BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Prospectus

Marketplace at the Bottom of the Cloud

A de-centralized platform for trading bare-metal servers between nontrusting datacenter Tenants.

by

SAHIL TIKALE

B.E., L.D.College of Engineering, 2003
M.S., Nanyang Technological University, 2010

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2020

Marketplace at the Bottom of the Cloud

A de-centralized platform for trading bare-metal servers between nontrusting datacenter Tenants.

SAHIL TIKALE

Boston University, College of Engineering, 2020

Major Professors: Orran Krieger, PhD
Professor of Electrical and Computer Engineering

Larry Rudolph, PhD
Professor of Computer Science and Artificial Intelligence Laboratory, MIT

David Starobinsky, PhD
Professor of Electrical and Computer Engineering

Ayse Coskun, PhD
Asso. Professor of Electrical and Computer Engineering

Peter Desnoyers, PhD
Asso. Professor of Electrical and Computer Engineering, Northeastern University

ABSTRACT

The two big advantages of a cloud is elasticity and a rich set of services. Yet many organizations are not willing to use a single-provider commercial cloud because they do not offer complete control of its hardware, does not allow custom automation for deployment, do not support stringent security requirements and have high storage costs. Currently the Infrastructure-as-a-Service (IaaS) layer is owned and controlled as a black-box by every cloud provider. The lack of visibility into the bottom-most layer of the cloud results in inefficient management and poor allocation of resources. Additionally, hiding operational details from the academia and research community stifles innovation.

This thesis proposes a multi-owner cloud to address these limitations. It specifically focuses on creating a multi-owned federated IaaS layer where many stakeholders, rather than just a single provider, participate in implementing and operating it. It describes the challenges that needs to be addressed for such a cloud to exist. Systems developed to address each challenge can be used independently or as a cohesive unit of a larger system-of-systems such as the multi-owned IaaS layer.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Only a handful of organizations have the capital to invest in setting up an IaaS cloud which naturally leads to an oligopoly – a condition where few organizations dominate the market. To remain competitive IaaS is offered as a black box to its customers where cloud providers often restrict access to information related to type and quantity of hardware, custom deployment practices and data related to its day to day operations. situation leads to the following issues: Opportunities for innovation are limited to large organizations as most of the research and academic community does not have access to the internal workings of an at-scale IaaS 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 [44]. Even their pricing and economic models are geared towards locking the customer into their ecosystem [16].

Many organizations that prefer • complete control over their hardware; • require custom automation for deployment; • have stringent security requirements and • do not wish to pay for high prices of storage [17, 32] continue to operate their clusters outside of the cloud. Open cloud architectures like OpenStack [1] provide an alternative for organizations reluctant to move their computation to a single provider public cloud.

Currently, the option for the industry is to either accept the restrained options offered by the public cloud and benefit from its enormous economies of scale, on demand elasticity and pay-as-you-go model or host a private cloud that provides total control and complete visibility but requires high upfront investment and does not have on-demand elasticity. In case of the research community that wish to innovate in the cloud-computing space, they can either host their own cloud or collaborate with other institutions to build a cloud that might start to resemble the grid computing model but does not match the massive scale at which today's clouds operate.

It is a well known fact that competition breeds excellence and provides a fertile environment for innovation. There is a need to build an at-scale public cloud – a single platform where stake-holders from both industry and academia together host their hardware and services. Such a cloud would allow participants to compete, collaborate and innovate while learning from each other at every level of the cloud

stack. Our hypothesis is that by opening up the IaaS layer to multiple stakeholders would offer new avenues where industry and academia can work together to develop innovative solutions and services that will be truly economic and beneficial for the whole community. The fact that there does not exist any such IaaS layer, let alone a complete cloud, anywhere in the world suggests that this is a promising area of investigation with potentially many open research problems.

The aim of this thesis is to design and implement the Infrastructure-as-a-Service (IaaS) layer of such a multi-owner cloud. It envisions to be a collection of bare-metal servers and networks offered by different organizations in a manner that improves economic efficiency, operational transparency and provides a conducive environment for fostering innovation. To realize this vision this thesis proposes to investigate the following research questions:

1. Is it possible to build an IaaS cloud where multiple organizations, instead of a single provider, can offer their hardware resources to customers and each other in a way that provides visibility into the allocation of resources ? 2. What mechanisms are required to build cloud that is self-regulating without any central authority dictating and controlling its day to day operations ? 3. In the absence of a central authority promising security guarantees to the customers, what security model will allow entities with different levels of security requirements to participate and exchange resources with each other ? 4. How fast is it possible to re-purpose hardware resources among competing services such that hardware providers can easily join and leave without creating vendor lock-ins for the customers. In other words, how fast is it possible to move hardware between the services so as to make such a multi-owner cloud economically feasible. 5. In the absence of any centralized scheduling authority what is the best model to decide which jobs get priority for resource allocation in a way that is transparent and agreeable to all stakeholders?

1.2 Background and Motivation

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 [41, 30, 14]. Each stake-holder decided what fraction and granularity of resources to share with the grid [28]. 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 [27], Globus [25, 5] and Condor [34, 45]. 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 lot 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, 8, 7]. For competitive reasons, each uses proprietary (black box) solutions for orchestration of hardware resources also known as the Infrastructure-as-a-Service layer. 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 [37, 38], OpenNebula [35], Nimbus [31] and OpenStack [1] were developed as an alternative to the single provider public cloud.

Sahil: following paragraphs are repetition of the overview section. Elaborate on the points and turn the questions into statements. Add one or two lines for question/statement describing what you wish to investigate in it.

Currently, the option is to either accept the restricted options offered by the public cloud and benefit from its enormous economies of scale, on demand elasticity and pay-as-you-go model or host a private cloud that provides total control and complete visibility but requires high upfront investment and does not have on-demand elasticity. In case of researchers that wish to innovate in the cloud-computing space, they can either host their own cloud or collaborate with other institutions to build a cloud that might again start to resemble the grid computing model but does not match the massive scale at which today's clouds operate.

There is a need to build an at-scale public cloud where stake-holders from industry and academia host their hardware and services. Such a cloud would allow participants to compete, collaborate and innovate while learning from each other at every level of the cloud stack.

The aim of this thesis is to design and implement the Infrastructure-as-a-Service (IaaS) layer of such a multi-owner cloud. It envisions to be a collection of bare-metal servers and networks offered by different organizations in a manner that improves economic efficiency, operational transparency and provides conducive environment for fostering innovation. To realize this vision following research questions needs to be investigated:

1. Is it possible to build an IaaS cloud where multiple organizations, instead of a single provider, can offer their hardware resources to customers and each other in a

way that provides visibility into the allocation of resources ? 2. What mechanisms are required to build cloud that is self-regulating without any central authority dictating and controlling its day to day operations ? 3. In the absence of a central authority promising security guarantees to the customers, what security model will allow entities with different levels of security requirements to participate and exchange resources with each other ? 4. How fast is it possible to re-purpose hardware resources among competing services such that hardware providers can easily join and leave without creating vendor lock-ins for the customers. In other words, how fast is it possible to move hardware between the services so as to make such a multi-owner cloud economically feasible. 5. In the absence of any centralized scheduling authority what is the best model to decide which jobs get priority for resource allocation in a way that is transparent and agreeable to all stakeholders?

Building a cloud that answers all of the above questions in affirmative requires a detailed understanding of the preferences and constraints of the participating organizations.

Chapter 2

Software Architecture

This section discusses use-cases from different organizations that are interested and will benefit from a multi-owner cloud. Insights from each use-case will be used to derive a global list of requirements that the system has to satisfy. We take the microservice architecture approach where each constituent component is a microservice fulfilling a subset of the requirements.

2.1 Use-Cases

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

A typical job that runs on high performance computing or high throughput computing cluster is a non-interactive, batch job with very low tolerance to performance jitters and network latencies but highly tolerant towards delays in execution. It is normal for HPC and HTC jobs to wait for a long time in the queue until optimal infrastructure is available. Setting up these high (performance and throughput) computing clusters requires considerable investment therefore the preference of HPC administrators is to maximize overall utilization of the cluster.

Figure 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, the HTC cluster can benefit from borrowing extra capacity from any other cluster. The only constraint is that the

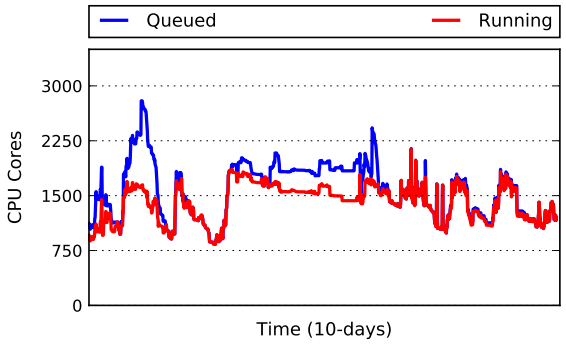

Figure 2·1: Luxembourg University (HPC)

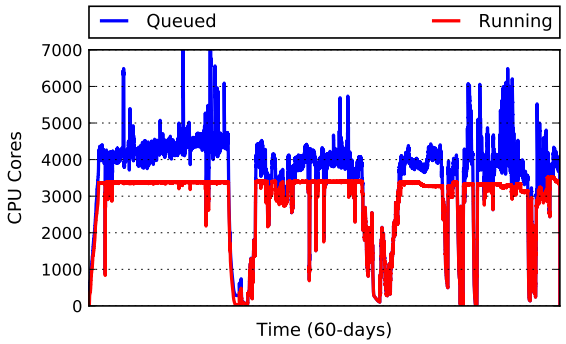

Figure 2·2: Atlas (HTC)

server should be made available bare-metal so that it can be seamlessly added to the current deployment of the HTC cluster.

Similarly, HPC HTC clusters can offer bare-metal nodes to other clusters when they need it. Since the effectiveness of the HPC/HTC clusters is determined by the aggregate CPU utilization over a long period of time, they can afford to operate at lower capacity in the short term if there is an arrangement to ultimately receive more than they give.

System Requirement: 1. Bare-metal multiplexing A system which allows such exchange should support migration of bare-metal servers between clusters with different software stack and methods of deploying it. The migration should be sufficiently fast to address short term change in demand.

2.1.2 Clouds

Applications that are best fitted to run in the cloud are the ones with high variable demand, extremely sensitive (on the order of seconds) to response time when scaling up and are robust to performance jitters and network latencies common in a virtualization based solution. Clouds are a source of revenue generation for many organizations. The preference of a cloud is to maximize its profits. Sensitivity to delay is one of the important constraints that can directly affect its profits.

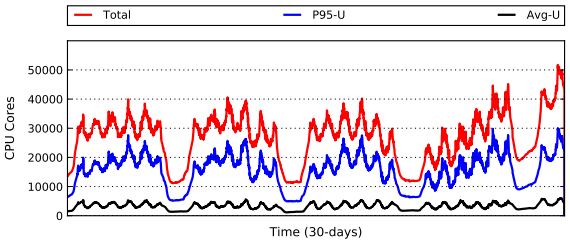

Figure 2·3: Microsoft-Azure (Cloud)

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 offer it to other clusters. For example, say most of the usage is during the day then it is possible to offer the machines to other groups during night time. Even the utilization of cores allocated to VMs is low enough that several VMs can be consolidated on a single physical machine to free up more capacity as is suggested in the work of Cortez et. al [20]. A cloud provider would be interested in offering the excess capacity to other clusters if there is an arrangement where clusters hosting clouds get paid for offering their idle servers outside of the cloud as it would contribute towards offsetting their operating costs. But, they would be willing to offer resources only if there is a surety that when the cloud experiences surge in demand they can get their servers back from other clusters in the shortest possible time.

System Requirement: 2. Fast provisioning To support scenario discussed above, the system will have to provide mechanisms where a bare-metal node is re-

purposed with different system and software stack in the shortest possible time.

Sahil: I am not sure if I should introduce currency here as a requirement. I feel it should be it own discussion where currency is a standard for measuring relative importance of computation.

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. For reproducibility of experiments 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 homogenous servers and in quantity adequate to perform their experiments. Researchers are also highly tolerant to delays in the availability of servers. They can wait till weekend or do experiments at night if more hardware is made available. Their constraint is that they are less tolerant to variation in the type of hardware. For the consistency of results their preference would be to run all experiments on the same set of hardware.

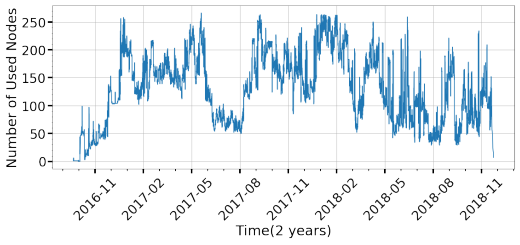

Figure 2·4: Bare-metal allocation variability over a period of few months. High demand near the systems research deadlines.

System Requirement: 3. Security: As is evident from the utilization curve as shown in Figure 3·1 research test beds have enough idle capacity that can be offered to other clusters operated by other tenants. One caveat of using this servers is lack of security. There is no guarantee that the servers other recieve from a research group do not have a compromised or malicious firmware. Using nodes from a research group requires a security model that allows other clusters to verify the integrity of the machines before allowing them to join their workloads.

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 shared data-center agree to let the federal agencies use a dedicated ratio of their infrastructure in times of national emergency it would require all participating organizations to provide some security framework and ability to rapidly repurpose hardware for government use. Such an arrangement will be both time sensitive and security sensitive.

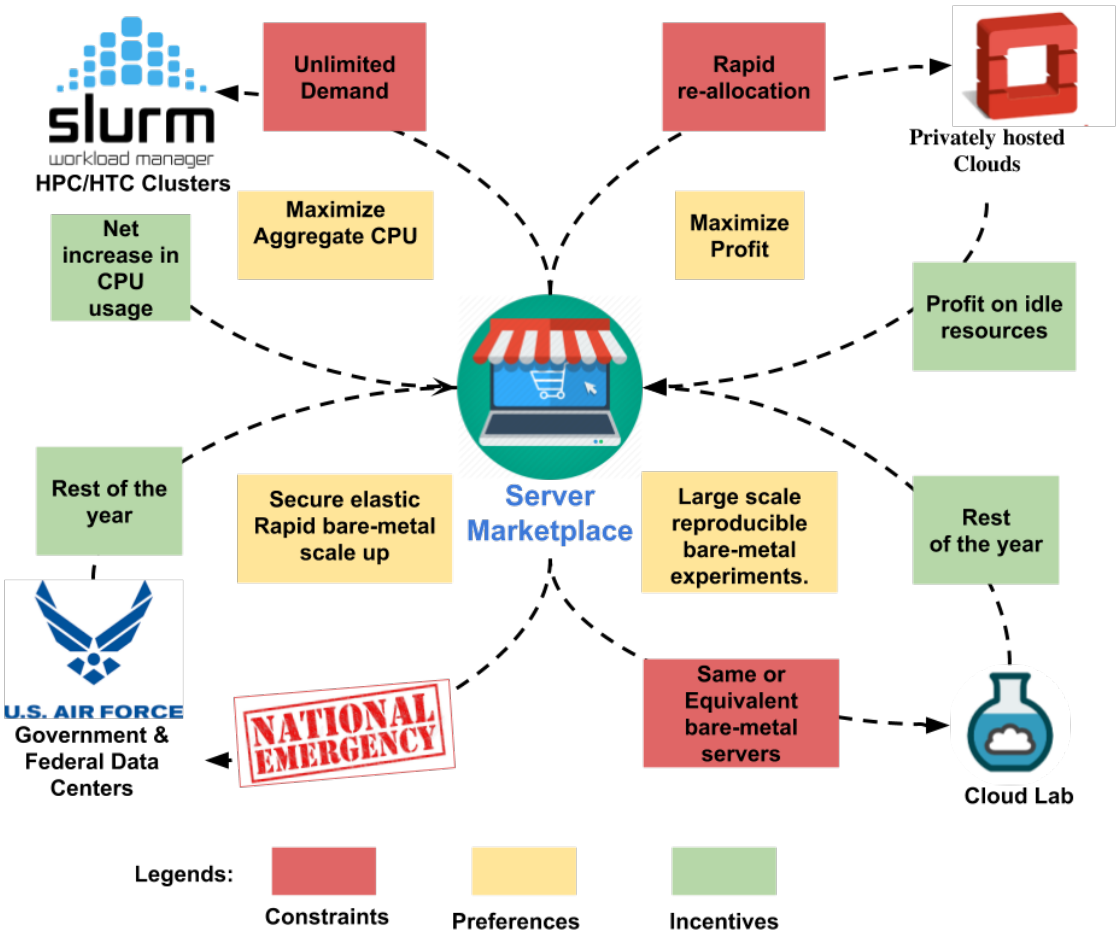

Figure 2·5: Different self-interests of each autonomous organization that motivates them to participate in server trading market

Figure 2·5 is a diagramatic representation of multiple tenants supplying physical servers as per their own preferences and constraints along with their incentives to

participate in such an arrangement. It is possible that the interval of peak demand on one cluster may coincide with the interval of low-utilization for other cluster. In such cases, both the tenants can benefit from exchange of resources.

2.2 Requirements

Based on the use-cases following are the requirements that our system needs to satisfy:

1. **Bare-metal Multiplexing:** Any participating cluster that wishes to offer or use bare-metal capacity for a few hours should have a fast way to add machines to and from their clusters. It is important that they have complete control on how they wish to share their bare-metal servers with others.

2. **Fast provisioning:** Each cluster and organization have their own procedures and custom methods of deploying system and software stack on a bare-metal node required for servicing the jobs. Moving a bare-metal node from one cluster to another is a two step process.

   (a) Release the node from existing cluster. It includes removing all the data, application and system software stack of the releasing cluster. (b) Adding the node the new cluster. It includes deploying system and application software on the node to make it ready to service the workload of the receiving cluster.

   Minimizing the time to move a bare-metal node is very important to increase the chances of different organizations willing to offer their infrastructure for short intervals of time.

3. **Security:** Different organizations value security differently. Security sensitive organizations like government agencies, financial companies and medical institutions invest substantial time and effort towards the upkeep of their infrastructure security compared to a university or a startup company. Sharing a bare-metal node between clusters with different security requirements can be a major concern especially. To make server exchange possible between group with different standards for security we need a mechanism to verify whether a server matches the security standard of the organization.

4. **Marketplace:** Building a system that allows sharing a bare-metal node in a fast and secure manner between organizations does not ensure that the said sharing will actually take place. The use-cases are an indicative list of different organization 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. A marketplace that rewards sharing of resources between organizations seems to be a good fit.

2.3 Design Principles and Choice of Architecture

The architecture should support the following properties.

- **Decentralization:** Any architecture design with centralized control is a non-starter because all the organizations would like to retain complete control over how and with whom they share their infrastructure. A software architecture that supports decentralized administration and allocation of resources would be suitable building such a system.
- **Diversity:** Creating an environment that promotes innovation requires providing space for new solutions. This includes the ability to offer new services and heterogenous hardware.
- **Adapatability:** Aggregate capacity of servers available in the cloud will change as stakeholders choose to add new servers to the pool or retire old servers. Capacity may also change when new stakeholders join the cloud or some choose to leave it and take their hardware with them. The system should be able to adapt to the dynamic changes in the underlying capacity.
- **Interoperatability:** There should be a standard form of communication for different services to communicate with each other.
- **Fault Tolerant:** Service replacement can be planned, coordinated while failures are unpredictable. Such a system should be resilient to failure. There must not be any single domain of failure.

All of the above requirements eliminates any monolithic software arhitecture as the candidate of choice. A monolithic software is designed as a single system that requires all of its components to work correctly for it to be operational. Any changes to a single module requires making changes to the whole system. Failure of a single component may result whole system being non-operational.

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."* [21]. Loose decoupling and collaboration by message passing via standard interfaces 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 [36, 21].

Microservice Architecture of a multi-owner cloud

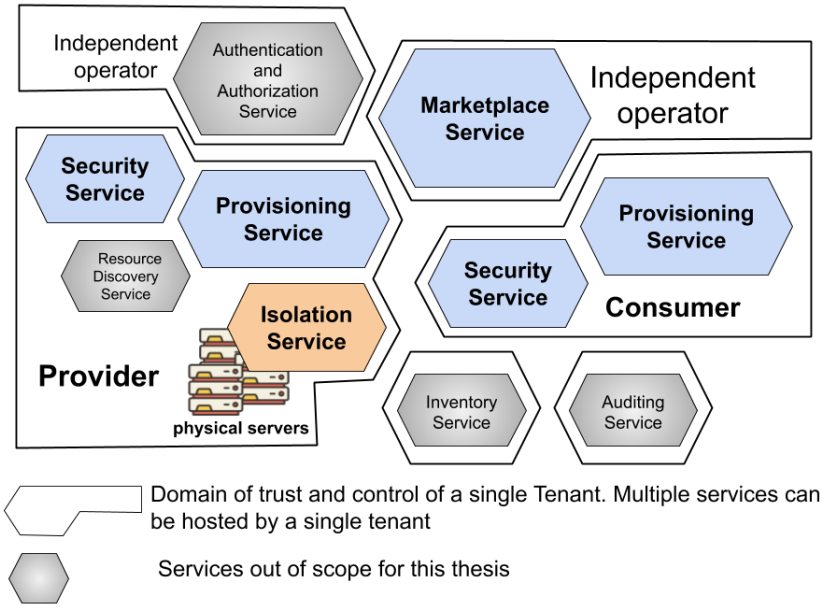

Figure 2·6: A consumer has to depend upon only Isolation Service deployed by the provider. Consumers can either deploy all of the required services themselves or use third party services or use services offered by the provider. In this thesis we will focus only on 1. Isolation Service; 2. Provisioning Service; 3. Security Service; 4. Marketplace service.

For a multi-owner IaaS cloud multiple microservices will be required. Each of which can be hosted by an independent organization. Figure 2·6 shows the microservice architecture where different organizations are free to either host their own services or use the ones offered by third party. Isolation service is the only service that only the organization providing the bare-metal servers can host.Consuming organization will have to rely on it to use the hardware resources of the providing organization. Following are some of the benefits of having the flexibility of choice between host-your-own vs use-third-party services

- Smaller barrier to entry: small organizations like start-ups with limited resources can still participate by being consumers of third party services. They can choose from multiple choices available.
- Independence: Organizations that have the ability and resources to implement their own services can do so and still enjoy the option of using third party services.
- Competition breeds excellence: With multiple organizations hosting similar services, it provides a good environment for encouraging innovation.

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

Chapter 3

Completed work

The following sections discuss the work done so far to satisfy the requirements essential for building FLOCX . We developed sub-systems each of which satisfies one or more of the requirements listed in the previous section. All of them are open-source and follow the micro-service approach. Besides being scalable and fault-tolerant, being microservices also allows for independent development and maintenance of features for each sub-system. We have designed them such that they not only contribute towards building FLOCX but each can be used independently by organizations to fit their purpose. We will discuss in detail about *Hardware Isolation Layer* (HIL) a subsystem built to satisfy the requirements of *Control* and *Connectivity*. It also satisfies the requirement of *speed of multiplexing* along with the subsystem called *Bare metal Imaging Service*(BMIS) and *Bolted* is the implementation of HIL and BMIS that satisfies the requirement of *security*.

3.1 Hardware Isolation Layer (HIL)

Building a platform that allows different organizations to contribute their clusters to collectievely host, operate and consume an IaaS cloud requires that each cluster can scale up or shrink down on demand. Such elasticity at IaaS layer means the ability to add more physical servers to the existing cluster during peak demand and release physical servers when not wanted. The challenge is in providing such elasticity without affecting the traditional processes and tools of controlling the hardware which is different for each organization. It is tempting to simplify the problem by enforcing all organizations to use a single deployment system (eg. OpenStack) but it would be a non-starter for following reasons.

- Different tools are better suited for some applications than others; e.g. Ironic is used for OpenStack deployment, while xCAT is well-suited for large High Performance Computing (HPC) deployments.
- No organization would be willing to abandon their battle-tested solutions and practices developed over long period of time, customizing it for their environment.

- Enforcing the use of single system would impede innovation and discourage new organizations with potentially better solutions from joining the platform.

A problem with these tools is that each 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 datacenter operator who wants to use multiple tools is forced to statically partition the data center into silos of hardware.

Breaking this silos without changing how organizations control their hardware requires a new fundamental layer that allows resources to move back and forth between them. The following subsection describes the design of such a layer, called the Hardware Isolation Layer (HIL), which adopts an Exokernel-like[23] approach.

With this approach, rather than providing higher level abstractions that virtualize the physical resources, 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.

HIL doesn't hide functionality; in some cases it abstracts datacenter-administrative interfaces that can differ between nodes or switches while preserving the behavior and functionality needed by provisioning tools.

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 HIL to 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.

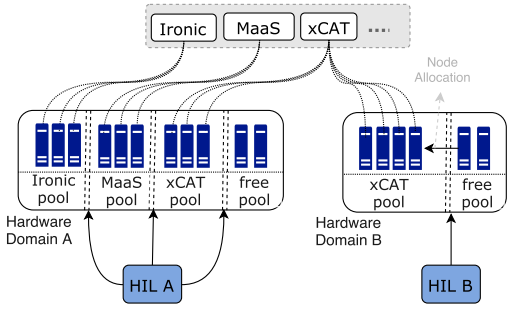

Figure 3·1: HIL provides strong network-based isolation between flexibly-allocated pools of hardware resources, enabling normally incompatible provisioning engines (e.g. Ironic, MaaS, xCAT) to manage nodes in a data center.

Thus HIL satisfies the first two requirements of control and connectivity. A large data center may contain multiple server pools, clusters, and compute services managed by different companies, institutions, or research groups.

HIL can be used in such an environment to break these siloes to allow infrastructure to be moved to wherever it is needed. In such a data center there would be multiple HIL service instances, one per server pool or administrative entity.

3.1.1 HIL - Architecture

The HIL architecture is shown in Figure 3·2. It provides a REST API, and is implemented by components linked via REST APIs or programmatic interfaces. These components can be categorized as: • Core HIL components, • System drivers: pluggable implementations of a standard interface to a specific external system (e.g. network switch control), • Optional services: HIL-related services which can be overridden on a per-allocation basis.

Core components and drivers

The *core* components are (a) the HIL server itself, implementing the HIL API, (b) the database, which persists HIL state and configuration, and (c) the *operation engine*, which sequences potentially long-running network operations. **HIL Server** implements most of the HIL logic and exposes HIL's REST API, interfacing with the database, the operation engine, and the out-of-band management (OBM) drivers. The primary aspects of its logic are:

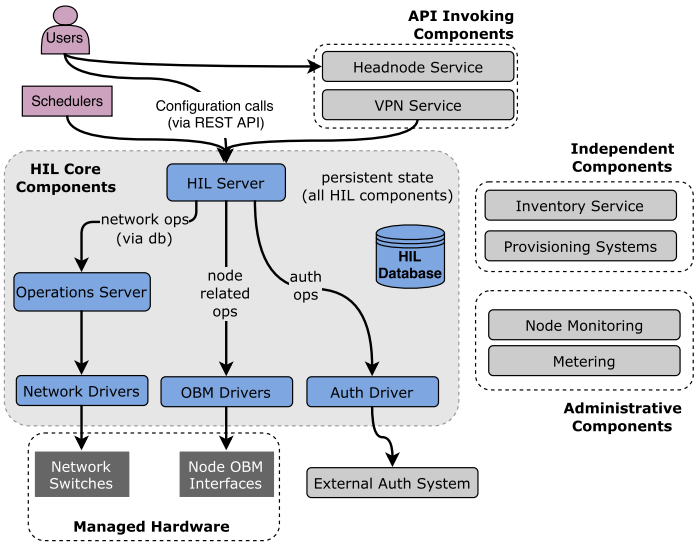

Figure 3·2: Components in a HIL deployment.

• authentication and authorization of requests, applying controls based on identity, object ownership, and configured state (e.g. permissions, quotas), • node control operations, which are executed via an appropriate OBM driver, and • network configuration actions, which are forwarded to the operation engine via an in-database request queue. Most requests take effect on the database and complete immediately; longer-running requests return a pending status and the API client is responsible for polling for completion.

OBM and Auth drivers: The HIL server controls individual bare-metal nodes via the *OBM driver*, which provides functions to power cycle a node, force network booting, and access the serial console log. The OBM driver exports these functions over a programmatic interface, and is implemented using IPMI with vendor-specific extensions as necessary. Authentication and authorization decisions are forwarded to the *auth driver*; authorization is performed by passing a request description and receiving in response a decision (allow/deny) and optionally an authorization token which may be forwarded to other services for delegation of authorization. Currently two authentication drivers are available: one uses simple database tables, while the

other forwards requests to OpenStack Keystone [40], providing access to multiple authentication backends as well as token-based delegation of authorization.

Operation Engine is responsible for sequencing and coordinating operations which change network configuration. While an operation is pending against a NIC, HIL prevents certain operations (like the free'ing of nodes or making new requests against the same NIC) which could compromise a network's isolation.

Switch drivers

These are used by the operation engine to implement the functions which manipulate network connectivity, with implementation (but not interface) varying by the network technology being used (e.g. VLAN), as well as vendor and model of the device being controlled. By manipulating network connectivity HIL is able to protect nodes within a project (or their management interfaces) against access from other projects or external systems or even other HIL deployments.

HIL Optional Components

The remaining components of a HIL deployment are *optional* on a project-by-project basis. The full set of features is in fact quite similar to that of Emulab or GENI; however by taking the modest step of allowing users to forgo services such as software provisioning, the range of compatible applications is vastly increased.

3.1.2 HIL - Evaluation

We have developed an initial prototype of the HIL architecture described in the previous section. The prototype is very simple, with less than 3,000 lines of code in the core functionality Even though this is a proof of concept implementation, the functionality offered makes operations management so much easier that the prototype is already being used in production on a daily basis in a cluster of 48 Cisco UCS C220 M3 nodes, each with one 10 Gb NIC, dual 6-core CPUs with hyper-threading enabled, 128 GB of RAM, and one or two disks.

Sahil: Update this to reflect current clusters in MOC

In the remainder of this section we report on experience gained in using HIL to deploy a variety of applications, and describe a model showing the economic value of shifting hardware resources between different services.

Metrics and Scaling

The runtime of key HIL operations is shown in Table 3.1. For database-only operations that do not interact with the switch (i.e. allocating or freeing a project, node or network), completion time is in the tens of milliseconds.

Interacting with the switch consists of two components: the time to complete and reply to each request—600-700 ms—and the time (several seconds) needed for the operation engine to establish a control session to the switch and send commands, although multiple commands may be sent over a single control connection, amortizing this overhead to some degree.

Table 3.1: Median and standard deviation times for key operations in HIL in seconds. Each operation was repeated 250 times.

Operation	Median (secs)	Standard Deviation
Create project	0.011	0.002
Delete project	0.017	0.003
Allocate node	0.011	0.003
Free node	0.098	0.008
Allocate network	0.017	0.004
Free network	0.016	0.004
Connect NIC	4.336	0.079
Detach NIC	2.301	0.14

Figure 3·3 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.

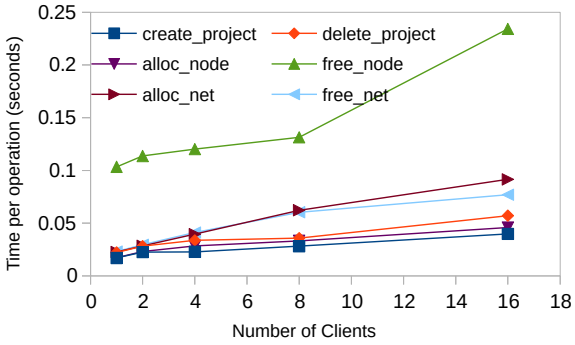

Figure 3·3: Scalability of HIL synchronous operations

Figure 3·4 shows the performance of asynchronous API operations.

These operations involve interactions with the networking switches and as a result take an order of magnitude longer to complete. In this graph, performance degrades further with more concurrent requests because all requests target the same networking switch which becomes a bottleneck. As the number of switches scale in a larger environment (with some optimization of HIL) concurrency should also improve.

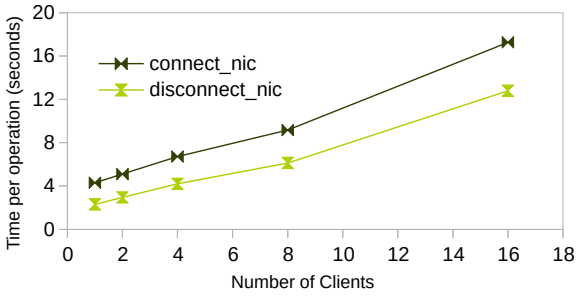

Figure 3·4: Scalability of HIL asynchronous operations

In actual usage, each node allocated is typically rebooted at least once, for software provisioning, before it may be used; the time for this boot process is significantly longer than that of any HIL operations. On server-class machines the hardware Power On Self-Test (POST) process may take minutes in hardware discovery, initialization,

Table 3.2: Power On Self Test median times and standard deviation on various platforms, optimized and unoptimized, 10 repetitions.

System	Unoptimized		Optimized	
	Median (sec)	Std.	Median (sec)	Std.
Lenovo M83	14.5	1.080	–	–
Dell R620	108.0	0.675	97.0	0.632
Cisco C220 M3	122.0	1.450	78.0	0.850

and verification, along with repeated user prompts and timeouts.

In Table 3.2 we see POST timings for three different systems, each in its default (unoptimized) configuration and with all optional POST features disabled (optimized). POST time for the desktop system measured was much lower but the two server-class systems each required about 1.5 minutes to boot in the best case, and over two minutes in the worst.

To estimate the maximum size of a single HIL deployment, we assume:

• average node lease time T_{up} is 3100 secs, based on median virtual machine lifespan measured from logs of an academic OpenStack cluster, • each node requires one network operation for allocation and one for deallocation, ignoring additional (and much faster) HIL operations, • network control connection overhead is amortized over sufficiently large numbers of operations; subtracting this overhead we have a per-operation cost of 0.7 secs, or a total cost $T_{net\ ops}$ of 1.4 secs for two operations, and • POST time T_{post} is 80 secs. Given N nodes, the total utilization of the HIL server is

$$\rho = \frac{N \cdot T_{net\ ops}}{T_{net\ ops} + T_{up} + T_{post}} \tag{3.1}$$

Based on this result, a single HIL instance would be able to handle over 1700 nodes with $\rho < 0.75$. In the worst-case scenario of simultaneous allocation or deallocation of large numbers of nodes, HIL would be able to handle $N = \frac{T_{post}}{0.7}$ or 117 allocations or deallocations before the worst-case operation time exceeded the node POST time.

These estimates are based on measurements of the current very simple HIL implementation that uses low performance Expect[1] scripts to interact with the switch. While we could improve this performance dramatically, the existing implementation is more than sufficient for the scale of individual PODs in our data center, and we have little incentive to improve the performance or scalability of this part of the implementation.

3.1.3 Related Work

The value of self-service access to physical infrastructure has been recognized for a long time, and there are a large number of production and research systems that have been developed [9, 15, 19, 42, 13, 39, 46, 18, 48, 24, 26, 11]. The sheer diversity of these projects suggest that it is unlikely that any one system will solve all use cases,

[1] `expect.sourceforge.net`

and there is value in a HIL-like layer to allow these services to co-exist in the data center and to allow new services and new research in provisioning systems.

A previous system that provides a very similar level of abstraction to HIL is Project Kittyhawk [12], which inspired the design of HIL. HIL differs from Kittyhawk in that Kittyhawk was designed around the capabilities of IBM's Blue Gene. In addition, HIL allows existing provisioning systems to be used unmodified, while Kittyhawk required users deploying a provisioning system to modify it to support Kittyhawk's abstractions.

3.2 Bare Metal Imaging Service

HIL satisfies the requirements of *Control, Connectivity* and partially *speed of multiplexing*. With HIL it is possible to dynamically adjust physical nodes between virtualized and non-virtualized clusters according to demand. But most of the provisioning systems or tools used for deployment systems and application software on a bare-metal physical server still take few hours to get the cluster ready for workload processing. The most time consuming step is installing system and application software on local disks.

3.3 Bare Metal Imaging

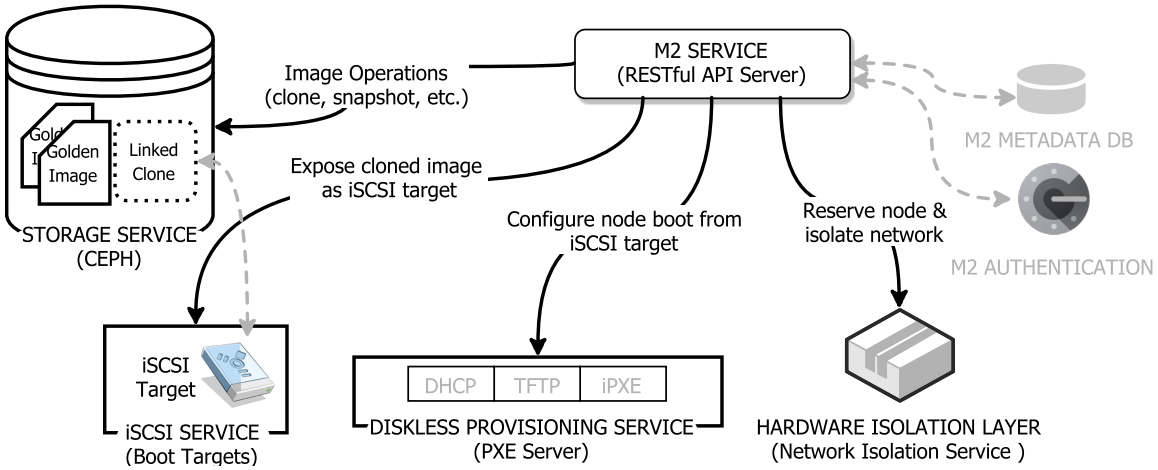

Figure 3·5: BMI architecture

Describe BMI here.

3.4 Bolted

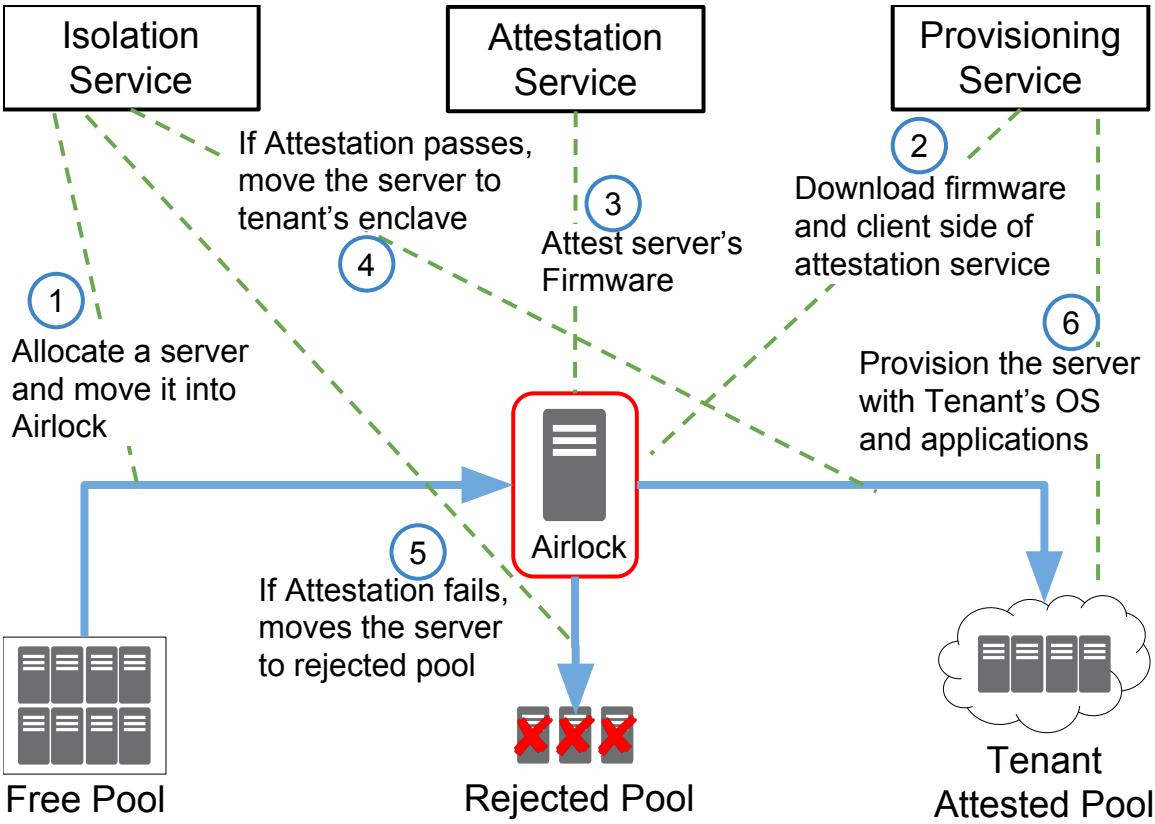

Figure 3·6: Bolted architecture

Describe Bolted here.

Chapter 4

Things to do: Create a marketplace

As described in the previous sections, we developed services namely HIL, BMI, Bolted which henceforth we will address collectively as Elastic Secure Infrastructure or ESI. ESI offers a collective solution where an organization can setup a cluster on demand using bare-metal servers owned by other organization. This opens the possibility of sharing physical servers between non-trusting entities to meet respective demands. But simply creating services that allow for moving of resources between clusters is not enough. We also a need a system that encourages exchange of resources. Such a system will have to satisfy the following requirements:

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:** The system can influence, encourage but cannot compell any organization to share its 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.

3. **Ability to work with minimal information:** The system does not have access to any information other than what the participant wishes make public. It means it has no direct access to the internal demand requirements of each cluster and cannot compare its relative importance. Yet it has to have a mechanism to allocate resources where they are most needed.

4. **Mechanics for capacity planning without knowing future requirements**

   Sahil: I feel this is more of a resulting benefit of the system rather than requirement.. open for discussion

   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.

The following section discusses why there is a need to develop a new system that satisfies the above requirements. We look at recent literature about similar systems and compare contrast it in the light of these requirements.

4.1 Literature Review

Dynamic partitioning of a large scale cluster and finding optimal method of resource allocation is a well known problem that has recieved lot of attention from research community over the years. It is difficult to cover all the research here, hence we cover relevant work and classify them as follows:

Types of schdulers

1. Dynamic partitioning systems with complete control on the underlying hardware capacity.
2. Dynamic partioning systems with no control on the underlying hardware capacity.
3.

Type 1

Borg: Borg [47] 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 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 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 [43] takes a two stage shared state approach where there is one global scheduler and mulitple 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

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

it. Although it is not clear what parameters are used to resolve such conflicts. In FLOCX 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 FLOCX it may increase the ratio of conflicted resources which would be difficult to resolve especially when FLOCX does not own the hardware resources.

Mesos: [29] resolves the issue of resource conflict by offering it sequentially, one at a time, to each of its participating clusters. Mesos calls these participating clusters as framework that runs its own scheduler while Mesos is a global scheduler that offers unused resources to each framework. 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 design choices to make decisions like using a common orchestration system (Virtualization, Containers etc) and assuming no malicious behaviour 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.

Type 2

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. Unlike the systems of *type 1* Boinc [10] does not control either the fuctioning 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 [33] 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 FLOCX is the segregation of resource allocation mechanism from strategies of bidding for resources. In contrast to the works of Type 1 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 [22] 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 pre-empt 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 pre-empt 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 5

Things to do: Create a marketplace

In addition to no central authority,a resources allocation system should also:

1. Allow complete freedom to choice whether an organization wishes to only use resources offered by others; only offer resources to others; do both or not participate at all.

2. Allow complete control on how each wishes to offer and for what duration or any other conditions that they would like to enforce for the use of their hardware.

3. Enable organizations to seek or offer hardware without revealing any private information about the usage patterns (what are they using it for, internal allocation patterns, etc) internal to the organization/cluster.

4. Allow organizations to reserve capacity for use in distant future.

Markets are the most suitable form of resource allocation system that fits the requirements listed above. It uses incentives to encourage rather than force individual participants to offer more goods and services. Often the prices of goods and services provide a standardized signal about the incentive to offer or buy more of an item. Price rise is a response to resource scarcity or high demand thereby rationing its use or allocate it to the most needy. Prices drop is indicative of resource abundance or depletion in demand. Prices also allows different entities to express how much they value a resource without revealing any private information (satisfies constratin 3).

The desirable properties that any market should possess in order to be efficient and sustainable is widely studied in economics. Following are the properties [1] that a bare-metal exchange market should have in order to satisfy the constraints enumerated above:

(a) *Individual Rationality:* Also known as *voluntary participation*. Allocaiton system should be designed in way a that every organization finds more value in offering their resources for use of others rather than letting them sit unused in their clusters. It is an important property that is required for operating a sustainable market without violating constraint 1.

[1] Myerson-Satterthwaite impossibility theorem states that any market can only optimize for 3 out of the 4 properties. Fourth being *budget balance* which means that the total amount of money exchanged among the stake-holders is constant. It is a zero-sum game.

(b) *Incentive compatibility:* Also known as *truthfulness* Marketplace should provides incentives that encourages each stake-holder to choose appropriate action that provides maximum value to their particular use-case. With limited information about each stake-holder, the crux of the research is to develop these incentives that is specific to a type of cluster (HPC, Cloud, Systems-research, Government, etc)

(c) *Pareto Optimal:* Also known as *allocation efficiency* which means that the resource (physical server) is always allocated to the cluster that most values it.

The relevant undesirable properties that may cause *market failures* are *adverse selection* and *moral hazard* . The former occurs when transactions occur between two parties, one of which has access to information about the resource that other may not have while later occurs when misleading information is provided by one of the participants involved in the transaction. This effects, if not checked early on, can bring the system to halt.

Sahil: working on examples of each, in terms of bare-metal node exchange

5.1 Research Questions

This section lists the questions (not in order) that this research thesis intends to answer.

5.1.1 Work in progress:

What are the research questions that this thesis intends to answer:
1. Cluster scale marketplace: 2. Multitenant control-plane for bare-metal marketplace: 3. Economic model for the marketplace:

5.1.2 Pareto Optimality:

With ESI layer, we can rapidly (in order of minutes) move hardware between different clusters. How much does the speed of multiplexing affect the efficiency of the marketplace ? What factors of marketplace does it affect and what are its implications ? for eg. How often the price changes due to rapid movement of hardware. Does it make anticipating price change difficult ? The time cost of moving hardware is relatievely static, but the price of the computation time varies with demand. Is it possible to take advantage of the fast movement to devise novel job allocation schemes not possible previously (when the movement was slow due to local installation)

Sahil: novel allocation scheme being cheese hole job allocation scheme where the job is checkpointed and migrated to different machines during its life-cycle, since cost of switching hardware is far less than cost of restarting the job.

Auctions: VCG scheme of 2nd price auction is known for encouraging bidders to express their true value of computation. This thesis seeks to find optimal pricing of multi-unti resource ? What are the pre-conditions under which it makes sense to sell single unit vs multi-unit ? How to price multi-unit, price less than single-unit or more that single-unit ?

Sahil: Need examples, scenarios, use-cases to elaborate on this questions

How do future contracts affect current prices ?

5.1.3 Individual Rationality:

How to ensure that people always participate in the market (offer or demand) and do not use alternatives like external clouds ? What incentives besides low price can we provide to keep critical mass of participants active in the marketplace ? The architecture supports the possibility of multiple markets. If each market has different rules, how to ensure that one does not collapse or cannibalizes other. eg. all offering nodes in the security market and none offering nodes in the university market. In other words how to ensure a degree of social welfare without seriously affecting allocation efficiency ?

5.1.4 Incentive Design:

As described in the use-cases each cluster stands to gain from the market system. How to form incentives that are robust to manipulation by malicious players. eg. Hoarding of resources as a denial of service attacks to the competitor or in general artificial shortage (or surplus). Incentives will have to change over-time as participants learn from the usage history. What attributes (to begin with) should the marketplace capture to adapt incentives as the system evolves ?

Sahil: this can be a bigger research piece involving machine learning which is out of context of this thesis

How do we attract new tenants or increase participation of less frequently participating tenants ? The issue stems from the observation that some players tend to dominate the use of resources in the marketplace and with experience become better than others at using the system. This is deterimental as new tenants shy from participating due to higher learning curve and higher resistance at recieving resources. For them to see the value of participation in such a marketplace is very important. To attract new players there needs to be an incentive for infrequent users to get nodes with average delays at max same as all other participants.

Develop incentives that not just encourages participants to share their underutilized or idle resources but modify their internal preference structure such that they reshuffle their internal job queues (migrate, move in future or killing running jobs) and provide nodes to the market pool when demand is high ?

Similarly during periods of low acitivity or less demand, how to encourage participants to buy the capacity from the marketplace so as to minimize under-utilization ?

What data marketplace needs to collect to learn about changing usage behaviour so as to update the incentives offered ?

5.2 Evaluation Criteria for the Marketplace

- Job throughput over an epoch. Using individual traces with distinct peak intervals and duration, can we how much of this demand was satisfied by exchanging nodes between clusters. Efficiency in terms of jobs throughput. No of jobs per time slice should be a bigger number in the marketplace than in individual customers.
- examples in the flocx wiki

5.3 Marketplace Architecture

Describe the general archietecture where a owner can offer nodes in multiple markets.

Talk about the micro-services

5.3.1 Lease Management Service

Talk about the lease management service

5.3.2 Trading Platform

Talk about the trading platform

5.3.3 Bid-Offer Matcing Engine

Talk about how auctions will take place

5.3.4 Accounting Engine

Describe the role and properties of accounting engine

5.3.5 Contract Management Service

Talk about the contract management service. Life-cycle of the contract service.

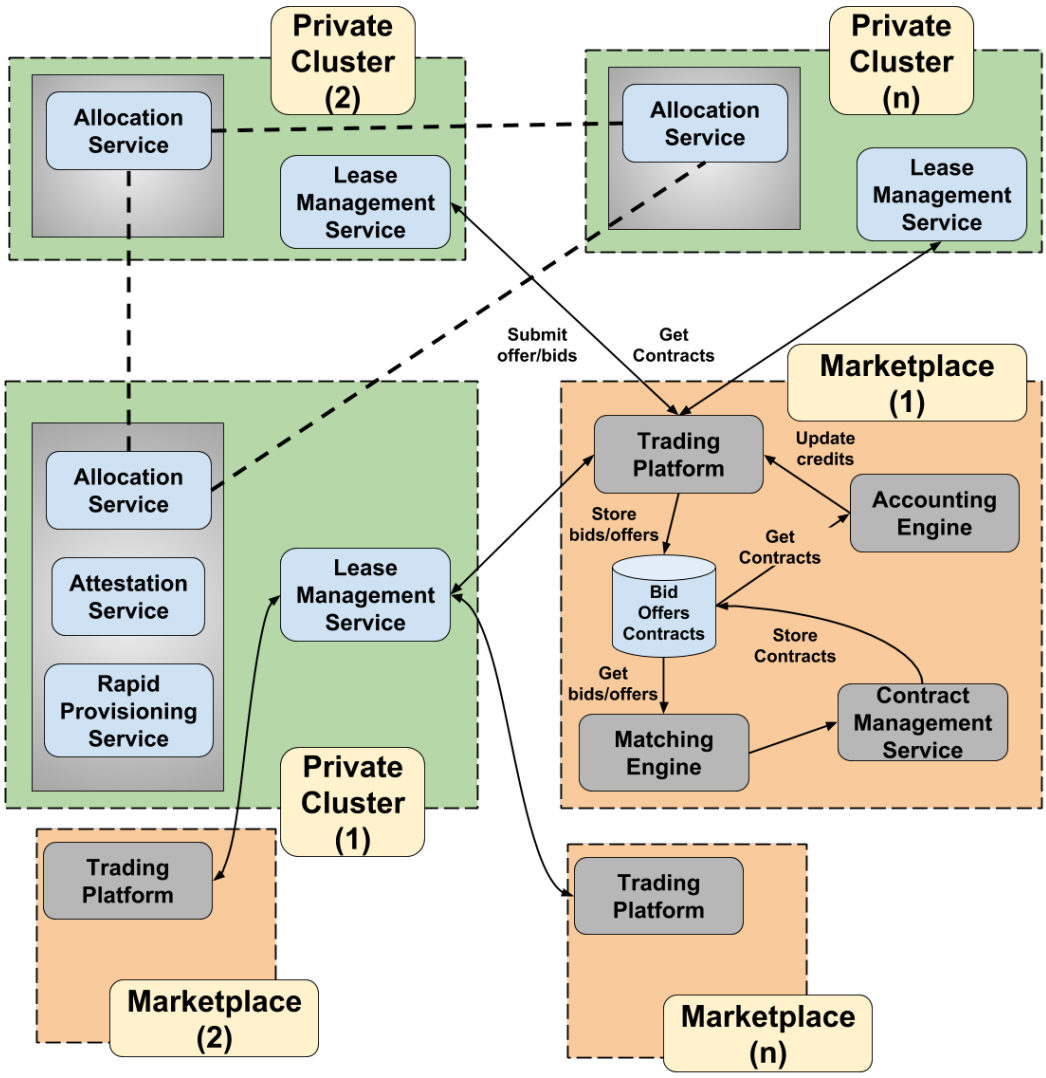

Figure 5·1: Marketplace architecture

5.4 Related Work

- Differentiate with central scheduler(intractable), fair-share scheduler(high cost of multiplexing and congestion), hybrid schedulers(Mesos, Borg, Boinc, Tycoon)
-

Chapter 6

Works that needs to be done

6.1 Things to do

- Build the set of micro-services which will collectievely function as a marketplace. (end of summer 2019)
- Require a solution for providing network overlays between HIL owned by different entities. (End of September 2019)
- build a 2nd prize based auction engine to map demands (bids) with supply (offers) (Need a timeline)
- Install a prototype: eg 45 nodes with 15 nodes belonging to separate group, each having its own network isolation layer and provisioning system, each trading nodes with each other. (By October 2019)
- Finish the simulator that will mimic the FLOCX system (By November 2019)
  - Move nodes between the clusters
  - Use the auction engine to determine server placement
  - Validate the simulator with the prototype
  - Run traces from different clusters.
- Run experiments (time to be decided)
- Write thesis

Chapter 7

Conclusions

7.1 Summary of the thesis

YEAR	2014	2015			2016			2017			2018			2019	
Semester	F	S1	S2	F	S1	S2	F	S1	S2	F	S1	S2	F	S1	S2
RCE								X							
Req 1 & 2 (HIL)	X	X	X	X	X	X	X								
Req 3 (BMI)				X	X	X	X	X	X	X	X				
Req 4 (Bolted)						X	X	X	X	X	X	X	X	X	
Req 5 & 6 (Marketplace)											X	X	X	X	X
Prospectus															
Simulations															
Experiments															
Thesis Writing															
Defense															

Appendix A

Proof of xyz

This is the appendix.

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

[9] Anderson, D., Hibler, M., Stoller, L., Stack, T., and Lepreau, J. Automatic online validation of network configuration in the Emulab network testbed. In *Int'l Conf. on Autonomic Computing* (June 2006).

[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] Andrea Righi, Ari Jort, Austin Gonyou, Ben Spade, Brian Elliott Finley, Curtis Zinzilieta, Dann Frazier, Denise Walters, Greg Pratt, Mastaler, J. R., and Josh Aas. SystemImager. http://systemimager.org/.

[12] Appavoo, J., Uhlig, V., and Waterland, A. Project kittyhawk: Building a global-scale computer: Blue gene/p as a generic computing platform. *SIGOPS Oper. Syst. Rev. 42*, 1, 77–84.

[13] Averitt, S., Bugaev, M., Peeler, A., et al. Virtual computing laboratory (VCL). In *Proceedings of the International Conference on the Virtual Computing Initiative* (2007), pp. 1–6.

[14] 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.

[15] 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.

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

[17] 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`.

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

[19] Chase, J. S., Irwin, D., Grit, L. E., Moore, J., and Sprenkle, S. Dynamic virtual clusters in a grid site manager. In *Int'l Symp. on High Performance Distributed Computing* (2003).

[20] Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (New York, NY, USA, 2017), SOSP '17, ACM, pp. 153–167. `http://doi.acm.org/10.1145/3132747.3132772`.

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] Gaggero, M., and Zanetti, G. *HaDeS: a Scalable Service Oriented Deployment System for Large Scale Installations*. Consorzio COMETA, Feb. 2009. 251-257.

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] 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`.

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

[34] 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.

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

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

[37] 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.

[38] 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.

[39] Omote, Y., Shinagawa, T., and Kato, K. Improving agility and elasticity in bare-metal clouds. In *ASPLOS* (2015).

[40] OpenStack Keystone Project. `https://wiki.openstack.org/wiki/Keystone`. `wiki.openstack.org/wiki/Keystone`, Aug 2015.

[41] 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.

[42] Peterson, L., Anderson, T., Culler, D., and Roscoe, T. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev. 33*, 1 (Jan. 2003), 59–64.

[43] 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.

[44] 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.

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

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

[47] 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.

[48] 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.