Bare-metal Marketplace at the Bottom of the Cloud

Sahil Tikale • PhD Thesis Prospectus • ECE, Boston University • 2019

Overview

Today's clouds offer huge benefits in terms of on-demand elasticity, economies of scale and its pay-as-you-go model. Yet many organizations continue to host their clusters outside of cloud for security, price or performance reasons. Such organizations form a large section of the economy including financial companies, medical institutions and government agencies. Clusters are typically stood up with sufficient capacity to deal with peak demand; resulting in silos of under-utilized hardware. This situation is common place not only within an individually owned data-center running multiple clusters but also across colocation facilities like the MGHPCC[8] – a 15 megawatt data-center that hosts infrastructure from five different universities.

In the thesis we ask the question, What if we could easily and securely move infrastructure across these silos to match demand? This would obviously increase utilization and reduce the cost. Moreover, as we will see, such a capability could enable an alternative model of cloud; an Open Cloud eXchange (OCX) where multiple organizations, freely cooperate and compete with each other for offering different hardware resources while customers can choose from numerous competing services instead of a single provider.

The objective of this thesis is to build a system that allows mutually non-trusting physically deployed services to efficiently share the physical servers of a data center. The approach proposed here is to build a system composed of a set of services each fulfilling a specific functionality. The scope of this work is limited to design and implementation of only the core set of functionalities critical for sharing bare-metal servers between clusters across different deployment systems and ownership.

These functionalities are: 1. *Bare-metal Allocation and Isolation Service:* to allow different users to stand up clusters from a common pool of hardware. 2. *Diskless Rapid Provisioning Service* that allows deploying the cluster fast enough to be able to respond to rapid fluctuations in demand. 3. *Security Framework* that enables cluster owners to control trade-offs between security, price, and performance. 4. *Market based incentive system* that uses an economic model of a marketplace to ensure that resources are allocated to the cluster that needs it most. we have completed (1-3) and describe our architecture and results. Its results will be discussed in brief. Thereafter open questions regarding the (4) will be discussed followed by proposed timeline for the work pending towards completion of this thesis.

Chapter 1

Introduction

This section describes the motivation for undertaking this research. We begin with reasons that leads to formation of clusters as silos of hardware in a data-center with no way of moving servers between them. We state the aim and scope of the research, differentiate it from related work and discuss its broad implications upon how to build clouds that are truly economic and provides an environment that fosters innovation at every layer of cloud-stack.

1.1 Background and Motivation

Today's clouds offer huge benefits in terms of on-demand elasticity, economies of scale and its pay-as-you-go model. Yet, a large section of the economy including financial companies, medical institutions and government agencies continue to host their own clusters outside of the public clouds. Organizations that need total control of their hardware; have custom deployment practices; require specific security requirements and do not wish to pay for high prices of storage [14, 26]in their own hardware than renting it from public cloud providers.

Many different mechanisms are available that simplify deployment of applications and services on physical systems such as OpenStack Ironic, Canonical MaaS, Emulab, GENI, Foreman, xCat, and others [38, 15, 41, 12, 19]. Each of these tools takes control of the hardware it manages, and each provides very different higher-level abstractions. A cluster operator must thus decide between, for example, Ironic or MaaS for software deployment; and the data center operator who wants to use multiple tools is forced to statically partition the data center into silos of hardware. Moreover, it is unlikely that most data centers will be able to transition fully to using a new tool; organizations may have decades of investment in legacy tools, e.g., for deploying HPC clusters, and will be slow to adopt new tools, leading to even more hardware silos.

Also, the investment cost of setting up these clusters is high as they are deployed with extra capacity to address peak demand. By definition, short term spike is less frequent which results in an under-utilized cluster. And when faced with demand that exceeds its capacity, a cluster to suffer from degraded quality of service (QoS) or violation of service level agreements (SLAs) while enough capacity of physical servers may be available in a rack right next to it. Those servers cannot be used even if they are sitting idle simply because there is no way to move them between silos of clusters.

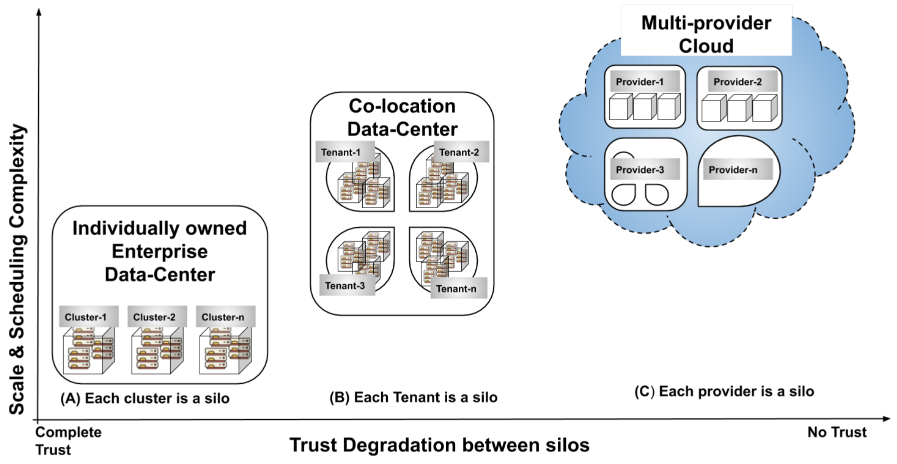
Multi-provider Cloud
Provider-1
Provider-2
Provider-3
Provider-n
Co-location Data-Center
Tenant-1
Tenant-2
Tenant-3
Tenant-n
Individually owned Enterprise Data-Center
Cluster-1
Cluster-2
Cluster-n
(A) Each cluster is a silo
(B) Each Tenant is a silo
(C) Each provider is a silo
Scale & Scheduling Complexity
Complete Trust
Trust Degradation between silos
No Trust

Figure 1·1: Type(A) Trust is highest as silos belong to a single owner. Complexity of scheduling is low. **Type(B)** silos are each tenant of a colocation data-center. Trust lower between tenants of a colocation datacenter. Scheduling complexity is higher as it includes constraints of moving hardware within tenants' owned clusters (type-A silos) and inter-tenant clusters. **Type(C)** has each provider in a multi-cloud provider as a silo. The trust is least in between providers that are have competing objectives (includes type(A) + type(B) silos). The scale of operations and scheduling complexity is highest.

The problem is sufficiently general whether it is a large enterprise hosting multiple clusters in its own data-center (type-A silos, Fig:1·1) or a colocation data-center that hosts clusters from different tenant organizations (type-B silos,Fig:1·1).

In this thesis we ask the following questions • Can we design a system such that it is possible to move hardware from one cluster to another on demand? • How can we make setting up a cluster fast enough to be able to respond to rapid fluctuations in demand? • Can we design a single system which is appropriate for a wide variety of scenarios, from multiple clusters of a single company to different tenants in the same colocation facility, to new model of cloud with multiple providers[13]? • Can we design a marketplace model, which we know is needed for a cloud that is sufficiently general to support a broad set of use cases for sharing resources?

To address this questions, the thesis propose a general architecture that we call the Elastic Secure Infrastructure or (ESI). It is a system composed of a set of micro-services that allows mutually non-trusting physically deployed services to efficiently share the physical servers of a data center. ESI requires that only a minimal functionality be trusted by everyone while rest can be deployed by each participating tenant themselves.

Broader Impact: An alternative to single provider public clouds

The bottom-most layer of any single provider public cloud is either a virtualization based black-box with no visibility into the allocation of resources or a bare-metal solution with limited deployment options and no choice but to trust the cloud provider completely for its security. Only a handful of organizations have the capital for setting up a public cloud. This seriously limits avenues for innovation because the research and academic community does not have access to the internal workings of an at-scale cloud. Moreover, every cloud provider seeks to promote its business and any of the solutions and services that they develop are increasingly incompatible with that of other clouds [35]. Even their pricing and economic models are geared towards locking the customer into their eco-system [13].

We believe that the public cloud can evolve into a marketplace in which many actors can compete and cooperate with each other, and innovation can flourish. We refer to this model as the Open Cloud eXchange model or (OCX) [13]. In addition to breaking silos ESI is a critical requirement to enable OCX as the bottom most stack of a multi-provider cloud – a single platform where stake-holders from both industry and academia together host their hardware and services to each other and to customers on pay-as-you-go basis.

The scope of this thesis is the design and implementation of the following functionalities. • *Bare-metal Allocation and Isolation Service* that uses network isolation technologies to isolate tenants' bare-metal servers • *Diskless provisioning service* that installs software on servers using network mounted storage. • *Security framework* that allows different cluster owners to attest the integrity of the physical server before choosing to add it to its cluster. • *Decentralized distribution system* that use the economic model of a marketplace to encourage different clusters owners to share their hardware with others.

We have prototype implementation of the *Bare-metal Allocation and Isolation Service*, *Diskless provisioning service* and *The Security framework*. We will discuss each one briefly in the next chapter along with results that demonstrate that it is possible to not only move hardware between silos of clusters with significantly different security requirements but it also possible to deploy it at speeds comparable to hosting a virtualized infrastructure.

1.2 Related Work

The idea of aggregating computational resources from multiple stake-holders and offer it as a single large pool is not a new one. Grid computing was one of the initial efforts at giving users access to a massive pool of computational resources that were heterogeneous, geographically distributed and were owned, controlled and operated by independent organizations [33, 24, 11]. Each stake-holder decided what fraction and granularity of resources to share with the grid [22]. Middlewares were developed that offered a single view of all available resources to the end user. The most relevant work for this thesis is Legion [21], Globus [20, 5] and Condor [28, 36]. Globus and Legion offer great insight into the process of designing system of systems that adheres to the sharing policies of the organizations providing the hardware; keeps all the complexity transparent from the end user and addresses issues common to both users and resource providers such as centralized user-authentication, dynamic resource discovery, transparent matching of job to resources and security. Traditionally grid computing solutions consisted of hardware resources like high end servers and HPC systems, Condor demonstrated that it is possible to do grid computation by harvesting idle CPU cycles from commodity class hardware like user workstations. Despite of a great deal of academic and industry interest over the years grid computing did not become as popular or mainstream as cloud computing is today.

Cloud can be considered as a grid computing model with easy user interface; owned, offered and controlled by a single provider and users pay only for the resources they use. Stupendous success of the first single provider public cloud attracted more companies to offer cloud platforms [2, 6, 9, 7]. For competitive reasons, each uses proprietary (black box) solutions for orchestration of its hardware resources. Restrained solutions for software deployment, blanket security model and opaque practices about allocating hardware resources to jobs makes it difficult for certain sections of the industry, academia and research organizations from adopting the cloud. This includes finance companies, medical institutions and research organizations that still form a considerable part of the economy. Open cloud architectures such as Eucalyptus [31, 32], OpenNebula [29], Nimbus [25] and OpenStack [1] were developed as an alternative to the single provider public cloud. Being open-source they have reduced the opacity prevalent in the single provider public clouds, still their architecture does not allow for multiple stake-holders to control and operate the hardware resources. The work of Bestavros et.al.[13] explains the need and provides the vision to have an at-scale multi-provider cloud. It proposes a fully open and fair marketplace model called the *Open Cloud eXchange* (OCX) where multiple stake-holders, instead of a single provider, compete and co-operate to offer services to the customers at every layer of the cloud stack.

Related Work for Incentive System

Dynamic partitioning of a large scale cluster and finding optimal method of resource allocation is a well known problem that has received a lot of attention from research community over the years. It is difficult to cover all the research here, hence we cover relevant work and justify why there is a need to develop a new partitioning system.

Borg: Borg [39] is a massive scale distributed scheduler that internally used at Google. It is distributed in allocation of resources but centralized when it comes to visibility and ownership of resources. The scheduler is distributed in the sense that each instance of scheduler is responsible for allocation of resources on a subset of machines called *cells*[1]. Borg controls all the hardware and all the aspects of the life-cycle of a job from installing necessary software to evicting jobs. Users have no freedom to choose the hardware or where there jobs will be placed or when will it start running. Consumers have priorities and quota that they can purchase. It is not clear how the purchases happen since it outside the system and not described. Jobs

[1]A google data-center is divided in clusters and each cluster is divided into cells each which contains few thousand machines.

are allocated resources or evicted based on their importance in the priority hierarchy and whether their resource demand matches their assigned quota.

Omega: Unlike Borg where each instance of the scheduler is responsible for allocation decision within a limited section (cells) of a cluster, Omega [34] takes a two stage shared state approach where there is one global scheduler and multiple mini-schedulers with different job allocation priorities. Each mini-scheduler has complete view of global utilization of the cluster and can use the information to place its jobs in any part of the cluster. If there is an allocation conflict where multiple schedulers claim ownership of a single resource, the global scheduler steps in to resolve it. Although it is not clear what parameters are used to resolve such conflicts. In ESI where it has no control on the number of participating clusters each with its own scheduling priorities, this can be a seriously detrimental to the scalability. With each new cluster joining ESIit may increase the ratio of conflicted resources which would be difficult to resolve especially when ESIdoes not own the hardware resources.

Mesos: [23] resolves the issue of resource conflict by offering it sequentially, one at a time, to each of its participating clusters. Based on their local demand each framework can either accept or reject the resources. Mesos follows the fair share schduling approach where each framework are allowed to scale up to their designated shared in the cluster and can grow above their designated share only if unused resources are available. Any jobs running on resources that exceed the allocated share can be pre-empted. The limitation of these approach is that the resource offering is only one way i.e. from Mesos to participating frameworks. Framework cannot proactively ask for resources, they have to wait their turn to get the resources. Also it is not clear when does Mesos decide to revoke the resources to offer to other frameworks since it has no visibility into the demands of each framework.

All of the above systems have certain freedom to make design choices like using a common orchestration system (Virtualization, Containers etc) and assuming no malicious behavior among co-located jobs from different application and regulating the relative priority of jobs.

The locus of control gets reduced when the scheduler has no control on the availability of the underlying hardware. Following section describes some work that addresses such constraints.

Boinc: It is a middleware that enable large scale distributed applications to use publicly available computational resources such a personal a home computer connected to the internet. Boinc [10] does not control either the functioning of the distributed applications (referred in the work as *projects*) or the availability of the computational resources. Individual can choose to donate idle resources (CPU, network, storage) of their personal computer to one or more projects of their choice. Boinc provides incentives in the form of credits for usage and leaderboards for maximum credits to encourage more people to offer computational resources. A serious limitation of the work is the lack of prevention against any malicious activity especially when a computer can be shared among multiple projects.

Tycoon: Unlike Boinc where the provider (computer owner) decides the ratio of resources to be distributed among the co-located applications, Tycoon [27] uses a market based auctioning system for allocating resources among competing applications. The currency is still virtual without any real monetary value but it is close to a real market based economic model in mimicking the allocation of resources per host. There is still an implicit trust between applications that allows them to share the resources on the single host. Some of the notable contribution that can be helpful to ESIis the segregation of resource allocation mechanism from strategies of bidding for resources. There is no central authority deciding on the priority of the jobs to be scheduled on the resources. The importance of any job is left to the respective users and their their willingness to pay for the resources. The auction based sharing is limited to a single host and the distribution of application across hosts is still conducted centrally. Also the virtual money exchange is only one way where the money is paid by the users to the providers. The providers have to return the money to the bank – a central entity that overlooks distribution of applications across hosts. Users get money at regular intervals from the bank to keep the model functioning.

Recent work of Duplyakin et. al [17] takes an approach where each cluster manages its local demand independently and contacts the global scheduler only when it needs more resources. There is no concept of proactively releasing the servers so that other cluster can make use of it. Instead every cluster assigns values to each of its nodes based on how much resources are occupied or free. The global scheduler uses this information to preempt nodes from one cluster to satisfy demands of other cluster. Such a system is unsuitable for our needs because its global scheduler needs to have complete control of the hardware pool to preempt the nodes (violates condition 1), needs to have all the information about the internal layout of how jobs are allocted within a given cluster and clusters have no choice but to accept the nodes offered by the global scheduler.

Chapter 2

Systems Architecture

Here we describe different use-cases that would benefit from having the ability to move hardware between clusters of physical servers. We define requirements from these use-cases that the ESI system should satisfy followed by the architecture of the system.

2.1 Characteristics of different clusters in a Datacenter

This sections describes workload characteristics, preferences and constraints of few type of clusters that will benefit from using ESI to share their resources.

2.1.1 High Performance Computing (HPC) & High Throughput Computing (HTC) Clusters

A typical HPC or HTC job is a non-interactive, batch job with very low tolerance to performance jitters (only HPC) and network latency but highly tolerant towards delays in execution. Such jobs stay queued for a long time until required infrastructure is available. Setting up these high (performance and throughput) computing clusters requires considerable investment therefore the preference is to maximize overall utilization of the cluster.

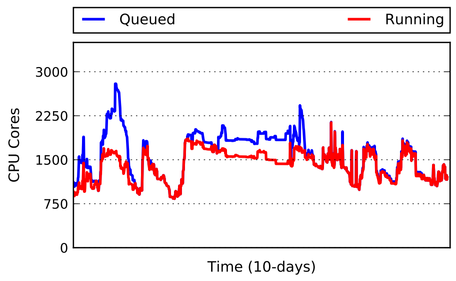

Figure 2·1: Luxembourg University (HPC)

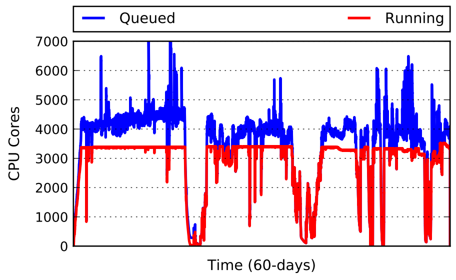

Figure 2·2: Atlas (HTC)

Figure 2·2 show usage of a HTC cluster for a period of 60 days. The red curve shows the utilization and the blue curve shows the cumulative capacity required to satisfy all the queued jobs. Clearly, these clusters can benefit if any extra capacity of physical servers is offered for even a short span of time. One major constraint is that any new server should be offered such that their in-house deployment system has complete control of it. This means that any extra servers should be made available bare-metal so that it can be seamlessly added using their own deployment methods.

2.1.2 Clouds

Applications that are best fitted to run in the cloud are the ones with high variable demand and extremely interactive in nature. Also, they are robust to performance jitters and network latency common in a virtualization based solution. The goal of a cloud is to maximize its profits. They want to be responsive to demand so they don't have to turn away customers, which directly affects their profits. Figure 2·3 show utilization curve of a commercial cloud where average life-span of a VM is around 3 days. The red curve show the cores allocated to VMs while blue and the black curve represent 95 percentile usage and average usage respectively. It is clear from the figure that the utilization follows a predictable pattern, which can be exploited to free up excess capacity and possibly re-allocate it outside the cloud to increase revenue.

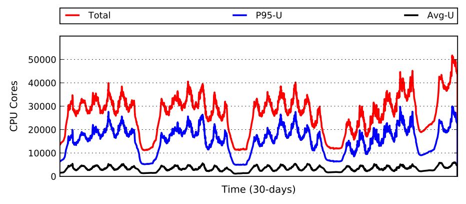

Figure 2·3: Microsoft-Azure (Cloud)

A cloud provider may be willing to offer its servers to other clusters if there is a surety that when the cloud experiences a surge in demand it can get sufficient servers from other clusters.

2.1.3 Systems Research Testbed

Scientific groups and researchers involved in low-level OS and systems research require access to raw (sometimes specialized) hardware, ability to setup custom automation for deployment of complex environments and support for reproducible experiments. Thus their constraint is to have access to the same or equivalent type of hardware. CloudLab [4] is a platform that allows researchers to build bare-metal clusters to perform OS and systems experiments at scale. Figure 3·1 shows the typical usage of one hardware type over a span of two years. As is evident from the graph, the utilization peaks when more and more research groups run experiments as the deadline for systems conferences approaches followed by a period of low utilization.

When the deadline is close and hardware is limited for experiments researchers will welcome any extra capacity from other clusters as long as they get homogeneous servers and in quantity adequate to perform their experiments. Researchers are also highly tolerant to delays in the availability of servers. If there is an assurance of getting consistent hardware of preferred configuration they may be willing to wait till weekend

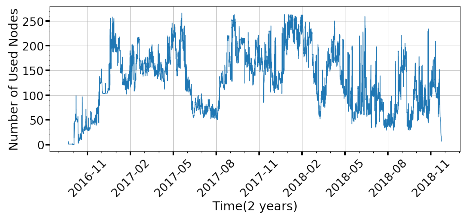

Figure 2·4: Bare-metal allocation variability over a period of few months. High demand near the systems research deadlines.

or do experiments at odd hours.

2.1.4 Government Data Centers

The defense and federal agencies have multiple data-centers ready for response in times of national emergency. Because of the rare nature of national emergencies these data-center are grossly under-utilized while incurring constant maintenance cost. Their preference is to have access to large scale compute resources in times of emergency without having to incur continuous cost of maintaining the infrastructure. According to the RFI[3] released by IARPA their constraint is to be able to scale up rapidly in times of emergency but with security equivalent to an air-gapped private enclave.

If the tenants of the co-location data-center agree to let the federal agencies use infrastructure in times of national emergency it would require all participating organizations to provide some security framework and ability to rapidly re-purpose hardware for government use. Such an arrangement will be both time sensitive and security sensitive.

2.2 Requirements

Following are the requirements derived from the use-cases that ESI needs to satisfy:

1. **Isolation & Bare-metal Multiplexing:** Each organization has developed in-house deployment methods and processes that help them to maximize usage of their resources. For a system that allows borrowing of hardware resources from one owner to another, it must provide a mechanism to provide complete control to the borrower over physical machines that includes all the capabilities they use to debug an installation when there is a failure. Also, the borrowed portion of the cluster should be isolated from the infrastructure used by other users.

2. **Fast provisioning:** Many organizations would be reluctant to offer their resources outside of their cluster if there is no surety of getting them when they need it. Especially for clusters with short term unpredictable demand and time sensitive response (see 2.1.2:clouds), the ability to have hardware ready for servicing requests rapidly is important. A mechanism that allows clusters to release under-utilized servers to other clusters or expand their cluster by borrowing servers from other organization is an important requirement if clusters with different time-sensitivity to demand are to participate and offer their resources to each other.

3. **Security:** Different organizations value security differently. A systems researcher (see: 2.1.3) may not spend as much time and effort to ensure the

security of its system as would a security sensitive organization (use-case: 2.1.4). Sharing a bare-metal node between clusters with different security requirements can be a major source of concern. To make server exchange possible between group with different standards for security we need a mechanism that allows the borrowing organization to verify whether a server matches the security standard of its cluster.

4. **Incentive System:** Building a system that provides fast and secure mechanism for sharing physical servers between organizations does not guarantee that the said sharing will occur. The use-cases are an indicative list of organizations which may not trust each other or would be otherwise in competition with each other. We need a mechanism to encourage exchange of resources where no one organization dictates how the resources are allocated. Ideally, we would want a system that allocates resources where it is needed most; provides incentives to participating clusters to actively share their under-utilized resources. We adopt a marketplace like mechanism that rewards sharing of resources between organizations is an important requirement.

2.3 Design Principles and Choice of Architecture

Among distributed systems the *Microservices Architecture* is a best fit for building such a system. It is a system composed of multiple services, each of which caters to a specific function; can scale on demand without affecting other components; has an independent life-cycle of development; and has a well defined API interface for collaborating with other services. They are termed as microservices to capture the notion that each service is a small code-base that caters to small set of functions that can be developed, maintained and improved in a short time by a small team of developers. In literature a microservice architecture is defined as *"A distributed application where all its modules are microservices."* [16]. Loose coupling and interaction by message passing via well defined APIs has the advantage of composing the services in more than one way where a microservice can be either setup to work independently or can be setup to manage other microservices [30, 16].

The focus of this thesis is limited to implementing the isolation service, the provisioning service, the security framework and the Incentive System.

1. *The Isolation Service* is a new fundamental layer that allows different physical provisioning systems to share the data center while allowing resources to move back and forth between them. We take an Exokernel-like [19] approach where the lowest layer only isolates/multiplexes the physical resources. It partitions physical hardware and connectivity, while enabling direct access to those resources to the physical provisioning systems that use them.

2. *The Provisioning Service* serves user images that contain the operating system (OS) and applications from remote-mounted boot drives. It relies on a fast and reliable distributed a storage system (CEPH [21], [22] in our implementation) for hosting images of provisioned bare-metal instances.

3. *The Security Framework* presents a new architecture for bare-metal clusters that enables tenants to control tradeoffs between security, price and performance. Security-sensitive tenants can minimize their trust in the provider and achieve similar levels of security and control that they can obtain in their private data center. At the same time, this framework does not impose overhead on tenants that are security insensitive nor compromises the flexibility or operational efficiency of the provider. It includes the attestation service and a Linux based firmware.

4. *The Incentive System* proposed here follows a market based economic model where • every tenants earns revenue for offering resources, • change in overall demand and supply is reflected by dynamic changes in the price of the resource, • auctions decide the best placement of resources among competing demands.

Microservice Architecture of a Elastic Secure Infratructure (ESI)

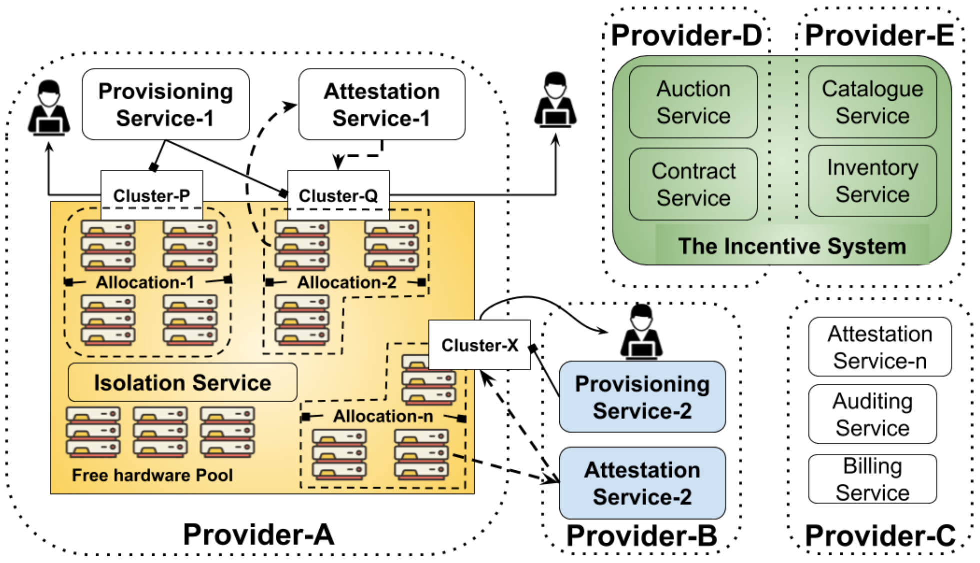

Figure 2·5: *The Isolation Service* is the only service of the provider (*provider-A*) that tenants have to trust. *Cluster-P* is setup using *Isolation service + Provisioning service-1*. *Cluster-Q* is setup using *Isolation service + Provisioning service-1 + Attestation service-1*. *Cluster-X* is setup using the *Isolation service + Provisioning Service-2 + Attestation Service-2*. Users can also use third party services offered by providers like *Provider-C*. The *incentive system* shows tentative list of services, that may be hosted by different providers, here (*Provider-D,E*).

A multi-provider elastic secure infrastructure shall be composed of microservices owned by different tenants. Figure 2·5 shows the architecture of the multi-provider

ESI. *The Isolation Service* is the only service of the provider (Fig.2·5, *provider-A*) that tenants have to trust. Tenants can choose to setup their clusters depending on their level of trust on the provider. For example, users that fully trust the provider (Fig.2·5, *Cluster-P*, used internally by *Provider-A*) may simplify their deployment process by not choosing to use the security framework or choose to depend on the provider for its security (Fig.2·5, *Cluster-Q*). Tenants that do not wish to trust the provider can either host their own services (Fig.2·5, *cluster-X* used by *provider-B*) or use third party services (Fig. 2·5, *attestation service-n* from *Provider-C*). Similarly, the *Incentive System* is proposed to be a combination of services that may be hosted by different providers (Fig.2·5, *Provider-D,E*).

Following the OCX model, a multi-provider ESI shall support multiple systems, each composed of multiple services in variety of combination of ownership. This project has involved large number of different people including students, researchers, and developers. I have co-led the design, development and prototype implementation of the isolation service and the security framework while worked as a part of the team that designed and implemented the provisioning service. I am leading the efforts towards building the incentive system.

Chapter 3

Work Completed

Here we describe in brief the design and prototype implementation of the *Bare-metal Allocation and Isolation Service* that uses network isolation technologies to isolate tenants' bare-metal servers; *Diskless provisioning service* that installs software on servers using network mounted storage and *A Security framework* that allows different cluster owners to attest the integrity of the physical server before choosing to add it to its cluster.

3.1 Hardware Isolation Layer (HIL)

Our implementation of the *Bare-metal Allocation and Isolation Service* is called the "Hardware Isolation Layer" or HIL. It decouples the functionality of resource allocation of bare-metal servers from the functionality of installing systems and application software on the servers. It takes the Exokernel-like[18] approach where the lowest layer only isolates/multiplexes the resources and richer functionality is provided by systems that run on top of it. With HIL, we partition physical hardware and connectivity, while enabling direct access to those resources to the physical provisioning systems that use them. This approach allows existing provisioning systems, including Ironic, MaaS, and Foreman, to be adapted with little or no change. Figure 3·1 provides a diagrammatic overview of how HIL can enable provisioning systems to build clusters using servers from different pools of hardware. The fundamental operations HIL provides are *1)* allocation of physical nodes, *2)* allocation of networks, and *3)* connecting these nodes and networks. In normal use a user would interact with HILto allocate nodes into a pool, create a management network between the nodes, and then connect this network to a provisioning tool such as Ironic or MaaS. As demand grows, the user can allocate additional nodes from the free pool; when demand shrinks, they may be released for other use. Developed with less than 3,000 lines of code in the

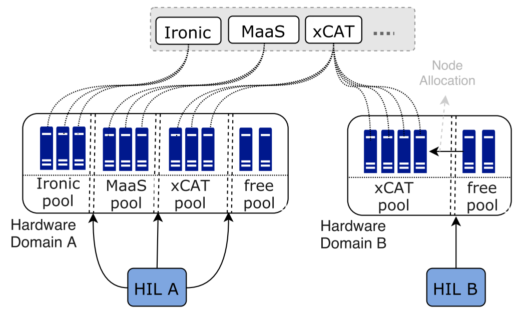

Figure 3·1: HIL provides strong network-based isolation between flexibly-allocated pools of hardware resources, enabling normally incompatible provisioning engines (e.g. Ironic, MaaS, xCAT) to manage nodes in a data center.

core functionality, this is a minimalist and the only service that has to be trusted by the organizations borrowing bare-metal nodes from owning clusters.

3.2 Bare-metal Provisioning Service (BMI)

Our implementation of *Provisioning Service* is called Bare-metal Provisioning Service or BMI. It provides the following features towards fulfilling the requirement of "fast provisioning" as discussed in section (2.2)

- *Rapid provisioning:* BMI uses diskless provisioning mechanism where the server is booted using an image mounted from a network-accessible remote logical disk that appears as a local disk to bare-metal systems. Since it does not require copying tens of gigabyte of data to the local disk or multiple boots per installation, it reduces the time it takes to stand up a ready-to-service cluster from bunch of bare-metal servers.

- *Rapid snapshotting, re-purposing, reprovisioning:* BMI architecture chooses to host the boot images from a high performance distributed storage system. Many modern distributed storage systems support capabilities such as "copy-on-write" (COW), "de-duplication" and "linked-cloning". BMI uses these abilities for quickly snapshotting the OS and applications, releasing a server when unused, and quickly provision a server using a previous snapshot in case of hardware failure. These features allows the service to offer "elasticity" to the applications it hosts.

- *Rapid cloning:* The ability to rapidly stand up a large number of servers concurrently using the same saved image is a common request in Infrastructure-as-a-Service clouds. This feature enables easy deployment of parallel/distributed applications and scalability.

It consists of five components namely 1. API server 2. Storage Service 3. iSCSI Service 4. Diskless Provisioning Service.

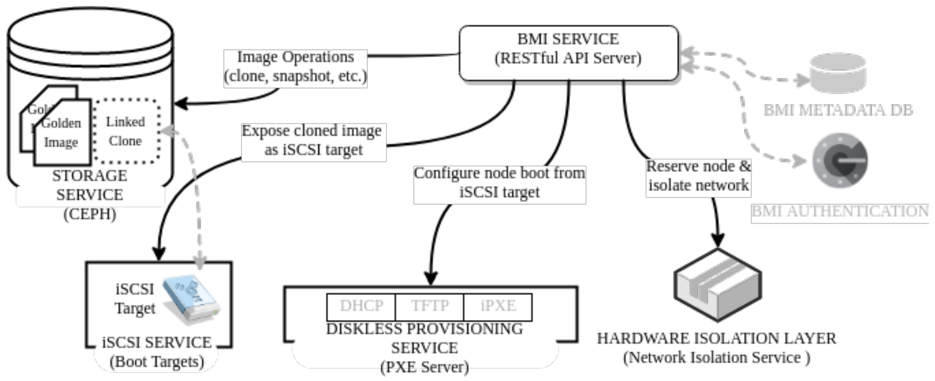

Figure 3·2: BMI components and architecture.

Figure (3·2) displays these five components. BMI follows a driver-based approach and provides an abstraction for each component. This allows systems administrators to replace the solution used for any of these components. For example, in the current implementation Ceph [40] is used as the storage service that supports COW, but it is possible to replace it with any other storage service that supports COW or equivalent features. Similarly, diskless provisioning service implementation uses the Linux SCSI Target Framework (TGT)[37] to expose the boot images as iSCSI targets which are booted using the Preboot eXecution Environment (PXE) specification. The API server is a python-based RESTful web service that controls the flow between different components of BMI. For network isolation it depends on HIL which was described in section (**??**)

3.3 Bolted - The Security Framework

Bolted is an implementation of the *security framework* that enables tenants to control tradeoffs between security, price, and performance. Security-sensitive tenants can minimize their trust in the provider and achieve similar levels of security and control that they can obtain in their own private data centers. At the same time, Boltedneither imposes overhead on tenants that are security insensitive nor compromises the flexibility or operational efficiency of the provider.

The key goals of Bolted are: (1) to minimize the trust that a tenant needs to place in the provider, (2) to enable tenants with specialized security expertise to implement the functionality themselves, and (3) to enable tenants to make their own cost/performance/security tradeoffs – in bare-metal clouds.

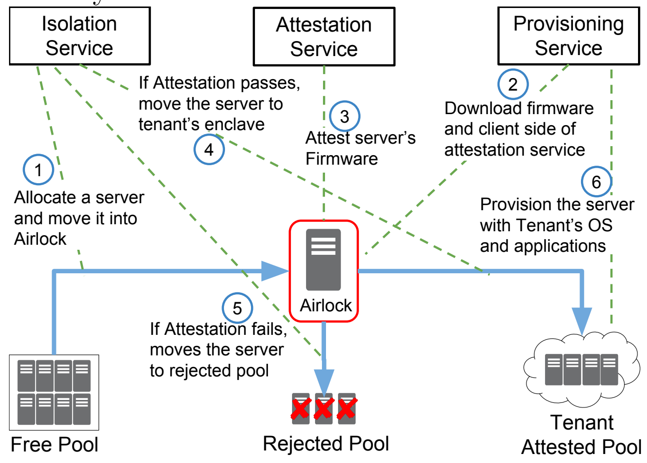
Isolation Service
Attestation Service
Provisioning Service
If Attestation passes, move the server to tenant's enclave
4
3
Attest server's Firmware
2
Download firmware and client side of attestation service
1
Allocate a server and move it into Airlock
6
Provision the server with Tenant's OS and applications
Airlock
5
If Attestation fails, moves the server to rejected pool
Free Pool
Rejected Pool
Tenant Attested Pool

Figure 3·3: Bolted's Architecture: Blue arrows show state changes and green dotted lines shows the actions during a state change.

The different Bolted components do not directly interact with each other, but instead, are orchestrated by user-controlled scripts. Figure 3·3 shows the **life-cycle of a typical secure server** (in the case of security-sensitive tenant), which progresses through six steps: (**1**) The tenant uses the Isolation Service to allocate a new bare metal server, create an airlock network, and move the server to that airlock, shared with the Attestation and the Provisioning networks; we need to isolate servers in the airlock state from other servers in the same state so that a compromised server cannot infect other un-compromised servers. (**2**) The secure firmware is executed (if stored in system flash) or provisioned onto the server along with a boot-loader, attestation software agent, and any other related software. With these in place, (**3**) the Attestation Service attests the integrity of the firmware of this server. Once initial attestation completes, (**4**) the tenant again employs the Isolation Service to move the server from the airlock network. If firmware attestation failed (**5**) it is moved into the Rejected Pool, isolated from the rest of the cloud; if attestation was successful, the server is made part of the tenant's enclave by connecting it to the tenant networks. In order to make use of the server, further provisioning is required (**6**) so the tenant again uses the Provisioning Service to install the tenant operating system and any other required applications.

3.4 Results

Here we present some results demonstrating the effectiveness of each service in moving resources between mutually non-trusting bare-metal clusters with different security requirements.

Hardware Isolation Layer

Figure 3·4 shows the performance of synchronous HIL API operations as we scale the number of concurrent clients from 1 to 16, while making requests in a tight loop.

As expected, operations that primarily make use of the DB, like allocating or deallocating a project, node or network, complete in less than a tenth of second even with 16 concurrent clients. Freeing a node takes about 5x the time of the other allocation-related operations and degrades more rapidly with increased concurrency because of a call out to ipmitool to ensure that any consoles connected to the node are released before it is de-allocated.

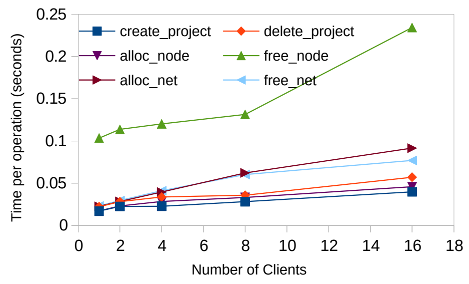

Figure 3·4: Scalability of HIL synchronous operations

Bare-metal Isolation Service

Figure 3·5 presents the time comparison of BMI with Foreman, and OpenStack/Ironic, two widely used provisioning systems, when we provision a single bare-metal server in our first environment with a bare RHEL 7.1 operating system. As seen in the figure, the BMI provisioning time is around five minutes. Note that firmware initialization of these bare-metal servers requires more than three minutes; hence half of the BMI provisioning time is spent in firmware initialization. Both Foreman, and OpenStack/Ironic have to go through firmware initialization phase twice.

Furthermore, they have to install or network-transfer the OS to a local disk, whereas BMI simply provisions the server out of a remote disk containing the operating system. Due to these advantages, BMI provisions servers around three times faster than both Foreman and OpenStack/Ironic[1]. Also, with BMIthe time to re-provision the new server is significantly reduced as there is no requirement to re-install the operating system or any Hadoop packages. As shown in the last bar of Figure 3·5, BMI reduces the re-provisioning time of the servers by up to 5 times as compared to Foreman or Ironic.

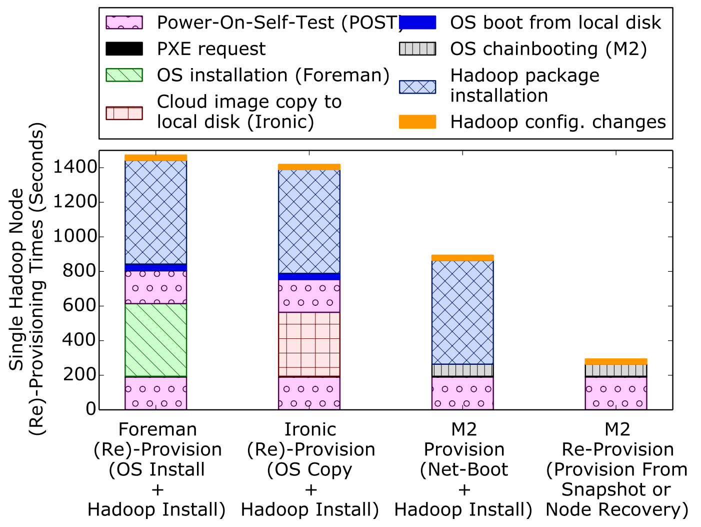

Figure 3·5: Single Hadoop compute server (re)-provisioning time comparison between BMI , Ironic and Foreman.

Bolted - The Security Framework

To understand the elasticity Bolted supports, we first examine its performance for provisioning servers under different assumptions of security. Figure 3·6 compares the

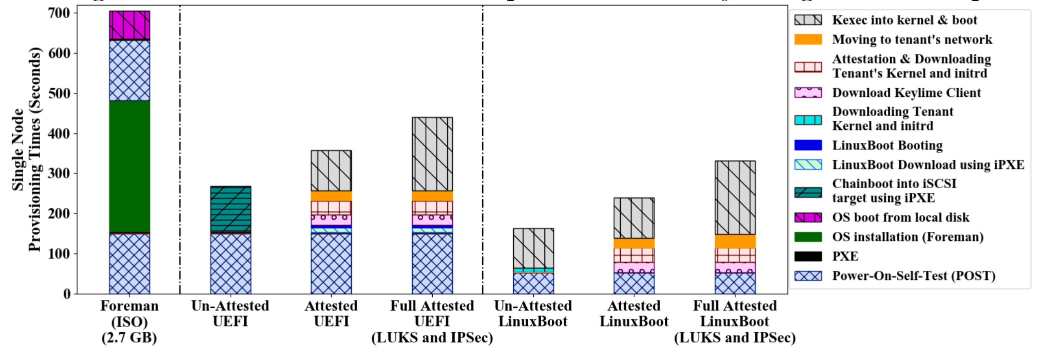

Figure 3·6: Provisioning time of one server.

time to provision a server with Foreman [?] to Bolted with both UEFI and LinuxBoot firmware under 3 scenarios: *no attestation* which would be used by clients that are insensitive to security, *attestation* where the tenant trusts the provider, but uses

[1]In Figure 3·5, we do not include the time taken to prepare the provisioning target for the provisioning systems since this is a manual process for Foreman.

(provider deployed) attestation to ensure that previous tenants have not compromised the server, and *Full attestation*, where a security-sensitive tenant that does not trust the provider uses LUKS to encrypt the disk and IPsec to encrypt the path between the client and iSCSI server.

There are a number of important high-level results from this figure. First for tenants that trust the provider, Bolted using LinuxBoot burned in the ROM is able to provision a server in under 3 minutes in the unattested case and under 4 minutes in the attested case; numbers that are very competitive with virtualized clouds. Second, attestation adds only a modest cost to provisioning a server and is likely a reasonable step for all systems. Third, even for tenants that do not trust the provider, (i.e. LUKS & IPsec) on servers with UEFI, Bolted at ∼7 minutes is still 1.6x faster than Foreman provisioning; note that Foreman implements no security procedures and is likely faster than existing cloud provisioning systems that use techniques like re-flashing firmware to protect tenants from firmware attacks.

While the steps for attestation were complex to implement, the overall performance cost is relatively modest, adding only around 25% to the cost of provisioning a server.[2] This is an important result given a large number of bare-metal systems (e.g. CloudLab, Chameleon, Foreman, . . .), that take no security measure today to ensure that firmware has not been corrupted. There is no performance justification today for not using attestation, and our project has demonstrated that it is possible to measure all components needed to boot a server securely.

[2]Moreover, given that performance is sufficient, we have so far made no effort to optimize the implementation. Obvious opportunities include better download protocols than HTTP, porting the Keylime Agent from python to Rust, etc.

Chapter 4

Pending work

This section discusses the open questions needed to be addressed for the completion of the thesis.

4.1 Distribution System

Building a system that provides fast and secure mechanism for sharing physical servers between organizations does not guarantee that the said sharing will occur. We need a mechanism to encourage exchange of resources where no one organization dictates how the resources are allocated. Ideally, we would want a system that allocates resources where it is needed most; provides incentives to participating clusters to actively share their under-utilized resources. Such a system may have to work with the following constraints.

1. **No Control over hardware resources:** Such a system will have to be resilient to changes in the inventory of the underlying hardware resource pool. Each organization owns and has complete control of the resources. Also, any organization has complete freedom to join or leave the system whenever they wish.

2. **No control to dictate distribution of hardware resources:** Each organization reserves the right to share, how much to share or not share its resources with all, or a specific group of other organizations. Since the distribution service cannot compel any provider to offer its resources, an alternative is to provide incentives to encourage providers to offer their resources proactively.

3. **Ability to work with minimal information:** The distribution system will have to develop mechanisms to decide relative priority of multiple demands without having any knowledge of how resources are deployed internal to the organizations.

4. **Mechanics for capacity planning without knowing future requirements** The system should provide a way that mutually non-trusting organizations can safely express their future needs and make reservations for future allocations in advance.

4.2 Questions I need to answer for the completion of this PhD

1. When there are multiple providers and multiple consumers what is the best way to match the demand with supply such that the computational resources are available to the job that needs it most. How to determine relative importance of jobs when the resources available are less than the cumulative demand. Since it is a decentralized model how is the allocation done which is transparent and fair.

   Sahil: example: do I do fair-share, priority, auctions, reservations, price mechanism etc.

2. Every group/cluster/workload has their own optimization model. What incentives to provide such that everyone is motivated to share their hardware in the marketplace as soon as it is available. Is there a way to measure it. As in, how do I know that they are more willing to share their nodes with others now then they were with respect to some (still need to figure out that) reference.

   Sahil: example: with a centralized scheduler, there is no incentive to do internal migration of jobs for a supplying cluster. In a marketplace mechanism a supplying cluster may reshape its own workload so as to make its machines available in future. (Reservation mechanism)

3. What are the possibilities where such a market would crash and is there a way to have built in mechanisms to discourage such behavior ?

   Sahil: example: For provider and a customer to get into a future reservation, they will have to park some fraction money with the clearing house for the contract to be formed.

4.2.1 Evaluation Metrics for the Marketplace:

1. **HIL + Marketplace:** Since each group/cluster/workload has its own objective, how much less hardware is required when they trade hardware with each other as compared to when they were statically partitioned.

2. **HIL + Marketplace:** How much "extra-demand" were they able to meet which would not have been possible when they were statically partitioned.

3. **HIL + BMI + Marketplace:** Compared to a diskful approach (foreman) what is the effect of using BMI on the demand a marketplace is able to meet (for each workload).

4. **HIL + BMI + Security + Marketplace:** Implications of ability to share a node between security sensitive cluster with not-so-sensitive cluster on the price of the physical servers. With the ability to share physical servers between security-sensitive clusters vs normal (common security practices) clusters, what is the benefit that secure clusters have over statically partitioned and/or slow provisioning clusters.

4.2.2 Stretch Goals:

What incentives are available so as to make the marketplace attractive not just for providers with massive capacities of hardware but even smallest providers (say a researcher with a single machine)

Sahil: example: Provide subsidies for new players, where their jobs priorities is bumped up in way that they have higher probability of receiving hardware. Conversely, their hardware gets allocated for some initial period of time. For eg. system compensates if they quote higher price than the asking price of the market. This is for limited time until they get used trading in the marketplace.

4.3 Proposed Timeline

- Allocation and Isolation Service: Done
- Provisioning Service: Done
- Security Framework: Done
- Decentralized Distribution System:
  - Design: End of January 2020
  - Prototype: End of February 2020
  - Evaluation: End of March 2020
- Thesis writing – First Draft: April 2020
- Thesis defense: May 2020

Bibliography

[1] Openstack, build the future of Open Infrastructure. `https://www.openstack.org/software`, Last accessed: 2019-10-09.

[2] Amazon Web Services (AWS), 2006. `https://aws.amazon.com/about-aws`, Last accessed on 2019-10-21.

[3] Creating a Classified Processing Enclave in the Public Cloud |IARPA, 2017. `https://www.iarpa.gov/index.php/working-with-iarpa/requests-for-information/creating-a-classified-processing-enclave-in-the-public-cloud`.

[4] CloudLab, 2019. `https://www.cloudlab.us/`.

[5] globus/globus-toolkit, Sept. 2019. original-date: 2013-11-04T17:20:35Z.

[6] The Google Cloud, 2019. `https://cloud.google.com`, Last accessed on 2019-10-21.

[7] IBM Cloud, 2019. `https://www.ibm.com/cloud`, Last accessed on 2019-10-21.

[8] The Massachussets Green High Performance Computing Center, 2019. `https://www.mghpcc.org/`.

[9] Microsoft Azure, 2019. `https://azure.microsoft.com/en-us/overview`, Last accessed on 2019-10-21.

[10] Anderson, D. P. Boinc: A system for public-resource computing and storage. In *proceedings of the 5th IEEE/ACM International Workshop on Grid Computing* (2004), IEEE Computer Society, pp. 4–10.

[11] Azzedin, F., and Maheswaran, M. Evolving and managing trust in grid computing systems. In *IEEE CCECE2002. Canadian Conference on Electrical and Computer Engineering. Conference Proceedings (Cat. No.02CH37373)* (May 2002), vol. 3, pp. 1424–1429 vol.3.

[12] Berman, M., Chase, J. S., Landweber, L., et al. Geni: A federated testbed for innovative network experiments. *Computer Networks 61*, 0 (2014), 5 – 23. Special issue on Future Internet Testbeds.

[13] Bestavros, A., and Krieger, O. Toward an open cloud marketplace: Vision and first steps. *IEEE Internet Computing 18*, 1 (Jan 2014), 72–77.

[14] Butler, B. Which is cheaper: Public or private clouds?, Oct. 2016. `https://www.networkworld.com/article/3131942/which-is-cheaper-public-or-private-clouds.html`.

[15] Canonical. Metal as a Service. `https://maas.ubuntu.com/`.

[16] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., and Safina, L. *Microservices: Yesterday, Today, and Tomorrow.* Springer International Publishing, Cham, 2017, pp. 195–216.

[17] Duplyakin, D., Johnson, D., and Ricci, R. The part-time cloud: Enabling balanced elasticity between diverse computing environments. In *Proceedings of the 8th Workshop on Scientific Cloud Computing* (New York, NY, USA, 2017), ScienceCloud '17, ACM, pp. 1–8.

[18] Engler, D. R., Kaashoek, M. F., and Otoole, J. Exokernel: an operating system architecture for application-level resource management. In *ACM SIGOPS Operating Systems Review* (New York, NY, USA, 1995), SOSP '95, pp. 251–266.

[19] Foreman. Foreman provisioning and configuration system. `http://theforeman.org`.

[20] Foster, I., and Kesselman, C. Globus: a metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing 11*, 2 (1997), 115–128.

[21] Grimshaw, A. S., Wulf, W. A., French, J. C., Weaver, A. C., and Reynolds Jr, P. Legion: The next logical step toward a nationwide virtual computer. Tech. rep., Technical Report CS-94-21, University of Virginia, 1994.

[22] Hart, D. L. Measuring teragrid: workload characterization for a high-performance computing federation. *The International Journal of High Performance Computing Applications 25*, 4 (2011), 451–465.

[23] Hindman, B., Konwinski, A., Zaharia, M., Ghodsi, A., Joseph, A. D., Katz, R. H., Shenker, S., and Stoica, I. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI* (2011), vol. 11, pp. 22–22.

[24] Karnow, C. E. The grid: Blueprint for a new computing infrastructure ed. by ian foster and carl kesselman. *Leonardo 32*, 4 (1999), 331–331.

[25] Keahey, K., Figueiredo, R., Fortes, J., Freeman, T., and Tsugawa, M. Science clouds: Early experiences in cloud computing for scientific applications. *Cloud computing and applications 2008* (2008), 825–830.

[26] Kirkwood, G., and Suarez, A. Cloud Wars! Public vs Private Cloud Economics, 2017. `https://www.openstack.org/summit/boston-2017/summit-schedule/events/17910/cloud-wars-public-vs-private-cloud-economics`.

[27] Lai, K., Huberman, B. A., and Fine, L. Tycoon: A distributed market-based resource allocation system. *arXiv preprint cs/0404013* (2004).

[28] Litzkow, M. J., Livny, M., and Mutka, M. W. Condor-a hunter of idle workstations. Tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 1987.

[29] Milojii, D., Llorente, I. M., and Montero, R. S. Opennebula: A cloud management tool. *IEEE Internet Computing 15*, 2 (March 2011), 11–14.

[30] Newman, S. *Building Microservices*, 1st ed. O'Reilly Media, Inc., 2015.

[31] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. Eucalyptus : A technical report on an elastic utility computing architecture linking your programs to useful systems, 2008.

[32] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. The eucalyptus open-source cloud-computing system. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid* (Washington, DC, USA, 2009), CCGRID '09, IEEE Computer Society, pp. 124–131.

[33] Parashar, M., and Lee, C. A. Grid computing: introduction and overview. *Proceedings of the IEEE, special issue on grid computing 93*, 3 (2005), 479–484.

[34] Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., and Wilkes, J. Omega: flexible, scalable schedulers for large compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)* (Prague, Czech Republic, 2013), pp. 351–364.

[35] Silva, G. C., Rose, L. M., and Calinescu, R. A systematic review of cloud lock-in solutions. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science* (Dec 2013), vol. 2, pp. 363–368.

[36] Thain, D., Tannenbaum, T., and Livny, M. Condor and the grid. *Grid computing: Making the global infrastructure a reality* (2003), 299–335.

[37] Tomonori, F., and Christie, M. tgt: Framework for storage target drivers. In *Linux Symposium* (2006).

[38] Van der Veen, D., et al. Openstack ironic wiki. `https://wiki.openstack.org/wiki/Ironic`.

[39] Verma, A., Pedrosa, L., Korupolu, M., Oppenheimer, D., Tune, E., and Wilkes, J. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 18:1–18:17.

[40] Weil, S. A. *Ceph: reliable, scalable, and high-performance distributed storage*, vol. 68. 2007.

[41] White, B., Lepreau, J., Stoller, L., Ricci, R., et al. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev. 36*, SI (Dec. 2002), 255–270.