our article makes several contributions:

(1) an efficient greedy heuristic that determines which applications should be moved to minimize cost, as well as an optimal ILP formulation;
(2) a precopying algorithm that decides which overload-prone applications should be proactively replicated to the cloud to enable much faster VM migrations;
(3) a prototype of Seagull using a Xen-based local data center and the Amazon EC2 cloud platform;
(4) a detailed experimental evaluation of Seagull for different applications.

Seagull supports live and non-live migration to enable cloud bursting and we show Seagull's placement algorithm can make intelligent decisions about which applications to move, lowering the cost of resolving an overloaded large scale data center by over 45% in some settings. We also demonstrate our precopying algorithm can dramatically lower the time needed to move applications into the cloud while incurring only a small cost to retain replicated states in the cloud.

2. BACKGROUND AND PROBLEM STATEMENT

This section provides intuition of the hybrid model and background information on existing cloud bursting tools. We then detail the challenges faced by a cloud bursting management system and describe the problem we seek to resolve.

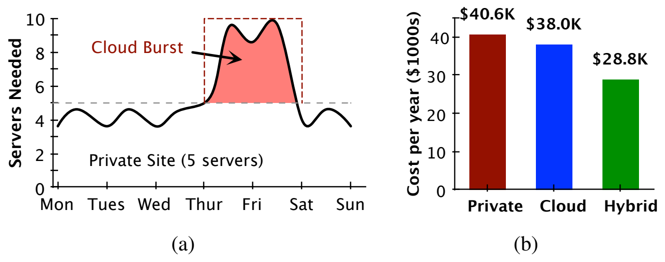


Fig. 1: Hybrid clouds can utilize cheaper private resources the majority of the time and burst to the cloud only during periods of peak demand, providing lower cost than exclusively private or public cloud based solutions.

2.1. Cloud Bursting Background

Employing cloud bursting can save enterprises a significant amount of money. Typically provisioning all resources locally implies buying sufficient servers to handle peak load. In contrast, while renting a server in the cloud is typically more expensive than buying one, the ability of to rent a smaller number of servers most of the time and adding additional servers only during peak periods can yield cost savings over locally owning servers. The hybrid approach of owning some resources locally and renting additional servers from the cloud when needed is cheaper than both approaches by extracting the best of both worlds. Figure 1 illustrates a example where a business typically requires five "extra large" servers for its daily needs, but two days a week experiences a spike up to ten servers. Using Amazon's EC2 Cost Calculator [AWSECO 2013], we find that using private resources for this would cost about $40K a year since it requires all ten servers be paid for up front. A cloud-only solution provides greater elasticity, allowing the business to pay for ten servers only during the two days a week they are needed, but the premium paid for the cloud negates much of this benefit, only lowering the cost by $2,600. However, a hybrid approach could drop the total price by a further $9,200, a saving of 27%, due to more efficient local and cloud resource utilization.

These observations have resulted in new product offerings from software vendors and cloud providers such as Amazon, VMware, and Rackspace that help businesses connect and manage "hybrid" clouds that span both private and public resources. Cloud management tools such as OpenNebula and Eucalyptus have begun to support flexible placement models where new applications can be easily deployed into either a local or public cloud.

However, these existing solutions are generally designed to move resources between private and public clouds at very coarse time scales. For example, Terremark's cloud bursting system for a government agency could take between two and ten days to fully burst from the local to cloud site [Terremark 2012]. This slowdown is caused by the large amount of application (disk) state that must be transferred over relatively slow links between a private and a public data center and any manual configuration necessary to bring up the application in the cloud. Our work seeks to enable agile cloud bursting that can respond to moderate workload spikes within hours or even minutes.

A further limitation is that existing cloud bursting products focus primarily on providing the low-level mechanisms that transfer data or enable cross data center communication. The difficult high-level policy decisions of when to invoke these migration tools, which applications to move and for how long, are still done manually by system administrators. These decisions are nontrivial, particularly in private cloud data centers with a large number of applications.

2.2. System Model and Problem Statement

Our work assumes a medium or large enterprise that has its private cloud infrastructure housed in one or more data centers. We further assume that: i) each application is virtualized and comprises one or more VMs and all VMs that make up the application must be kept together either in the

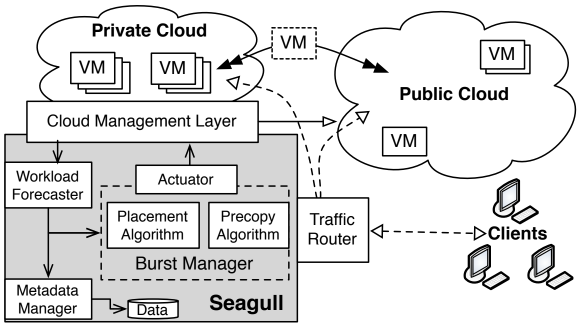


Fig. 2: Seagull architecture

private site or the cloud. ii) components within an application can be scaled either horizontally or vertically;[2] iii) each server may host one or more VMs from different applications; and iv) data centers are virtualized and are agile in terms of allocating server capacity to applications;

The goal of our work is to design a system that can both automate and optimize cloud bursting tasks. We assume that the application resource needs and constraints such as whether an application is horizontally or vertically scalable are known and so is the cloud pricing model which dictates server rental and network I/O costs. Given this knowledge, our system must answer the following questions: *(i)* When to trigger a cloud burst? *(ii)* Which applications to cloud burst so that cloud server and I/O costs are optimized? *(iii)* How to judiciously employ precopying to reduce cloud bursting latency? In what follows, we present the design and implementation of our Seagull system to address these questions.

3. SEAGULL DESIGN: BURSTING TO THE CLOUD

Figure 2 depicts the architecture of Seagull. The main components of our system include algorithms to control placement and precopying, the actuator that enacts the decisions of these algorithms, and the cloud management layer which translates generic Seagull orders into data center or cloud specific commands. At the heart of our system are its intelligent bursting and opportunistic precopying algorithms that we describe in this section.

The decision of when to trigger a cloud burst involves monitoring one or more metrics and using a threshold on them. Depending on the scenario, Seagull can use system-level metrics (such as CPU utilization, disk/network utilization or memory page fault rate) or application-level metrics such as response time. We assume that the system administrator makes a one-time decision on which metrics are relevant and the corresponding thresholds. In case of system-level metrics, the desired metrics can be monitored at the hypervisor-level, and for application-level metrics, we assume the presence of a monitoring system such as Ganglia that supports extensions to monitor any application metric of interest. Once an overload warning is triggered, Seagull can use either an optimal ILP based algorithm or a greedy heuristic to decide which applications to move to the cloud.

3.1. Optimal Cloud Bursting Algorithm

Seagull's cloud bursting algorithm must determine which applications to move to the public cloud when local resources are overloaded. A naïve approach is to simply move the overloaded applications themselves and allocate additional resources to each such application in the public cloud. However a key premise of Seagull is that, in many scenarios it may be cheaper to move one or more

[2]In horizontal scaling, the application capacity is increased by starting new VM replicas. In vertical scaling, a VM's capacity is increased, possibly after migrating to a more powerful server. An application in Seagull may employ both scaling methods.

"smaller" applications to the public cloud and use the freed-up server resources to locally allocate additional capacity to overloaded applications. Our optimization algorithm can be formulated as a integer linear program (ILP) that "searches" through various such possibilities of which applications to move to cloud for the cheapest solution to alleviate the overload in the private cloud.

Our ILP algorithm assumes that there are N distributed applications in the private cloud. Each application i can be composed of M_i virtual machines. Let L be the number of different cloud locations, with the first being the private cloud location (in the simplest case where there is one private and one public cloud, $L = 2$; however our approach can also handle bursting to multiple public cloud providers). Let H_l be the number of hosts in location l. The resource requirement of each virtual machine is specified as a resource vector that specifies its CPU allocation (in terms of number of cores[3]), its memory needs, its disk and network bandwidth needs: $(p_{ijkl}, r_{ijkl}, d_{ijkl}, b_{ijkl})$, where subscripts indicate the j^{th} VM of i^{th} application on k^{th} host of l^{th} location. Let $(P_{lk}, R_{lk}, D_{lk}, B_{lk})$ be the cores, RAM, disk size and network interface bandwidth respectively, of k^{th} host at l^{th} location. Let $\mathcal{C}_{ijkl}$ be the cost of moving the j^{th} VM of i^{th} application to k^{th} host at l^{th} location; this cost depends on the VM's resource usage, and whether it is a local move within the current location or a wide area transfer as defined in the next section.

Let α_{ijkl} and β_{ijkl} be binary variables, such that:

$$\alpha_{ijkl} = \begin{cases} 1 \text{ if } j^{th} \text{ VM of } i^{th} \text{ app is on } k^{th} \text{ host of } l^{th} \text{ loc} \\ 0 \quad \text{otherwise} \end{cases}$$

$$\beta_{il} = \begin{cases} 1 \text{ if } i^{th} \text{ app is at the } l^{th} \text{ loc} \\ 0 \quad \text{otherwise} \end{cases}$$

The ILP-based algorithm is as follows:

$$\text{minimize} \quad \sum_{i=1}^{N} \sum_{j=1}^{M_i} \sum_{l=1}^{L} \sum_{k=1}^{H_l} \alpha_{ijk} \mathcal{C}ost_{ijkl}$$

subject to

$$\sum_{l=1}^{L} \sum_{k=1}^{H_l} \alpha_{ijkl} = 1 \qquad \forall j = 1 \ldots M_i, \forall i = 1 \ldots N \tag{1}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M_i} \alpha_{ijkl} p_{ijkl} \leq P_{lk} \qquad \forall k = 1 \ldots H_l, \forall l = 1 \ldots L \tag{2}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M_i} \alpha_{ijkl} r_{ijkl} \leq R_{lk} \qquad \forall k = 1 \ldots H_l, \forall l = 1 \ldots L \tag{3}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M_i} \alpha_{ijkl} d_{ijkl} \leq D_{lk} \qquad \forall k = 1 \ldots H_l, \forall l = 1 \ldots L \tag{4}$$

$$\sum_{i=1}^{N} \sum_{j=1}^{M_i} \alpha_{ijkl} b_{ijkl} \leq B_{lk} \qquad \forall k = 1 \ldots H_l, \forall l = 1 \ldots L \tag{5}$$

$$\sum_{j=1}^{M_i} \sum_{l=1}^{L} \sum_{k=1}^{H_l} \alpha_{ijkl} = M_i \qquad \forall i = 1 \ldots N \tag{6}$$

[3]the number of cores required on each host varies depending on the hardware of host; thus the number of cores also depends on the host k

$$(1/H_l)\sum_{j=1}^{M_i}\sum_{k=1}^{H_l}\alpha_{ijkl}p_{ijkl} \leq \beta_{il} \qquad \forall i = 1\dots N, \forall l = 1\dots L \tag{7}$$

$$\sum_{l=1}^{L}\beta_{il} = 1 \qquad \forall i = 1\dots N \tag{8}$$

$$\alpha_{ijk}, \beta_{il} \in \{0,1\} \qquad \forall i,j,l,k \tag{9}$$

Constraint (1) ensures that each VM is on a single host. Constraints (2) through (5) ensure that CPU, memory, disk and network resources, respectively, used by VMs do not exceed the host capacity, while constraint (6) ensures that all the VMs of each application are placed. Constraints (7) and (8) together ensure that all the VMs of an application stay in one location.

The objective function minimizes the cost of migration. The cost $\mathcal{C}ost_{ijkl}$ captures the cost of migrating a VM either from one location to another or from one host to another. We classify the migrations to be of two essential types, namely i) *inter cloud migration, i.e. cloud bursting*, and ii) *intra-cloud migration, i.e. local migration within the private cloud.* In either case, Seagull requires knowledge of τ, the predicted length of the overload period.

Inter cloud migration: In this work we have considered only two locations, namely private cloud and public cloud. We thus define the inter data center migration cost (i.e. cost of bursting a VM to public cloud) as the sum of i) transferring the memory and storage from private cloud to public cloud, ii) storing the data, and iii) running the VMs in public cloud. More formally, to move the j^{th} VM of i^{th} application to the k^{th} host at *remote* location l has cost:

$$\mathcal{C}ost_{ijkl} = T_{ijkl} + (R_{ijkl} * \tau) + (S_{ijkl} * months(\tau)) \tag{10}$$

$$T_{ijkl} = TS_{ijkl} + TM_{ijkl} \tag{11}$$

where T_{ijkl} stands for the network cost of transferring all the VM's data to the cloud and is represented as the sum of the dollar cost of transferring the VM's storage (i.e., TS_{ijkl}) and the memory pages (i.e., TM_{ijkl}) to the cloud. The term R_{ijkl} describes the hourly cost of running an equivalent VM instance in the the public cloud. We use the resource mapping approach [Sharma et al. 2011] to determine the corresponding instance type in Amazon. Therefore, multiplying by the burst duration τ gives the total running cost for the j^{th} VM in the public cloud. Finally, S_{ijkl} is the cost to store the VM's data in the public cloud, which is typically paid for in a minimum of monthly increments.

Intra cloud migration: Seagull accounts for the cost of migrations within a data center in order to prevent the optimization function from completely reorganizing the local site. While local migrations have minimal economic cost compared to a cloud burst, it is still desirable to minimize their impact on application performance. To do this, Seagull requires the system administrator to define a parameter, $\delta \ll 1$, that represents the relative cost of local and remote transfers. Using this, Seagull calculates the cost of a *local* migration as:

$$\mathcal{C}ost_{ijkl} = \delta * TM_{ijkl} \tag{12}$$

since local migrations only require the VM's memory to be shifted. We assume that local storage and compute resources have already been paid for upfront, and thus do not need to be included in the cost.

The optimization problem Seagull must solve is a multi-dimensional bin packing problem, and is known to be NP-hard [Coffman et al. 1997; Dowsland and Dowsland 1992]. We can use an ILP solver such as CPLEX to implement this optimization algorithm. Not surprisingly, we found that such numerical ILP solvers yield good solutions for small to medium-sized private clouds and a small number of applications, but the algorithm becomes increasingly expensive as the size of the private cloud data center and applications begins to grow. Consequently, we next present a greedy heuristic that approximates this ILP optimization but can scale to much larger settings. Seagull supports both the ILP-based algorithm as well as the greedy approximation and favors the more accurate ILP when possible, while resorting to the greedy approach for tractability in larger settings. We also compare the ILP-based approach and our greedy heuristic in our evaluation section.

3.2. An Efficient Greedy Bursting Algorithm

The intuition behind Seagull's greedy algorithm is to maximize the utilization of local resources, which are cheaper than public resources, and migrate the cheapest applications when local resources are stressed. Seagull uses the following algorithms to determine when a data center is overloaded, whether to cloud burst, and which applications to migrate.

Use local resources first when possible. We assume that the capacity increase C necessary for each overloaded application can be determined using one of the many empirical or analytical methods proposed in the literature [Urgaonkar et al. 2005; Shivam et al. 2005]. Seagull first examines if any of the local servers have sufficient idle capacity to satisfy this desired capacity C. If so, the overloaded application can be live migrated to this server (for vertical scaling) or a new VM replica can be spawned on the server (for horizontal scaling). This is the simplest scenario for addressing the overload; Seagull also supports more sophisticated local "repacking" of VMs to first free up the desired capacity C on a particular server and then move (or replicate) the application to that server. This is done in Seagull using a greedy technique that sorts all servers in decreasing order of free capacity. Starting with the first server on the list, if its idle capacity is less than C, the technique examines if one or more current VMs can be moved to a different server to free up sufficient capacity C. If so, this sequence of VM moves can address the overload. Otherwise it moves on the next server with the most idle capacity and repeats.

Move the cheapest applications first. If the overload cannot be handled locally, Seagull must choose a set of applications to burst to the cloud. The objective is to pick the cheaper option between the overloaded application and a set of other applications for cloud bursting. To do so in a cost-effective manner, the algorithm picks those applications to move that free up the most local resources relative to their cost of running in the cloud.[4]

To determine which applications should be moved, we assume the duration of the workload spike, τ, and the desired capacity C for each VM are known and may be obtained from a workload forecasting technique such as [Zhang et al. 2000] [Ranjan et al. 2002] Note that C is a vector representing the CPU, disk, network and memory capacity needs of the VM. For a VM j, we use $C_j = < cpu_j, disk_j, network_j, mem_j >$ as a representation.

The VMs in the overloaded application are considered in decreasing order of their resource requirements. For each of these VMs, Seagull considers the potential hosts in the local data center sorted by their free capacity in descending order. This approach is biased towards utilizing the free capacity in the local data center first, potentially reducing the number of applications that needs to be moved to the cloud.

The secondary sorting criteria (tie breaker) we consider is the total cost of moving all *applications* on a host to the cloud; this includes the cost of not only the VMs on that particular host, but all other VMs that belong to those applications. This causes the hosts that run primarily low cost applications to be considered first.

We define the *cost* of bursting an application, say A, that is composed of n VMs using:

$$Cost_A = \sum_{j=1}^{n} Cost_{Ajkl} \tag{13}$$

[4]Additional administrative criteria such as security policies may also preclude some applications from being valid cloud burst targets; we assume that system administrators provide this information as a cloud bursting black list.

where $Cost_{Ajkl}$ is defined in equations (10), and k and l are used to determine the price of the required instance type of VM_i in the cloud. We sum over the cost of all n VMs of application A in Equation(13) to account for the constraint that all VMs that comprise an application be grouped together either in the local data center or on the cloud.

When hosts have been sorted in this way, the algorithm considers the first host and attempts to decide if a set of VMs on that host can be moved in order to create space for the overloaded VM. Each virtual machine, VM_j that is part of some application A, on the host is ranked based on:

$$C_j/Cost_A \tag{14}$$

where $Cost_k$ is the cost of transferring and running the full application that VM_j is part of. The C_j metric represents the resource capacity required by the virtual machine. In our current implementation, we set $C_j = num\_cores_j$, but this could be trivially extended to support multiple resources by using either an aggregate metric such as server volume as proposed in [Wood et al. 2009] or resource skew as suggested by [Xiao et al. 2013]. The VMs on the host are considered in decreasing order of this criteria, and the first k VMs are selected such that the free capacity they will generate is sufficient to host the overloaded VM. The intuition behind this greedy heuristic is that it *optimizes the amount of local capacity freed per dollar*.

Each of the overloaded applications is considered for bursting systematically. When a solution is found, the total cost of moving all of the marked applications is compared to moving just the overloaded application; the cheaper option is chosen to form the cloud bursting list.

3.3. Opportunistic Precopying Algorithm

An application's state may consume tens or hundreds of gigabytes. Migrating all this data at cloud bursting time will take hours or even days, significantly reducing the agility a data center needs for varying workloads.

Seagull reduces migration time by precopying an incremental snapshot of some virtual machines' disk-state to the cloud. Seagull's precopying technique must make two important decisions: (i) *which* applications to precopy, and (ii) *how frequently* to precopy each one. The precopying algorithm takes two inputs: the Burst Capacity, B, and a Burst Deadline, D. The first term represents the expected amount of capacity that may need to be burst into the cloud while the second provides a time deadline by which that amount of capacity should be able to be transitioned from the local site to the cloud. These knobs allow the data center administrator to control the agility with which it can respond to workload fluctuations without requiring them to know specific details about individual applications. The output of the precopy selection algorithm will be a set of applications and a precopying schedule which will allow the configured workload size to be shifted to the cloud within the required interval.

Selection Strategy: Seagull decides which applications to precopy based on what it would most likely pick for cloudbursting if a workload spike arrived. To do this, Seagull runs one of the cloud bursting algorithms, described in Sec 3.1 and Sec 3.2, in an *offline* mode with an extra application added which consumes B total resources. The extra application is pinned to the local site so it cannot be selected for bursting, and ensures that the desired amount of capacity will be freed by moving other applications. The result is a list of applications that can be run in the cloud at the lowest cost; together these form the *precopying set*.

Precopying Frequency: Disk state of applications in the *precopying set* is replicated to the cloud based on a frequency strategy. In the simplest case the precopying frequency can be chosen statically, e.g., once a day or once a week. Alternatively, Seagull can analyze the write rates of the virtual disks of each application and compute the frequency that data needs to be synchronized to ensure that the remaining data can always be sent to the cloud within the Burst Deadline, D.

Precopying Cost: The economic benefit of cloud bursting comes from the fact that local resources are more economical than cloud resources if they are being fully utilized. Thus, in order for pre-

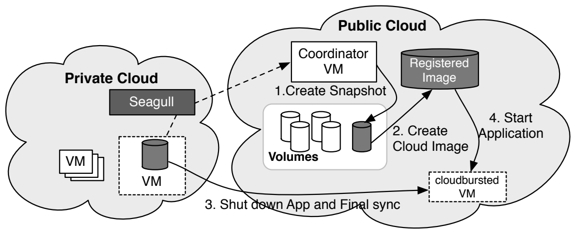


Fig. 3: Seagull Cloud Bursting Procedure

copying to be economical, it must not increase the cost by more than the premium charged for cloud resources.

In general, the cost of storage is substantially lower than executing virtual machines in the cloud (e.g., $0.10 for 1GB of storage versus $43.20 to run a small VM for a month). However, precopying also incurs network transfer costs to continuously synchronize disk state. Seagull can use the calculated schedule and disk write rates to estimate the cost of performing precopying. Seagull can then give purchasing recommendations to the data center administrator by comparing this cost to the expected price of buying additional servers for the local data center.

Cloud bursting with precopying: Both the ILP and the greedy heuristics can easily handle the case when some applications are subjected to pre-copying. Since each approach must estimate the copying overheads for bursting, we simply use the incremental cost to copy newly modified data since the last precopy as the "cost" for each precopied VM; the cost of other VMs is the full copy cost. Thus, both approaches will automatically favor precopied VMs whenever possible.

4. CLOUD BURSTING: HOW TO BURST?

Once Seagull determines a plan for which applications to move to the cloud, the *Actuator* must execute this plan. Bursting an application involves copying its virtual disk image to the cloud and starting a VM that contains all the data. In practice, current cloud platforms require several additional steps to prepare a disk image for booting within the cloud, thus Seagull uses the following procedure, shown in Figure 3, to "burst" a VM to the cloud:

(1) *Create a Snapshot:* Seagull creates a snapshot of the VM's file system as the VM is running and transfers the snapshot to the cloud to create a new disk image.
(2) *Create a Replicable Public Cloud Image:* Seagull then takes the image and transforms it to the public cloud's usable image format. After that Seagull then registers the image with the public cloud management system and boots the VM.
(3) *Synchronize Image:* Seagull next shuts down the application in local data center to synchronize any file system changes, that have occurred since the snapshot was taken, with the new VM.
(4) *Start Application and Redirect Traffic:* Seagull restarts the application in the cloud VM and redirects the application workload accordingly.

We implemented these procedures for Amazon EC2 in Seagull. This approach is designed to minimize the amount of downtime incurred during a migration; We will not stop the application until step 3. Seagull also supports precopying by periodically copying the delta difference between local application and the cloud VM, reducing the data that must be sent in step 1.

4.1. Supporting Live Cloud Bursting

Since today's public cloud platforms such as EC2 and Azure do not support live VM migration, Seagull's cloud bursting mechanisms employ migration strategies that involve VM and application downtimes. However, if live migration were to be supported by public cloud platforms, Seagull's cloud bursting mechanisms could be easily adapted to take advantage of such features. For instance, we recently proposed the CloudNet system to support live migration of VMs over a WAN from one data center to another [Wood et al. 2011]. Seagull can easily support live cloud bursting by employing our CloudNet system to live migrate virtual machines from a private cloud to a public cloud over a WAN. CloudNet employs VPN and VPLS protocols to enable transparent migration of a VM's IP address from one WAN location to another and CloudNet uses several optimization techniques such as content-based redundancy elimination and block deltas to efficiently transfer memory (and disk) state of the VM over slow WAN links, all of which can be directly employed by Seagull.

In Section 6.2.3, we experimentally demonstrate how Seagull can employ such WAN migration mechanisms from CloudNet to support live cloud bursting from a private cloud to a public cloud site.

4.2. Reverse Cloud Bursting

After the workload surge passes, it is important to bring the applications back into the private data center to save cost. We refer to such procedure as *reverse migration*.

To avoid migration oscillations between private and public cloud and also minimize the cloud costs, we make careful decisions about i) When to perform the reverse migration? ii) and who to bring back to the local data center? We actively monitor the local data center's resource utilization and trigger the reverse migration when the utilization falls below a threshold for a time window. We can adjust the parameters to control the frequency of reverse migration. We employ the algorithm in Sec 3 but attempts to choose the *most expensive* application possible back to the private cloud by sorting the cost in Equation(13) in descending order.

After determining the VM to bring back to the local data center, a similar procedure described in Sec 4 is performed to move back the VM disk state and to restart it locally.

5. SYSTEM OVERVIEW AND IMPLEMENTATION

This section describes Seagull's five main components shown in Fig 2.

Cloud Management Layer: We extended OpenNebula [Opennebula 2012] to implement the *Cloud Management Layer* in our prototype. This layer exposes the mechanism of cloud bursting, precopying, managing and monitoring VMs' resources across private and public cloud as XML-RPC API. Also, it offers a common abstract interface to the public cloud for all the other functional blocks of Seagull and adapts accordingly based on different destination clouds.

Resource Monitor: We have extended OpenNebula to support sophisticated monitoring capabilities like that of EC2 cloudwatch. Our monitoring engine is implemented using the Ganglia monitoring system. Ganglia consists of a monitoring agent (gmond), which runs inside each host machine and VMs, and a gathering daemon (gmetad), that aggregates monitoring statistics from multiple monitoring/gathering daemons. Each VM image used by applications is pre-configured with a monitoring agent; thus, when new virtual machines are dynamically deployed, the Ganglia system automatically recognizes new servers and begins to monitor them. When cloud bursting happens, we tune the monitoring agents to report data according to the destination cloud setting, e.g., using EC2's cloudwatch service.

Metadata Manager: In our prototype, we have implemented the *Metadata Manager* as python classes for each application, which store their data, e.g., VMs and network configurations, in the backend MySQL database. This offers the functionality of safe retrieval and updating for application metadata.

Workload Forecaster: The workload analyzer uses the workload statistics to estimate future workloads. It obtains the application resource list and the workload statistics from both *Metadata Manager* and the *Cloud Management Layer*. It then coalesces the information to generate application level workload data from which a *forecaster* derives the future application workload. Seagull focuses on "*what* to migrate *where*", so the question of *when* to migrate is orthogonal to our main goals. The design of Seagull is generic and any time series based forecaster [Hellerstein et al.

1999],[Urgaonkar et al. 2008], conforming to the interface, can be used. In our current implementation, we have implemented an ARIMA *forecaster*. The ARIMA *forecaster* obtains a time-series of workload observations from the monitoring engine and models it as an ARIMA time-series. The forecaster detects when the aggregate workload of the private site's applications will exceed the available capacity. It then analyzes how long this workload peak is expected to last in order to calculate τ. Since cloud platforms typically charge by the hour, these predictions can be fairly coarse grain. Our evaluation illustrates how the accuracy of the forecaster affects both precopying cost and burst time. The results suggest that a conservative predictor that over predicts demand can cause a small cost increase, but that this prevents the substantial increase in burst time which can occur if the forecaster under predicts the workload.

Burst Manager: This is the core of Seagull that must i) find applications to move to public cloud and to bring back into the private cloud and ii) find applications to precopy and schedule their periodic precopying. Using the workload predictions and the application metadata, it runs the placement algorithm outlined in Sec 3. The output of the placement algorithm is a list of applications and their final destinations. If precopying is enabled, it also computes a list of applications to precopy, using the precopying algorithm outlined in Sec 3.3, and creates a precopying schedule.

Actuator: It takes the list of candidate VMs from *Burst Manager* for cloud bursting and precopying; by calls the API exposed by *Cloud Management Layer*. The *actuator* has two components:

Migrator: The migrator is implemented as python wrapper around the basic VM operations and supports both *live* and *non-live* migrations between private and public clouds.

Precopier: We have implemented a filesystem level precopier using *rsync* to replicate selected VMs between local and public clouds. Our *precopier* supports both live and non-live precopying. In the live precopying, Seagull's *precopier* could synchronize the VM image data while it is still serving workload. Similarily, *precopier* can also take care of the data copying task while a VM is shut down. *Precopier* has two phases 1) *Initial Copy*: In this phase the *precopier* executes the following control sequences: i) creates a storage volume on public cloud and attaches it to the Coordinator VM, ii) *rsync* the local VM data to the remote volume, iii) update the *Metadata Manager*. 2) *Subsequent Copy*: In this phases, the precopier performs the step (ii) of the *Initial Copy*.

6. EXPERIMENTAL SETUP AND EVALUATION

In this section we describe the experimental setup used for evaluating the performance of Seagull. We have created a private cloud environment on a lab cluster using OpenNebula [Opennebula 2012]; for public cloud we have used Amazon EC2. We conducted experiments to illustrate the effectiveness of our algorithms and the intuition behind them. We also present an analysis of the costs and benefits when moving applications to the cloud and using precopying.

Private Cloud: We have created a private cloud environment using two types of servers: 8-core 2GHz AMD Opteron 2350 servers and 4-core 2.4 GHz Intel Xeon X3220 systems. All machines run Xen 3.3 and Linux 2.6.18 (64bit kernel). We deployed OpenNebula on these machines to create a private cloud and manage a total of 44 cores distributed across 9 physical hosts. Our private cloud supports small, medium and large servers, comprising 1, 2 and 4 cores, respectively.

Public Cloud: We have used Amazon EC2 as the public cloud in our experiments. EC2 offers two type of storage solutions for their instances, i.e., S3 and EBS. We have used the latter for our experiments, primarily to simplify the implementation of data replication across cloud boundaries.

6.1. Application appliances

We use three applications, TPC-W, Wikibooks and CloudStone for our evaluation. We have created private-cloud as well as public-cloud appliances for each of these three applications and their respective client applications. An appliance instance will create the virtual machine(s) which house the complete application. We warm up each application, using its clients, for two minutes before collecting data.

TPC-W is a multi-tier transactional web benchmark that represents an online bookstore [ObjectWeb]. We use the Java implementation of TPC-W which has two-tiers: a Web server tier based

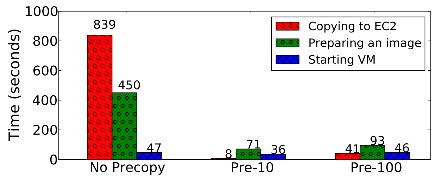


Fig. 4: Application size impact on cloud bursting.

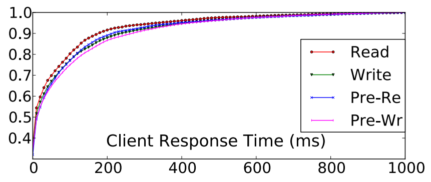


Fig. 5: CDF of client response time with heavy workload.

on Apache Tomcat servlet container and a database tier based on MySQL. In our appliance we have deployed both the tiers on the same VM.

Wikibooks is an open-content textbooks collection website for which an http replay tool has been developed [Cecchet et al. 2011]. It uses a MySQL database with a front-end PHP application. We have created two separate VMs for this application, one containing Wikimedia software and the other hosting the database.

CloudStone is a multi-platform, multi-language benchmark, which represents Web 2.0 applications in a Cloud Computing environment [Sobel et al. 2008]. It implements a social online calendar as an AJAX application, called Olio. Olio uses MySQL as the backend database and supports a memcached tier which can be horizontally scaled. We use CloudStone both in a single VM deployment and in a multi-node, replicated setup. We again use the http replay tool as a workload generator.

6.2. Migration and Precopying Tools

This section evaluates the tools used by Seagull to burst applications to the cloud and perform precopying.

6.2.1. Burst Operation Time Costs. We analyze the amount of time needed for each of the steps to burst an application to the cloud with and without precopy. The total cloud bursting time can be decomposed into three major parts: copying data to the cloud, preparing an application image and booting up the VM. We migrate CloudStone VM with a disk-state size of 5GB.

As shown in Figure 4, the total time to migrate an application with even a very small 5GB disk state, is 1336 secs; this clearly illustrates the need for precopying in enterprise applications that may have ten or more times as much state. We then precopy the application and reduce the delta (i.e. difference between the original and precopied snapshot) to 10MB or 100MB; the total time to burst the application significantly reduces to 115 secs for a delta of 10 MB. In our many trials, we notice that boot time stays almost constant and the image preparation time flatten around 70 secs which makes the data copying our primary concern and justifies the precopying approach.

6.2.2. Performance Impact of Precopying. To measure the impact of precopying on application's performance, we run a TPC-W application and continuously precopy its data to the cloud during a thirty minute measurement interval. While the replication process is running, we measure the response time observed by the TPC-W clients running a "shopping" workload. We repeat this experiment ten times and report average statistics across these runs.

The response time performance for light (100 clients) and heavy workloads (600 clients) is shown in Table I. When there is only a light workload, the average response time of all request types only increases by 2ms. When the workload is at peak capacity, the average response time of all requests increased by 19%, but write requests observed a 37% increase, which is mostly caused by a small number of outliers. As we observe in Figure 5, 90% of all requests see only a small performance change of less than 17%.

Table I: Average client response time (ms) comparison for TPC-W in Shopping Mode

TPC-W workload	READ		WRITE		ALL	
	None	Precopy	None	Precopy	None	Precopy
Shopping Light	29	31	21	25	28	30
Shopping Heavy	117	131	120	190	118	140

6.2.3. Migration Downtime. We next study the application downtime when migrating applications over the WAN with Seagull. Our current implementation relies on non-live migration of VMs for lack of live migration supports in existing clouds. However, we can test what the performance of live migration might be by running a process on the cloud platform that receives a stream of Xen migration data from a private data center.

Our experiment migrates a VM running TPC-W benchmark application, which is being accessed by a set of 200 clients running an "ordering" workload. The VM is configured with 1.7GB of RAM and a 5GB disk. When using non-live migration, the application is inaccessible during the shutdown process at the origin site (1.2 secs), for the final copy of data to the cloud (7.0 secs), and while the application is reinitialized in the cloud VM (1.2 secs). In total, Seagull's non-live migration incurs 9.4 seconds of downtime. Note that a naïve approach of VM migration could increase this to a minute or more if the cloud VM was not booted until after the origin VM was shutdown. In comparison, running a live migration to EC2 incurs only 0.978 seconds of downtime while copying the virtual machine's memory. Using live migration does cause a slight increase in cost since more data must be sent; in total, the memory migration added 1.84GB of data transfer and required 236 seconds to run.

Conclusion: Precopying has only a modest impact on response time, but can dramatically reduce the total time required to burst an application to the cloud and reduces the downtime to less than 10 seconds, even with non-live migration.

6.3. Placement Algorithms

In this section we examine the algorithm used by Seagull to decide which applications to burst.

6.3.1. Placement Decisions. We first analyze the placement efficiency of Seagull compared to a naïve algorithm in a small scenario that demonstrates the intuition behind Seagull's decision making. We show that when a hotspot occurs, Seagull is able to make better use of local resources as well as pick cheaper applications to move to the cloud.

We use three hosts of 6 cores, each hosting two applications, and three types of applications: TPC-W(A, D) Wikibooks (B, E), and CloudStone (C, F). Each application is running inside a single VM and we treat all applications in a scale up style. The initial arrangement of applications and the number of cores dedicated to each is shown under t_0 in Figure 6. To simplify the scenario, we assume that all applications have identical storage requirements.

We change application A's workload every hour (marked by instants t_i, where $i = 1 \dots 3$) such that its CPU requirement increases to 4 cores, then 6 cores, before falling back to four cores at t_3. To eliminate the impact of prediction errors in this experiment we assume a perfect forecaster.

Local Reshuffling: When Seagull detects the first upcoming workload spike at t_1, it attempts to resolve the hotspot by repacking the local machines, shifting application C to h_2 and then moving A to h_1 at effectively no cost. In the naïve solution, application A is cloud burst to EC2 directly without considering local reshuffling.

Bursting A Better Application: In the workload's second phase, Seagull migrates a cheaper application, D, to EC2 since the local data center could not provide enough capacity needed for A. On the other hand, the naïve algorithm had already moved A to the cloud, so it simply allocates extra resources to it making it more expensive.

Reverse Migration: Eventually, the workload spike for application A passes, Seagull migrates D back to the local data center while naïve algorithm, lacking the ability to perform local reshuffling, still needs to keep A in the cloud, consuming more money.

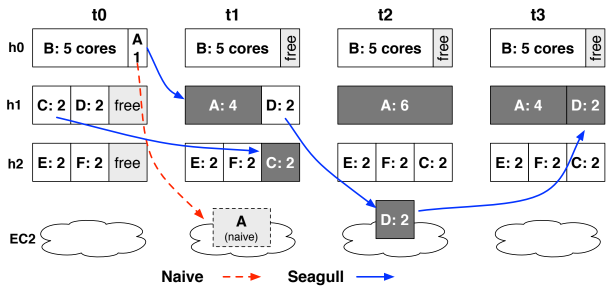


Fig. 6: The naïve approach uses only one migration, immediately moving A from h_0 to the cloud. Seagull initially avoids any cloud costs by rebalancing locally, and is able to move back from the cloud sooner than the naïve approach.

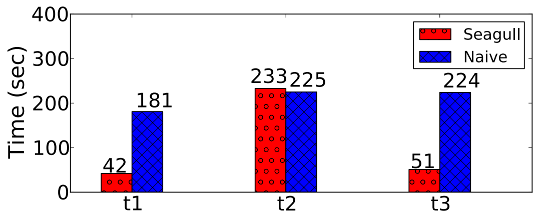


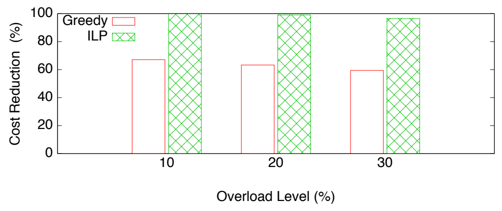
Fig. 7: Seagull uses local, live migrations at t_1, and benefits from reverse pre-copying at t_3, substantially reducing the time spent at each stage compared to naïvely cloud bursting at t_1 and restarting instances at t_2 and t_3.

Time and Monetary Cost: The use of local resources in Seagull allows it to respond to overload faster than the naïve approach. Figure 7 shows the amount of time spent by each approach to resolve the hotspots at each measurement interval; note that for both systems we precopy all applications once to the cloud before the experiment begins. Seagull is substantially faster because it uses only a local, live migration at t_1 whereas the naïve approach requires a full cloud burst. Subsequent actions performed by Naïve also incur substantial downtime since VMs must be rebooted in the cloud to adjust their instance type to obtain more cores. Seagull's migration back from the cloud at t_3 is also quite fast because it does not require the full image registration process needed for moving into the cloud. Most importantly, the fact that Seagull only requires a virtual machine in the cloud for the hour starting at t_2 means that it pays 30% less in cloud data transfer and instance running costs.

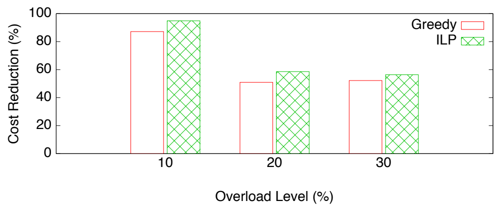
Conclusion: This experiment illustrates the intuition behind Seagull's placement algorithm. To find more capacity while minimizing the infrastructure cost and transition time, Seagull first tries to find free resources locally; if cloud resources are needed, then it moves the cheapest application possible to the public cloud.

6.3.2. Cost Efficiency. We compared the costs of our placement algorithm with both the naïve and ILP approach. We simulate a data center comprising 100 quad-core hosts and test these strategies using three types of applications randomly selected from Table II. We fill the private data center to approximately 70% of its capacity and then compute the cost of the cloud bursting decisions made while varying the percentage of applications (i.e. from 10% to 30%) to be overloaded. Both Figure 8(a) and Figure 8(b) presents the performance of these three algorithms averaged over 30 trials.

Conclusion: In Figure 8(a), our ILP algorithm achieves as low as zero monetary cost because it can perform an optimal bin packing of the local data center, which the heuristic-based algorithms are unable to do. This shows the importance of utilizing local resources before resorting to cloud



(a) With local reshuffling



(b) Without local reshuffling

Fig. 8: Comparison of cost reduction percent of cloudbursting with ILP solution compared to Naive approach

Table II: Application Details

App-Type	Reps	Active Disk Size	Image Size	Write Rate
Small	1-2	3-4GB	10G	1MB/min
Medium	3-6	6-8GB	15GB	2MB/min
Large	7-12	12-16GB	20GB	3MB/min

bursting. However, Seagull's greedy algorithm still lowers the cost by at least half compared to the naïve approach. Figure 8(b) shows that if we do not consider local reshuffling, the ILP algorithm provides only a small improvement beyond what the greedy algorithm offers. This suggests that maximizing the load displaced per dollars is an effective heuristic solution to this problem. Note that since the data center setups in Figure 8(a) and Figure 8(b) are not identical, their cost results cannot be directly compared.

6.4. Precopying Algorithm

In this section, we study the efficiency of Seagull's precopying algorithm provided different workload accuracies.

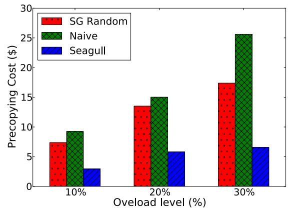
We compare Seagull's precopying algorithm to two alternatives. *Random Precopying:* randomly selects a set of applications for precopying. The number of applications is determined based on the maximum expected overload in the data center. *Naïve Precopying:* selects the set of applications that is predicted to become overloaded for precopying.

6.4.1. Perfect Workload Forecaster. This experiment evaluates the effectiveness of Seagull's intelligent precopy strategy compared to the random (*SG-random*) and naïve precopy strategies at a larger scale. We simulate a data center comprised of 200 quad-core hosts and test the strategies using three types of applications, as defined in Table II. To eliminate the impact of Seagull's local reshuffling on precopying efficiency, we assume that the data center runs only scale out applications, preventing the need for local reconsolidation. For computing the cost of each of these precopying strategies we use Seagull's placement algorithm.

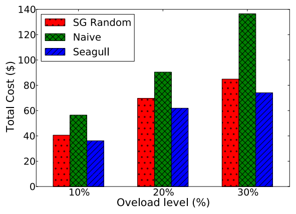
We use the perfect workload forecaster with a 24 hour horizon, and we perform precopying every hour. We study the decisions made when the level of overload in the data center increases from 10 to 30 percent. Figure 9 presents the average performance of these three strategies when the simulation is repeated 40 times for each level of overload.

In Figure 9(a), Seagull achieves the lowest precopying cost across all the overload levels. The benefits of Seagull increase with rising overload levels, and it is able to lower precopying costs by up to 75%. The naïve approach shows the highest cost because there are often applications which can be precopied more cheaply than those which are expected to become overloaded.

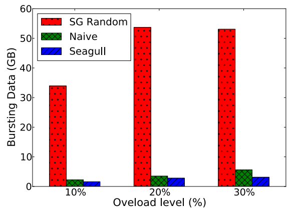
Figure 9(b) shows the total cost including both precopying and cloud bursting. Seagull reduces the cost by 45% compared to the naïve approach because running the overloaded applications in the public cloud is more costly. SG-Random and Seagull have similar total cost because they select the same applications to burst.



(a) Precopying Costs



(b) Total Costs



(c) Total Data

Fig. 9: Intelligent precopying reduces total cost and data transferred by over 45% compared to the naïve algorithm.

Figure 9(c) shows our intelligent precopying strategy outperforms SG-Random in saving data that need to be transferred because it has a poor chance of precopying the applications which will be selected by the placement algorithm. The naïve one behaves well because it selects the overloaded applications for precopying thereby substantially reducing the actual amount of data transferred during bursting.

6.4.2. With Prediction Errors. We next evaluate the workload predictor's error on Seagull's precopying strategy. In our experiment, we introduced three different types of prediction error, i.e., overestimate, underestimate and a mix of those two, when predicting the data center overloading.

We use the same setup as the previous experiment. However, during the precopy stage, the workload forecaster gives inaccurate predictions on the future resource requirements of all applications. We consider three different types of prediction error, i.e., overestimating demand, underestimating demand, and a mix of those two. We vary the accuracy of the predictor, but keep the actual overload at the time of the cloud burst to be 20%. We repeat the simulation 40 times for each error level.

Figure 10(a) shows the impact of different prediction errors on the monetary cost of precopying. When overestimating the overload scenario, Seagull thinks more applications will need to be burst to the cloud, so it precopies more data than necessary. This raises the cost of precopying by 26% to 147% relative to a perfect predictor as the prediction errors grow from 10% to 50%. On the other hand, underestimating reduces the precopying cost by 25% to 83% when the prediction errors increase from 10% to 50%, since Seagull expects fewer applications to be moved.

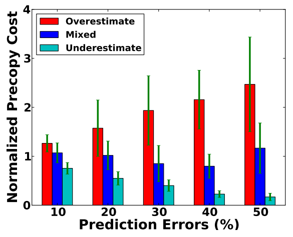
Figure 10(b) shows the impacts of different predictions on total monetary costs. Since precopying is a small fraction of the total cloud burst cost, the relative cost only rises by 14% in the worst case.

Figure 10(c) shows the amount of data that must be transferred to perform the cloud burst once the 20% overloading happens. We need to transfer up to 20.6 data during the cloud bursting when underestimating the overload level while only 1.12 when overestimating.

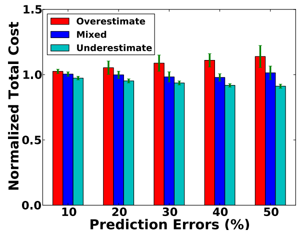
Conclusion:In all, when overestimating the overloading, we are paying more for precopying while reducing the amount of burst data. When underestimating, we need to burst much more data in exchange for a smaller precopying bill. In practice, we expect a good workload predictor will have less than 10% error.

6.4.3. Precopying Cost . Our evaluation has demonstrated that precopying can significantly reduce burst time with minimal impact on application performance, however, we must also consider the monetary cost added by precopying. To study this, we consider the results of simulation similar to that described in the previous section. We cause 30% of the system to become overloaded and compare Seagull with and without precopying.

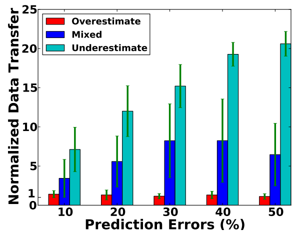
In Figure 11, it shows a modest 22% increase in cost and a substantial 95% saving in data transfer by using precopying. Since the migration time largely depends on the amount of data needed to transfer given the bandwidth, we conclude that our intelligent precopying strategy provides a reasonable tradeoff between cost and migration time.



(a) Precopying Costs

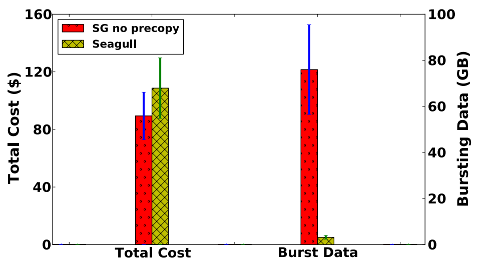


(b) Total Costs



(c) Total Data

Fig. 10: Three types of prediction errors' impact on the monetary cost and amount of burst data compared to perfect prediction.



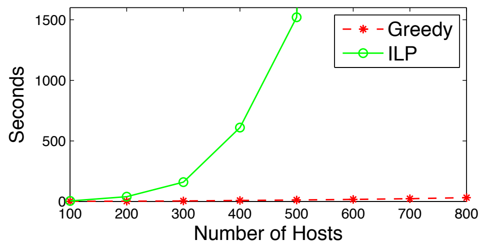


Fig. 11: Precopying causes a marginal increase in cost, but a dramatic reduction in burst time.

Fig. 12: Scalability of the algorithms.

Conclusion: Seagull saves up to 45% in total cost as compared to other precopying strategies while saving up to 95% in data transfer cost when compared to cloud bursting without precopying.

6.5. System Scalability

To test the scalability of each algorithm, we executed Seagull's heuristic placement algorithm as well as the ILP solver to obtain the solution on simulated data centers with variable numbers of hosts and applications (assuming that each application is completely packed in a single virtual machine). We again considered three types of applications as outlined in Table II. We increased the complexity of the application selection problem by increasing the number of hosts and proportionally the number of applications. We then measure the running time of the algorithms before they report the final solution. The results are plotted in Figure 12. As the size of the problem increases, the ILP solution becomes impractical. Seagull relies on its greedy heuristic to make its placement decisions. While this leads to a non-optimal solution, it makes this multi-resource bin-packing problem more tractable.

Seagull is able to process data centers of 800 hosts within 30 secs as compared to 7678 secs taken by the ILP solver. While large data centers may have many more hosts and virtual machines than this, we believe that Seagull can produce a solution within reasonable time.

6.6. Multiple Overload Scaling

In this section, we show how Seagull deals with simultaneous overload from multiple applications that require both scale up and scale out strategies. We deploy seven web applications across the five hosts in our local cluster as shown in Figure 13(a). Two of the applications, B and C, are VMs running the CloudStone application, which supports replication of its memcached layer and vertical scaling of other nodes. The other applications are scale up, and run different instances of TPC-W.

The full details of each application's characteristics are shown in Figure 13(c). At time $t = 12$ minutes, both applications A and B become overloaded for a period of ten minutes; later at $t = 70$ minutes, application B sees a second spike that lasts until the end of the experiment.

Seagull uses the workload forecast information to guide its precopying strategy, and immediately begins precopying Application A and F with a ten minute interval once the experiment starts. Throughout the experiment, we use xentop to gather the CPU utilization of Application A and B, which is illustrated in Figure 13(b), along with annotations indicating where each is located. The periodic spikes at ten minute intervals indicate the CPU overhead of precopying.

Seagull detects the overload condition expected to begin at $t = 12$ from A and B's rising demands. It decides that bursting Applications A and F is the most efficient use of resources; since these applications have already been precopied, Seagull is able to burst both applications to the cloud within one minute. Moving A to the cloud allows it to be started with a larger VM (a 4-core instance), causing its relative CPU consumption to decrease since it now has more resources available. Application B, running the replicable Cloudstone application, is able to expand its resource consumption in the local data center by spawning a new two core VM on host 5 using the resources freed up by Application F.

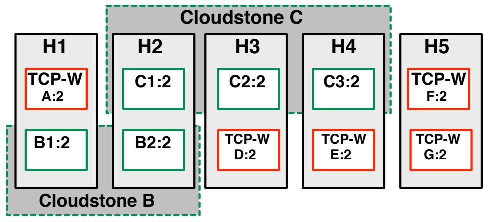
The applications continue running in this manner and the workload spike subsides. While it would be possible to immediately move the applications back to the local data center, EC2 charges in hourly increments, so there is no economic reason to do so. However, Seagull does continue to perform "backwards precopying" to replicate the state changes occurring to applications A and F in the cloud to the local data center. At $t = 70$ minutes, Seagull computes new placement decisions before the next hour of EC2 charges will begin. Seagull must plan for B's second workload spike, so it can only move either A or F back from the cloud. Since both are using the same cloud instance type, Seagull selects application F to move back to the private data center as it has a slightly higher cloud cost than A. At the end of the 80 minutes experiment, Application A remains in the cloud to make space for overloaded Application B.

7. RELATED WORK

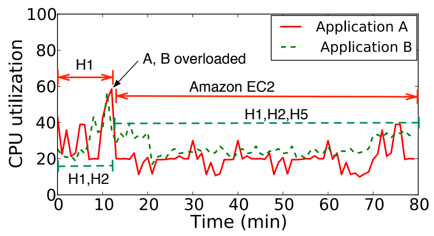
Cloud Computing covers a wide range of types of systems; in this work we focus on Infrastructure as a Service (IaaS) platforms such as Amazon's Elastic Compute Cloud. Armbrust et al. provide a survey of cloud computing in [Armbrust et al. 2009], and specifically list "scaling quickly" as one of the key opportunities in cloud computing. Our work tries to exploit the cloud's ability to rapidly obtain resources on-demand to enable rapid scale up of private data centers with minimal cost.

VM placement: VM placement inside a single data center is a well studied research area. This problem is usually formulated as a multi resource bin packing problem [Coffman et al. 1997] and different heuristic approaches [Ballani et al. 2011] [Guo et al. 2010] have been proposed to tackle this general problem in specific areas. Lee et. al. propose a VM placement algorithm with a focus on MapReduced-based jobs [Lee et al. 2010]. In SecondNet [Guo et al. 2010], the authors come up with a VM placement to guarantee fair VM bandwidth sharing. In [Rai et al. 2012], the authors design a specification language, Wrasse, which can also solve the VM placement besides other general resource allocation problems. In comparison, our work is application agnostic, focuses on VM placement across multiple data centers, and uses monetary cost of the public data centers as a metric for making the heuristic decision.

Automated resource management has been an important area of research and product development as the scale of data centers has grown beyond the control of system administrators. Research projects such as [Gulati et al. 2011; Shen et al. 2011; Xiao et al. 2013; Mishra and Sahoo 2011], automate virtual machine or storage migration to balance the CPU, memory, or I/O loads within a single data center. These approaches have been adopted by commercial products such as VMWare's Distributed Resource Scheduler [VMware DRS]. It is increasingly common for businesses and service providers to own multiple data centers, so managing resources across data centers is an increasingly important challenge [Rochwerger et al. 2011] [Buyya et al. 2010]. While our work builds on similar principles, the shift to managing applications *across* multiple data centers adds new chal-



(a)



(b)

	A	B	C	D	E	F	G
Num of VMs	1	2	3	1	1	1	1
Cores	(2)	(2,2)	(2,2,2)	(2)	(2)	(2)	(2)
Active Disk Size (GB)	(1.2)	(6.8,4.6)	(4.6,3.2,4.4)	(6.2)	(3.2)	(2.8)	(5.4)
Image Size (GB)	(10)	(12,12)	(10,10,10)	(10)	(10)	(10)	(10)
Write Rate (MB/min)	(1)	(2,2)	(1,1,1)	(1)	(1)	(1)	(1)

(c)

Fig. 13: **(a)** The initial set up of local data center. **(b)** The average CPU utilization of Application A and B over 80 minutes experiment. **(c)** Detail information of each application in the local data center.

lenges because of limited network bandwidth; Seagull mitigates these issues with its precopying system that proactively moves data in anticipation of an upcoming cloud burst.

Cloud Bursting was proposed as a way to allow enterprises who already own significant amounts of IT infrastructure to still make use of public clouds during periods of high loads [Cloudbursting 2008]. Researchers have been investigating the potential economic savings by using cloud bursting in specific domains such as medical image processing [Kim et al. 2009] and publishing [Kailasam et al. 2010]. Bicer et. al, study how a data-intensive application can be split across a hybrid cloud deployment, but their focus is on dividing data between the locations, rather than deciding how multiple applications should be placed in order to minimize cost [Bicer et al. 2011]. Hybrid clouds have become a popular service offering for hosting and data center companies, and also have been the subject of research [Mateescu et al. 2011; Sotomayor et al. 2009]. While this existing work shows the potential benefits of cloud bursting, to our knowledge there has not yet been comprehensive work on deciding what applications to run locally or in the cloud.

WAN Migration tools seek to move applications between data center sites with minimal downtime. Full VM WAN migration techniques such as [VMotion 2009; Nagin et al. 2011; Wood et al. 2011; Bradford et al. 2007] attempt to seamlessly move the memory and storage of a virtual machine, usually by building upon the existing LAN migration tools included in Xen [Clark et al. 2005] and VMWare [Nelson et al. 2005]. A recent work on Xen-blanket [Williams et al. 2012] builds a second layer on top of Xen to homogenize various cloud platforms. Their work relieves the need to worry about the VM formats of different cloud platforms during the live migration process. Alternatively, storage migration tools such as [Mashtizadeh et al. 2011; Zheng et al. 2011] only focus on moving the disk state of applications. Since current cloud platforms typically do not support live VM migration into the cloud, our work focuses only on storage migration. We use a simple rsync-based replication scheme, but we note that Seagull could easily be enhanced to use more advanced migration tools, including support for full VM live migration.

8. CONCLUSIONS

Cloud bursting is a technique to dynamically move applications running in a private data center to the public cloud to take advantage of additional resources there. In this work we propose Seagull, a cloud bursting system that automates the decision processes about which applications can be run in the cloud most efficiently. Seagull uses selective precopying to proactively replicate some applications from the private data center to the cloud, reducing the migration time of large applications by orders of magnitude. This allows Seagull to perform agile provisioning of resources across a local data center and the cloud, resulting in more efficient utilization of local resources while incurring only minimal expense in the cloud. Our evaluation shows Seagull has reasonable performance in minimizing costs compared to ILP solution and its scalability is much better. Seagull's placement algorithm considers both local re-consolidation opportunities and application cost characteristics, lowering the total cost of cloud bursting in response to data center overload by 45%. Seagull can burst applications to the cloud in under three minutes using the precopying algorithm, while incurring only minimal performance overhead.

REFERENCES

Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. 2009. *Above the Clouds: A Berkeley View of Cloud Computing*. Technical Report UCB/EECS-2009-28. EECS Department, University of California, Berkeley. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

AWSECO 2013. AWS Economics Center. http://aws.amazon.com/economics/. (2013).

Hitesh Ballani, Paolo Costa, Thomas Karagiannis, and Ant Rowstron. 2011. Towards predictable datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 conference (SIGCOMM '11)*. ACM, New York, NY, USA, 242–253.

T. Bicer, D. Chiu, and G. Agrawal. 2011. A Framework for Data-Intensive Computing with Cloud Bursting. In *2011 IEEE International Conference on Cluster Computing (CLUSTER)*. 169–177. DOI:http://dx.doi.org/10.1109/CLUSTER.2011.21

Robert Bradford, Evangelos Kotsovinos, Anja Feldmann, and Harald Schiöberg. 2007. Live wide-area migration of virtual machines including local persistent state. In *VEE*. ACM, San Diego, California, USA, 169–179. DOI:http://dx.doi.org/10.1145/1254810.1254834

Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. 2010. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *International Conference on Algorithms and Architectures for Parallel Processing*.

Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood, and Prashant Shenoy. 2011. BenchLab: An Open Testbed for Realistic Benchmarking of Web Applications. In *Proc. of 2nd USENIX Conference on Web Application Development (WebApps)*.

C. Clark, K. Fraser, S. Hand, J. G Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. 2005. Live migration of virtual machines. In *Proceedings of NSDI*.

Cloudbursting 2008. Cloudbursting - Hybrid Application Hosting. http://aws.typepad.com/aws/2008/08/cloudbursting-.html. (Aug. 2008). http://aws.typepad.com/aws/2008/08/cloudbursting-.html

E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. 1997. Approximation algorithms for NP-hard problems. Chapter Approximation algorithms for bin packing: a survey.

Kathryn A. Dowsland and William B. Dowsland. 1992. Packing problems. *European Journal of Operational Research* 56, 1 (1992), 2 – 14.

Ajay Gulati, Ganesha Shanmuganathan, Irfan Ahmad, Carl Waldspurger, and Mustafa Uysal. 2011. Pesto: online storage performance management in virtualized datacenters. In *SOCC (SOCC '11)*. ACM, New York, NY, USA, Article 19, 14 pages. DOI:http://dx.doi.org/10.1145/2038916.2038935

Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. 2010. SecondNet: a data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference (Co-NEXT '10)*. ACM, New York, NY, USA, Article 15, 12 pages.

J. Hellerstein, F. Zhang, and P. Shahabuddin. 1999. An Approach to Predictive Detection for Service Management. In *IEEE Intl. Conf. on Systems and Network Management*.

Sriram Kailasam, Nathan Gnanasambandam, Janakiram Dharanipragada, and Naveen Sharma. 2010. Optimizing Service Level Agreements for Autonomic Cloud Bursting Schedulers. In *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops (ICPPW '10)*. IEEE Computer Society, Washington, DC, USA, 285–294.

Hyunjoo Kim, Manish Parashar, David J. Foran, and Lin Yang. 2009. Investigating the use of autonomic cloudbursts for high-throughput medical image registration. In *GRID*. IEEE, 34–41.

Gunho Lee, Niraj Tolia, Parthasarathy Ranganathan, and Randy H. Katz. 2010. Topology-aware resource allocation for data-intensive workloads. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems (APSys '10)*. ACM, New York, NY, USA, 1–6.

Ali Mashtizadeh, Emré Celebi, Tal Garfinkel, and Min Cai. 2011. The design and evolution of live storage migration in VMware ESX. In *USENIX ATC*. Berkeley, CA, USA, 14–14. http://dl.acm.org/citation.cfm?id=2002181.2002195

Gabriel Mateescu, Wolfgang Gentzsch, and Calvin J. Ribbens. 2011. Hybrid Computing-Where HPC meets grid and Cloud Computing. *Future Gener. Comput. Syst.* 27 (May 2011), 440–453. Issue 5. DOI:http://dx.doi.org/10.1016/j.future.2010.11.003

M. Mishra and A. Sahoo. 2011. On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 275–282. DOI:http://dx.doi.org/10.1109/CLOUD.2011.38

Kenneth Nagin, David Hadas, Zvi Dubitzky, Alex Glikson, Irit Loy, Benny Rochwerger, and Liran Schour. 2011. Inter-cloud mobility of virtual machines. In *Annual International Conference on Systems and Storage (SYSTOR '11)*. ACM, New York, NY, USA, Article 3, 12 pages. DOI:http://dx.doi.org/10.1145/1987816.1987820

Michael Nelson, Beng-Hong Lim, and Greg Hutchins. 2005. Fast transparent migration for virtual machines. In *ATEC '05: USENIX ATC*. USENIX Association, Berkeley, CA, USA, 25.

ObjectWeb. the ObjectWeb TPC-W implementation. Website. (????). http://jmob.objectweb.org/tpcw.html.

Opennebula 2012. The Open Source Toolkit for Data Center Virtualization. (2012). http://www.opennebula.org/

Openstack 2012. openstack: Cloud Software. http://www.openstack.org. (????). http://www.openstack.org/

Anshul Rai, Ranjita Bhagwan, and Saikat Guha. 2012. Generalized resource allocation for the cloud. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*. ACM, New York, NY, USA, Article 15, 12 pages.

S. Ranjan, J. Rolia, H. Fu, and E. Knightly. 2002. QoS-driven Server Migration for Internet Data Centers. In *IWQoS*. 3–12.

Benny Rochwerger, David Breitgand, Amir Epstein, David Hadas, Irit Loy, Kenneth Nagin, Johan Tordsson, Carmelo Ragusa, Massimo Villari, Stuart Clayman, Eliezer Levy, Alessandro Maraschini, Philippe Massonet, Henar Munoz, and Giovanni Toffetti. 2011. Reservoir - When One Cloud Is Not Enough. *Computer* 44 (2011), 44–51. DOI:http://dx.doi.org/10.1109/MC.2011.64

Upendra Sharma, P. Shenoy, S. Sahu, and A. Shaikh. 2011. Kingfisher: Cost-aware elasticity in the cloud. In *INFOCOM, 2011 Proceedings IEEE*. 206–210. DOI:http://dx.doi.org/10.1109/INFCOM.2011.5935016

Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. CloudScale: elastic resource scaling for multi-tenant cloud systems *(SOCC '11)*. ACM, New York, NY, USA, Article 5, 14 pages. DOI:http://dx.doi.org/10.1145/2038916.2038921

Piyush Shivam, Adriana Iamnitchi, Aydan R. Yumerefendi, and Jeffrey S. Chase. 2005. Model-Driven Placement of Compute Tasks and Data in a Networked Utility. *ICAC* (2005). DOI:http://dx.doi.org/10.1109/ICAC.2005.41

W Sobel, S Subramanyam, A Sucharitakul, J Nguyen, H Wong, S Patil, A Fox, and D Patterson. 2008. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proc. of Cloud Computing and its Applications*.

B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. 2009. Virtual Infrastructure Management in Private and Hybrid Clouds. *Internet Computing, IEEE* 13, 5 (sept.-oct. 2009), 14 –22.

Terremark 2012. Study: USA.gov Achieves Cloud Bursting Efficiency Using Terremark Enterprise Cloud. http://terremark.com. (2012).

B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. 2005. An Analytical Model for Multi-tier Internet Services and Its Applications. In *Proceedings of the ACM Sigmetrics Conference, Banff, Canada*.

Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. 2008. Agile dynamic provisioning of multi-tier Internet applications. *ACM Trans. Auton. Adapt. Syst.* 3, Article 1 (March 2008), 39 pages. Issue 1.

VDataCenter 2012. VMware: Public & Hybrid Cloud Computing. http://www.vmware.com/solutions/cloud-computing/public-cloud/products.html. (2012).

VMotion 2009. Virtual Machine Mobility with VMware VMotion and Cisco Data Center Interconnect Technologies. http://www.cisco.com/en/US/solutions/collateral/ ns340/ns517/ns224/ns836/white_paper_c11-557822.pdf. (2009).

VMware DRS. Resource Management with VMware DRS. http://www.vmware.com/pdf/vmware_drs_wp.pdf. (????).

Dan Williams, Hani Jamjoom, and Hakim Weatherspoon. 2012. The Xen-Blanket: virtualize once, run everywhere. In *Proceedings of the 7th ACM european conference on Computer Systems (EuroSys '12)*. 113–126.

Timothy Wood, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. 2011. CloudNet : Dynamic Pooling of Cloud Resources by Live WAN Migration of Virtual Machines. In *VEE*. 121–132. DOI:http://dx.doi.org/10.1145/1952682.1952699

Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. 2009. Sandpiper: Black-Box and Gray-Box Resource Management For Virtual Machines. *Computer Networks: The International Journal of Computer and Telecommunications Networking* 53, 17 (Dec. 2009). http://portal.acm.org/citation.cfm?id=1663647.1663710

Zhen Xiao, Weijia Song, and Qi Chen. 2013. Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment. *IEEE Transactions on Parallel and Distributed Systems* 24, 6 (2013), 1107–1117. DOI:http://dx.doi.org/10.1109/TPDS.2012.283

Y. Zhang, V. Paxson, and S. Shenkar. 2000. *The Stationarity of Internet Path Properties: Routing, Loss, and Throughput*. Technical Report. AT&T Center for Internet Research at ICSI, http://www.aciri.org/.

Jie Zheng, Tze Sing Eugene Ng, and Kunwadee Sripanidkulchai. 2011. Workload-aware live storage migration for clouds. In *VEE (VEE '11)*. ACM, New York, NY, USA, 133–144. DOI:http://dx.doi.org/10.1145/1952682.1952700