HIL: Designing an Exokernel for the Data Center

Jason Hennessey
Boston University
henn@bu.edu

Sahil Tikale
Boston University
tikale@bu.edu

Ata Turk
Boston University
ataturk@bu.edu

Emine Ugur Kaynar
Boston University
ukaynar@bu.edu

Chris Hill
Massachusetts Institute of Technology
cnh@mit.edu

Peter Desnoyers
Northeastern University
pjd@ccs.neu.edu

Orran Krieger
Boston University
okrieg@bu.edu

Keywords cloud computing, datacenter management, bare metal, exokernel, IaaS, PaaS

Abstract

We propose a new Exokernel-like layer to allow mutually untrusting physically deployed services to efficiently share the resources of a data center. We believe that such a layer offers not only efficiency gains, but may also enable new economic models, new applications, and new security-sensitive uses. A prototype (currently in active use) demonstrates that the proposed layer is viable, and can support a variety of existing provisioning tools and use cases.

1. Introduction

There is a growing demand for mechanisms to simplify deployment of applications and services on physical systems, resulting in the development of tools such as OpenStack Ironic, Canonical MaaS, Emulab, GENI, Foreman, xCat, and others [10, 12, 22, 53, 55].

Each of these tools hides the low-level hardware features (IPMI, PXE, boot devices, etc.) from applications and provides higher-level abstractions such as image deployment, Juju charms, Puppet manifests, etc.—much like an operating system hides disks and address spaces behind file systems and processes. 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.

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.

We believe that there is a need for 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 describe our design of such a layer, called the Hardware Isolation Layer (HIL), which adopts an Exokernel-like [19] 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.

SoCC '16, October 05 - 07, 2016, Santa Clara, CA, USA.

ACM ISBN 978-1-4503-4525-5/16/10. . . $15.00.
DOI: http://dx.doi.org/10.1145/2987550.2987588

One major advantage of an Exokernel approach is simplicity. Our current prototype is less than 3,000 lines of code and is sufficiently functional that we use it in production on the Massachusetts Open Cloud (MOC)[1], in a 576-core cluster. Our current usage of HIL is limited to flexibly allocating infrastructure for staging efforts and research as needed; however, the potential value of the HIL approach is much larger. We envision HIL being used to:

- move resources between multiple production clusters in a data center, enabling resources to be moved to match demand, and enabling new models for supporting HPC applications at Cloud economics,
- enable a marketplace model for data centers [11, 17], providing the interface between hardware providers and users of this hardware, who are in turn free to adopt whichever provisioning tool or software deployment mechanism that best fits their needs,
- enable application specific provisioning services; imagine, in a general purpose data center, deploying an isolated 1000 node cluster in seconds to solve an interactive supercomputing problem such as was shown on specialized hardware by project Kittyhawk [7],
- enable security-sensitive applications that cannot take advantage of public cloud for security or compliance reasons (e.g., HIPAA, military, finance) to be deployed in the middle of a public cloud, and
- enable new scheduling services to move resources between all the services in a data center based on demand, economic models, or to meet power regulations [50].

The contributions of this paper are: (1) definition of a data center isolation layer that enables improved efficiency and utilization of resources, (2) definition of an architecture for realizing this layer and demonstration of a prototype implementation sufficiently functional for production use, (3) analytical and experimental evaluation of the prototype implementation that demonstrates its performance characteristics and value for a number of use cases, and (4) discussion of the change required to different provisioning tools to exploit this new layer.

Section 2 describes use cases HIL is designed to support; Section 3 provides a high-level view of the HIL functionality and approach; Section 4 discusses design requirements, Section 5 describes the technologies used to implement HIL, and Section 6 discusses the architecture and implementation. In Section 7 we present measurements of HIL in use, then in Sections 8, 9, and 10 discuss related work, future directions, and conclusions.

[1] HIL is being developed as part of the *Massachusetts Open Cloud* (MOC), a collaboration between the Commonwealth of Massachusetts, research universities, and an array of industry partners, to create a non-profit Infrastructure as a Service public cloud [2].

2. Physical provisioning use cases

Although hardware virtualization has been widely successful in the data center, there still remain many applications and systems which benefit from, or even require, "bare-metal" hardware.

Cloud/virtualization software: This is one of the most obvious cases, as these systems *provide* virtualized resources rather than *consuming* them. These systems are often enormously complicated to deploy, and come with their own provisioning system (e.g. Mirantis Fuel, Red Hats Open Stack Platform Director).

Big data platforms: Environments such as Hadoop have traditionally been deployed on physical resources because of the impact of virtualization on these workloads [20, 58]. More recently the use of on-demand virtualized deployments [34] such as Amazon EMR [4] and IBM BigInsights [30] has grown, due to their cost-effectiveness and convenience for smaller deployments. In search of higher performance, cloud vendors like IBM [48], Rackspace [44], and Internap [31] have recently started to provide on-demand bare-metal big data platforms.

HPC Systems: Many HPC applications achieve significantly higher performance on physical rather than virtualized infrastructure. Research institutions typically have large shared HPC clusters, deployed on statically configured infrastructure, and in many cases there is more demand than there is infrastructure to satisfy that demand. These sytems could usefully exploit any idle cycles on other data center infrastructure if those cycles could be made available to them.

Although efficient, large shared clusters present problems for highly specialized communities; for example, an informal survey of two national clusters shows that they support only about 10% of the tools typically used by the neuroscience community [26]. Such specialized users would be better served if they could stand up their own HPC clusters, perhaps for brief periods of time.

Latency-sensitive applications: For certain applications the higher variation in latency in a virtualized environment may lead directly to reduced performance. Examples of this include tightly-coupled numeric computations [21], as well as certain cyber-physical and gaming applications. In this case the use of dedicated hardware allows extreme measures to be taken such as disabling most sources of interrupts and installing special-purpose operating systems [47].

Security and compliance: The use of separate hardware may be required due to regulatory or contractual confidentiality requirements, or may be useful in providing enhanced security for sensitive data.

Staging and research: Physical hardware may also be required, typically for brief periods, to enable staging/testing of services before production deployment, and research that requires either deterministic results or specialized access to hardware. Despite heavy fluctuation in demand, these use cases are typically handled by statically allocating a fixed set of resources to handle them.

3. HIL overview

HIL is designed to manage server resources in a multi-tenant data center, providing a fundamental allocation and isolation layer that allows reconfiguration of physical resources without manual administrative effort. HIL takes a minimalist, "exokernel"-like approach, providing isolation between allocated pools of un-abstracted physical nodes requiring only a small amount of code to be trusted to provide the basic services. Most of the functionality which users depend on remains in the provisioning tools like Ironic and MaaS. 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 we will discuss, many tools can be used unmodified while others require only modest modifications. As demand grows, the user can allocate additional nodes from the free pool; when demand shrinks, they may be released for other use.

In addition to the HIL core services, we have developed a number of optional services that can be deployed for use cases that require them. Examples include *1)* a *virtual private network (VPN)* service to allow users to connect to HIL-managed networks, *2) node control* services, allowing users to safely use out-of-band management interfaces for tasks such as power-cycling nodes and selecting boot devices, and *3)* an *inventory service* for tracking characteristics of resources that may be allocated.

A large data center may contain multiple server pools, clusters, and compute services managed by different companies, institutions, or research groups. For example, the MGHPCC[2] data center hosts multiple single-institution research computing clusters, shared HPC clusters, shared IaaS and PaaS Cloud environments, and other specialized compute and storage environments stood up by various research groups. Today each of these environments exists in a separate "silo" of physical infrastructure.

[2] www.mghpcc.org, the Massachusetts Green High-Performance Computing Center, a modern 19-MW data center build by a consortium of MIT, Harvard. Boston University, Northeastern, UMass. This data center houses hundreds of thousands of CPU cores, 10s of petabytes of storage, and provides production services to a user base of over 10,000 researchers working in a wide range of research areas from neuroimaging to physics to digital humanities.

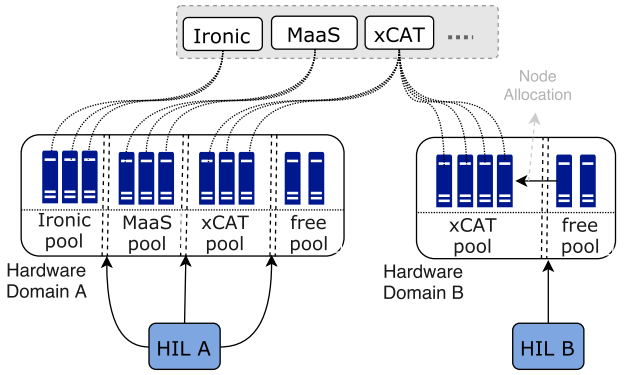

Figure 1. HIL provides strong network-based isolation between flexibly-allocated pools of hardware resources, enabling normally incompatible provisioning engines (e.g. Ironic, MaaS, xCAT) to manage nodes in a data center.

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. For example, in our current environment, in addition to the production HIL deployment at Northeastern[3] we have additional (experimental) HIL deployments at MIT and BU. Services like the MOC or compute clusters shared by multiple institutions can span these HIL instances, as shown in Figure 1. While some configuration and networking is required, little or no changes are needed to the underlying provisioning services (e.g. Ironic, xCAT in the illustration) which interact directly with the allocated hardware.

4. Requirements

If HIL is to be a fundamental layer in the data center, it must support both:

1. large long-lived production services (e.g., Clouds, shared HPC clusters, shared Big Data clusters) where HIL can be used to move resources between the different production environments as demand grows and shrinks. For example, in the MGHPCC we intend to grow and shrink the MOC IaaS environment depending on demand, and use any resources made available when demand is low to augment the HPC clusters in the data center.
2. short-term on-demand deployment of clusters (both small and large) for e.g. research, staging, on-demand Big Data platforms, etc., where HIL can be used to provision physical machines almost as simply as IaaS services may be used to provision virtual ones.

These require the following:

[3] or, rather, Northeastern's space at the MGHPCC

Control: Users should have full control over physical machines, including all the capabilities they use to debug an installation when there is a failure. This includes installing their own operating system, accessing a machine's console, power cycling the machine, and controlling the boot device.

Security: User security should be similar to that with user-purchased hardware. Users must be protected, or have the capability of protecting themselves from: 1) other concurrent users, 2) previous users using the same infrastructure, and 3) to the extent possible, the provider of the HIL service itself.

Interconnectability: A user should be able to federate nodes acquired from multiple HIL deployments. At the same time, the administrators of each HIL deployment should be able to address issues such as hardware failures and upgrades or firmware updates without coordination with other HIL operators.

Software provisioning: Users should be able to provide their own software provisioning services. Many large (e.g. institutional) users may have extensive investments in their in-house provisioning systems, as well as strong dependencies on those systems. Researchers may have need of services like Emulab or GENIE. Developers (in particular, the MOC development team) may need to use third-party installation mechanisms like Mirantis' Fuel [38] (for installing OpenStack) or OpenStack Ironic [53]

Compatibility: Users should be able to use existing software (OSes, schedulers, identity systems, and provisioning systems) within HIL with little or no effort. In addition, node and network configurations deployed without HIL should be able to be easily brought under HIL control. It would, as an example, be a major barrier to HIL adoption in the MGHPCC if one of the member institutions needed to bring down its entire production HPC cluster in order to incorporate it into a HIL environment.

Simplicity: Simple self-service use cases should be addressed simply. While IT departments may use sophisticated tools to create complex platforms, self-service by small-scale users will require a simple user experience.

5. Enabling technologies

In order to implement HIL and achieve these requirements, the key technologies used are:

IPMI: The Intelligent Platform Management Interface (IPMI) [16] allows remote management and monitoring of a node independent of a running OS. It allows users to power-cycle a node, select a boot device, and access a node's serial console.

Although important for remote management, IPMI requires isolation due to a number of security issues, including known default credentials and authentication weaknesses [15]. At present IPMI is typically accessed over a separate network interface, via a network accessible only to trusted administrators.

Because of these weaknesses, while HIL uses IPMI functionality itself and provides a subset of that functionality to its users through its API, it must prevent direct IPMI access in order to prevent malicious users from compromising systems.

PXE: The Preboot eXecution Environment (PXE) allows an OS to be loaded and booted over the network, using the Dynamic Host Configuration (DHCP) and Trivial File Transfer (TFTP) protocols, and is supported on virtually all computer systems with a network interface.

The DHCP protocol uses a client broadcast to request configuration information (e.g. IP address, default gateway, DNS server) from a configuration server; additional information may specify a boot file to be downloaded from a specified server via TFTP and executed. The lack of authentication in either protocol, as well as the use of broadcast in DHCP, allows malicious nodes to spoof server responses and even force nodes to boot into a malicious OS.

Because of these risks, only trusted users may be allowed to gain direct physical access to the provisioning network. In typical compute clusters this is done via OS security, by ensuring only trusted administrators have access to the privileges (root) needed for physical network access. For HIL, however, network mechanisms must be used to isolate the provisioning networks used for different tenants.

VLAN: Virtual LANs allow multiple virtual layer 2 networks to be created over a single switching infrastructure. Packets are tagged with a 12-bit VLAN ID, and switch rules are used to restrict a particular VLAN ID to a certain subset (i.e. set of ports) of the infrastructure. Other more flexible mechanisms (e.g. the UDP-based VXLAN protocol) are becoming available, however at present typically only VLAN support can be guaranteed across a large heterogeneous infrastructure. HIL depends on the isolation provided by VLAN support; however as discussed in Section 9, for scalability HIL will in the longer term need to take advantage of newer protocols.

6. HIL Architecture

The HIL architecture is shown in Figure 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,

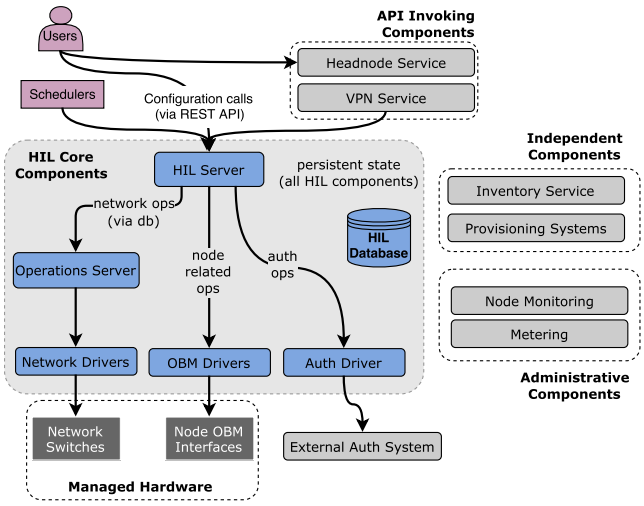

Figure 2. Components in a HIL deployment.

Table 1. Objects defined in HIL

Object name	Description Of Object
Project	Logical entity that can be assigned a set of user reconfigurable resources.
User	User of a HIL project.
Network	Medium by which nodes communicate within and across projects; is qualified by a type such as Ethernet.
Node	Physical, user-controlled server assigned to a project.
NIC	Compute Node's physical connection that can then be assigned to a network.
Switch	Physical entity to which NICs are connected and which can serve Networks.
HIL Service	Service providing the HIL REST API atop one or more physical or virtual server instances containing the state of the system. Carries out privileged isolation operations that partition resources.

- 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.

We begin our description of the architecture with an overview of the API, followed by a description of the individual components.

6.1 HIL API

The key objects in the HIL API are shown in Table 1. The basic container for resources in HIL is the *project*, which is managed by a set of *users*. Projects contain *nodes* and *networks*: nodes represent compute nodes, and are atomically allocated by assignment to a project; each node has a set of *NICs* which may be connected to networks.

Networks can either be private to a single project—e.g. used for the purpose of isolating management access–or public and accessible by all projects, as in the case of external internet access.

Operations that an end-user can perform are:

- create new projects and delete empty projects,
- create/delete networks in a project, and import public networks into a project,
- allocate/release nodes to and from projects,
- connect NICs to / disconnect NICs from networks,
- node management: power cycling and serial console access, and
- queries such as listing free nodes, nodes and networks assigned to a project, or node and network details.

Administrators may additionally manage topology and the authentication and authorization of HIL by creating or deleting users, adding users to projects, creating public networks, registering or removing nodes, NICs and switch ports, and connecting NICs to switch ports.

6.2 Core Components and Drivers

The *core* components are those which are used in all projects, and may not be overridden by user-provided functionality: these 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: this 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:

- 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. [c.f. § 4, *control*] 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 avail-

able: one uses simple database tables, while the other forwards requests to OpenStack Keystone [42], providing access to multiple authentication backends as well as token-based delegation of authorization.

Operation Engine: This is responsible for sequencing and coordinating operations which change network configuration. It receives requests from the API server via a queue, performs them in the order they were submitted and finally updates the database upon completion. In this way the database always represents the state after the most recent successful operation, simplifying the task of determining the legality of incoming requests and preventing failures from introducing undefined states. 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. In this way, the Operation Engine is a mitigation against degenerate scenarios where a user could briefly access the network of a previous or future user caused by certain race or failure conditions.

Serializing all network operations in this way may present a scalability challenge in the future. When or if that becomes an issue, we expect that introducing even rudimentary dependency analysis into the Operation Engine, for example only serializing requests with respect to a particular project, will allow HIL to scale by servicing requests in parallel.

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 [§ 4 *security*] while allowing explicit connection to external network services [§ 4 *provisioning*] or even other HIL deployments [§ 4 *Interconnectability*].

6.3 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 [§ 4 *compatibility*], in turn greatly increasing the utility of the cluster being managed.

Node access (headnode) services: In the simplest self-service usage, a user allocates a set of nodes and then deploys an operating system and applications of the user's choice on those systems. [§ 4 *simplicity, provisioning*] Core HIL functions may be used for allocation, establishing external connectivity, and rebooting, but a network boot server is needed for actual software deployment. The *headnode* service allocates a virtual machine connected to project-private networks, external user access via a pre-configured SSH key pair, and a pre-configured boot server to which the user may upload images.

In other cases it may be desirable to bypass the headnode provisioning service to a greater or lesser extent. [§4 *compatibility*] As one example, some distributed software applications (e.g. Mirantis Fuel [38], Red Hat Foreman [28]) base their installation mechanism on a pre-packaged network boot server; in this case the headnode service may be used to install this first node, which then deploys software to the remainder of the allocated nodes. In another case an institution may have a pre-existing software provisioning system; here core HIL functions may be used to connect the allocated nodes to this system[4].

Schedulers: Core HIL provides mechanisms for allocating, isolating, and connecting nodes; it does not prescribe policy, even to the extent of e.g. selecting which nodes to allocate. For small deployments this functionality may be sufficient, especially in combination with careful use of node naming. In more complex installations, however, users may need to to allocate nodes from specific pools, or for certain periods of time make reservations for the future. Cluster schedulers such as SLURM [56] or GridEngine [25] are typically used for this purpose in existing large installations, and could be readily adapted to provide these features on top of HIL.

VPN service: HIL provides a shared VPN service which may be used to connect project-private networks to external provisioning services [§ 4 *compatibility*], to establish connectivity between project networks on different HIL deployments [§ 4 *interconnectability*], or to connect to networks outside of a HIL deployment. Providing VPN as an optional HIL service not only simplifies interoperability by making use of a preconfigured VPN template, but may enable efficiencies due to sharing of resources, particularly if specialized resources such as hardware VPN accelerators (e.g. Juniper [40]) are available.

Node metadata / inventory service: While HIL maintains a list of available nodes and small amounts of configuration information in its database, it does not provide any description of the capabilities of those nodes (e.g., cores, memory, accelerators, network bandwidth) or administrative attributes such as ownership. Some of this information may be configured into a scheduler (e.g. in definitions of different pools of resources and their characteristics); the remainder is expected to be addressed by a separate inventory service. Such service could be something as simple as a wiki or more comprehensive like Clusto [14].

[4] Researchers working on improved software deployment mechanisms will no doubt wish to override the default service, as well.

6.4 HIL Administrative Services

The final HIL components are services provided to the administrators of a HIL deployment, rather than the users.

Node Monitoring: Detecting and managing failures is a significant concern in any large hardware installation, and there are a number of systems and approaches adopted by organizations managing such installations today. As an example, organizations participating in the MOC and deploying HIL to date make use of a variety of node monitoring systems, which in many cases rely heavily on IPMI access to detect node failures, memory errors, disk failures, and other adverse events. In order to allow continued use of such systems [§ 4 *compatibility*] HIL allows administrators to configure external access to IPMI for these existing monitoring systems.

Billing and Metering: Some environments in which HIL would be deployed will have existing mechanisms for metering and billing, whether providing full billing and chargeback or "showback", for instance usage reports which may be used for accounting or just for verification of equitable usage. HIL provides access to log records detailing resource allocation over time, allowing it to support existing billing mechanisms [§ 4 *compatibility*][5].

6.5 Other Requirements

The two remaining requirements from Section 4 are Security and Interoperability.

Interoperability: Rather than implementing a mechanism to support interconnectivity between deployments, HIL again takes a minimal approach. For simple use cases, a VPN-as-a-service mechanism allows users to join projects in separate HIL instances. In more complex cases users of HIL may deploy their own (possibly hardware-assisted) networking technologies [57] and services to interconnect projects across different instances of HIL. As an example, we are developing modifications to OpenStack to create and control overlays between projects.

Security: Tenants of a HIL service must be protected, or have the capability of protecting themselves, from concurrent tenants using disjoint resources as well as previous or future tenants of the same resources. In addition, when possible it is desirable to protect the tenant from the operator of the HIL infrastructure as well.

HIL performs strict network isolation of users' networks, in particular guarding networks used for DHCP and network boot services, to prevent spoofing and other attacks. In addition insecure services needed by tenants—IPMI, in particular—are proxied to prevent their use by malicious users in attacks on other tenants.

An additional aspect of security in a bare-metal environment is the possibility of attacks by users of the same hardware but separated in time. Previous users may attempt to change system or peripheral firmware, compromising future users; protection from this relies on hardware support to protect against unauthorized updates or a design that reduces or eliminates location of processor-accessible firmware (e.g. like AMD Seamicro or possibly HP MoonShot [39]). Sensitive information may be left resident in DRAM or on disk when the system is released to another user [27]; protection against this risk requires scrubbing RAM and use of disk encryption (either host- or disk-resident) to allow discard of on-disk data.

In contrast to virtualized approaches, HIL provides some degree of protection from the administrator, as all software on the node is owned and installed by the tenant. With the use of a user-specified provisioning system it provides more protection than Emulab or OpenStack Ironic, both of which place the entire software provisioning stack under provider control.

[5] Alternately other software packages accepting call detail records may be adapted for this use.

Table 2. Production HIL statistics.

Operations	Total
Create project	1973
Delete project	1960
Allocate node	3194
Free node	3145
Allocate network	1972
Free network	1958
Connect NIC	3505
Detach NIC	3441
Total operating time	13 months

7. Evaluation and Experience

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. A portion of the MOC's production Open Stack cloud is stood up on this cluster, and all experiments described in this section were obtained by using this HIL prototype. Table 2 presents the production HIL usage statistics we observed on this cluster during the last 13 months.

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.

Table 3. 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

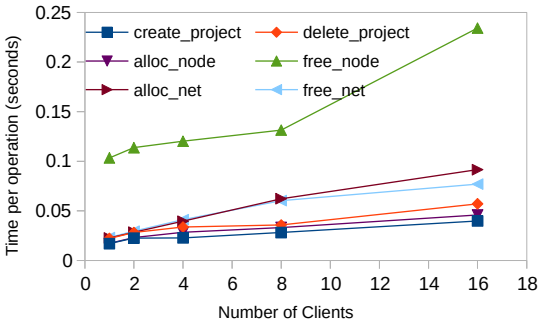

Figure 3. Scalability of HIL synchronous operations

7.1 Metrics and Scaling

The runtime of key HIL operations is shown in Table 3. 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.

Figure 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 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.

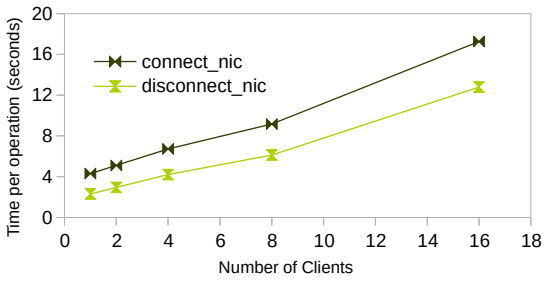

Figure 4. Scalability of HIL asynchronous operations

Table 4. 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

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, and verification, along with repeated user prompts and timeouts. In Table 4 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

$$\rho = \frac{N \cdot T_{net\ ops}}{T_{net\ ops} + T_{up} + T_{post}} \quad (1)$$

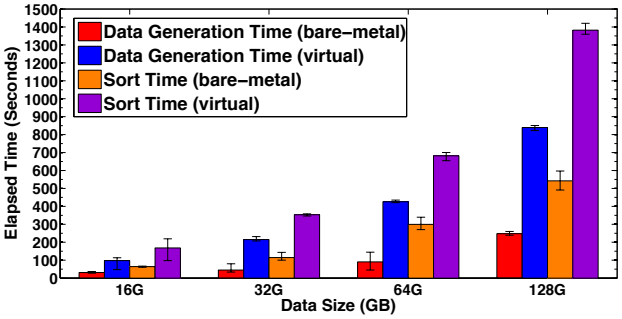

Figure 5. Performance of bare metal and virtualized Hadoop environments running on top of HIL carved deployments.

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[6] 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.

7.2 HIL as a bare metal service

To demonstrate the use of HIL as a bare-metal service, we used it to allocate a complete, isolated BigData environment. Steps included: 1) allocating 8 nodes through HIL, 2) using Foreman to provision them with Red Hat Enterprise Linux 7.1, 3) deploying a bare-metal Big Data processing environment using Apache BigTop [23], and 4) testing the performance of this environment using the basic Sort application within the CloudRank-D benchmark [36]. We compare this performance with the virtualized case by running the same experiment on the MOC's OpenStack (Kilo) cloud running on (idle) nodes in the same cluster.

In Fig. 5, we display the performance of the bare-metal and virtualized Big Data environments we tested[7]. The tests performed in here are the standard random write and sort benchmarks that come with Hadoop. Note that our goal in here is not to provide a thorough comparison of bare-metal and virtualized Big Data environments but rather to showcase that HIL can easily support environments such as cloud management and Big Data systems. To approximate an apples-to-apples comparison, we ran the virtual Hadoop environment with one VM per node, where each VM was provided 90% of the available resources of its host[8]. As seen in Fig. 5, bare metal outperforms the virtualized version in terms of performance, possibly due to overheads associated with virtualization.

7.3 HIL as a provisioning service enabler

As discussed earlier, there is a growing interest in bare-metal provisioning systems. However, existing bare-metal solutions can introduce long provisioning delays while installing and configuring the OS and applications to local disks. This has motivated recent research developing sophisticated mechanisms to optimize provisioning large clusters using technologies not currently in the mainstream offerings, such as de-virtualization [41], broadcast/multicast [6] or bittorrent [24].

We believe that the HIL model is ideal for enabling, testing and deploying these novel models. We recently had success using HIL to develop a novel network-mounting-based on-demand bare-metal big data provisioning system [51]. Our approach fast-provisions a big data cluster by booting servers from an iSCSI boot drive containing the OS and applications and uses the local drives for application data, enabling a BigData cluster to be allocated and ready within 5 minutes. This service compliments the HIL model by allowing nodes to be allocated and ready to use in short time.

7.4 Experience with HIL

We have deployed HIL on the 48-node cluster described at the beginning of this section, where it is being used on a daily basis. We have been successful in integrating it with several applications with small or no modifications. We present our experiences here.

Foreman: This is a system for provisioning, configuring, and monitoring servers, used by Red Hat for their supported configurations, and by our team on a daily basis. Foreman is easily installed on the headnode VM in a project, and then may be used to manage the allocated nodes in the same way as in non-HIL installations.

Canonical Metal As A Service (MaaS): This is a system for elastically provisioning bare-metal resources; it provides provisioning but not (to date) isolation. The initial non-HIL deployment of MaaS had a steep learning curve, requiring two weeks of work by a junior team member. After this, the same person needed 30 minutes to configure access to this external provisioning service for new HIL allocations.

OpenStack: To date this has been the primary production use of nodes allocated using HIL. By using HIL we are able to easily create multiple non-interfering OpenStack clouds for development, testing and staging, while provisioning them

[6] expect.sourceforge.net

[7] Reported results are average of 5 runs.

[8] Remaining resources were reserved for Hypervisors.

using existing mechanisms (e.g. Foreman, Fuel) which are supported by external partners.

Ironic: is an OpenStack project that allows bare metal nodes to be deployed using the same OpenStack API used for managing VMs. It requires access to a power-control API like IPMI in order to perform common operations like power on and reset. To integrate this, a simple driver was written which passes power operations to the HIL API as well as another driver to enable OpenStack's networking component to recognize a HIL network.

ATLAS: is a collaborative research initiative in physics with a heavy HPC component[1]. As part of a successful proof of concept, we explored with the ATLAS personnel at Boston University (BU) the potential for transitioning idle nodes in our HIL cluster into the "Tier 2" HPC grid. This was accomplished using a VPN to bridge BU's custom HPC provisioning and management to a privately allocated network.

7.5 Integration into an existing scheduler environment

HIL does not perform *scheduling*, i.e. selection of which nodes should be allocated to a particular request. The SLURM scheduler, widely used in HPC clusters, provides a rich variety of mechanisms and policies for allocating systems in response to user requests. Here we present a design for integrating HIL into SLURM, allowing existing (and well-understood) policies to be used for selecting nodes to be allocated.

This integration is based on SLURM resource reservation requests, which allow users to request resources as varying as compute nodes and licenses, allocating some quantity of resource for a period of time. A special namespace is defined for HIL requests, allowing HIL-specific rules and triggers to be defined. On receipt of a request, SLURM first allocates the specified number of nodes from the pool available for bare-metal allocation, and then executes a *reservation start* trigger which requests allocation and isolation of these nodes by HIL. On expiration of the reservation, a *reservation end* trigger returns the HIL allocation, returning their network configuration to the default state.

To illustrate the integration of HIL into a production SLURM environment we describe two projects where we are actively integrating HIL and SLURM.

7.5.1 Supporting a big-data systems experiment

In this example custom resource provisioning is used to support a database systems project that is experimenting heterogeneous database paradigms to support diverse data sources [18, 49]. The project is exploring the heterogeneous big-data paradigms in health-care [46] and uses an MIT Lincoln Laboratory developed software environment [45] as its core platform for deploying high-performance scalable data analytics tools. The platform used requires full control of the software stack and system operations for the duration of the experiment.

To support this work a SLURM resource reservation is requested with a name in the HIL namespace. The start of this reservation triggers a HIL request for the allocated nodes, providing isolation for the nodes' provisioning network. After the resource reservation expires, a cleanup event triggers HIL actions to restore nodes to their base network connectivity, as well as launching steps to restore nodes to their base SLURM state.

7.5.2 Supporting bursty Windows-based applications within a SLURM cluster

In this example end users of our SLURM cluster have workflows that involve a high-throughput cell image processing [35] that is tightly integrated into a Windows environment. The processing is part of a pipeline that integrates with other cluster resources and executes on a regular schedule. We can use HIL isolation again triggered from specially named SLURM resource reservations. In this case reservations are scheduled for particular time windows that align with associated physical experiment pipeline needs. Following this approach provides dynamic flexibility under user control to isolate and reconfigure parts of the SLURM based system, depending on experiment schedules and scale.

7.6 An economical incentive analysis

We consider the use of HIL by a hypothetical data center operator, allowing resources to be shifted between multiple uses. Consider a datacenter serving Amazon's Elastic Computing Cloud (EC2) service, where excess computing power not rented as on-demand instances is made available for spot purchasing. Using HIL, it is possible to elastically divide the set of nodes in the datacenter such that some portion of this excess is rented to users with Real-Time applications (in particular game server usage) or HPC workloads.

Massively multi-player online games (MMOGs) are generally served over dedicated distributed server farms [37], as they have very dynamic usage patterns and highly specialized software. Similarly, many HPC libraries and applications are designed to work closely with the hardware, requiring specific OS drivers and hardware support. Although HPC cloud offerings are available, they are based on dedicated HPC-specific clusters.

With HIL it is possible to dynamically adjust physical nodes between virtualized and non-virtualized services according to demand. We very conservatively estimate the overhead associated with moving a server (e.g. from an HPC allocation to a cloud allocation) as one hour, primarily due to the need for restart and software re-provisioning.

In Figure 6, we provide the number of EC2 c4.xlarge servers that we estimated to be required to serve the real-time workload described by a set of game server logs (MineCraft, Sep. 19-25, 2011 [32]) and an HPC workload (Sun Grid Engine logs from TU Delft ASCI, Feb. 22-29, 2005 [9]) for

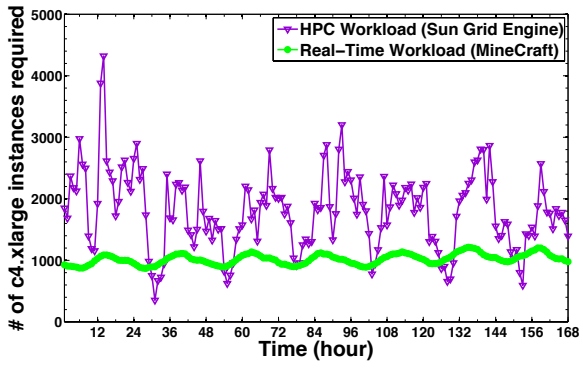

Figure 6. # of c4.xlarge instance equivalents required by sample Real-Time and HPC workloads for one week.

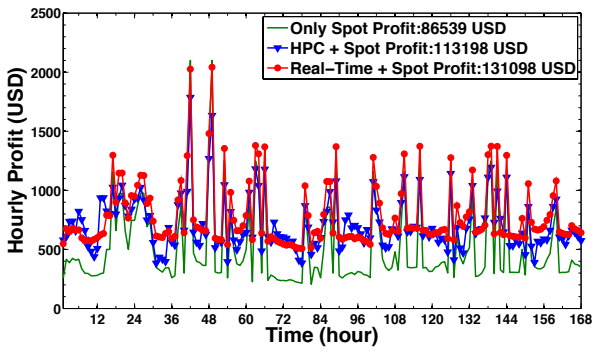

Figure 7. Weekly profit from a system that can host 5000 linux, c4.xlarge EC2 instances located in US-East-1a.

one week. Each point in the circle-ended green line and the triangle-ended purple line indicates the number of c4.xlarge instances required for serving the online game MineCraft and the Sun Grid Engine HPC load in the respective hour. The estimated c4.xlarge instance counts are computed by directly taking the active MineCraft server counts, and dividing the HPC CPU demands by four, as c4.xlarge instances have four Amazon vCPUs, which are roughly equivalent to four hyperthreading cores.

In Figure 7, we try to depict the weekly profit an AWS-like vendor can make assuming that it has 5000 linux, c4.xlarge EC2 instances that it cannot rent on-demand and has to rent via the spot market. We assume that these instances are served over 5000 physical nodes. The triangle-ended purple line indicates the hourly profit that this vendor can make by renting all of these extra 5000 instances over the spot market. The spot market prices used for the c4.xlarge instances in this figure are obtained from Amazon for July 13-20, 2015.

The straight green line indicates the hourly profit that this vendor can make if all the nodes are rented in the spot market. The triangle-ended blue line indicates the hourly profit that this vendor can make if some portion of the nodes are rented as bare-metal nodes to the HPC workload depicted in Figure 6 and the remaining nodes are virtualized and rented in the spot market. The hourly price of an HPC instance is taken from reported AWS HPC on-demand costs. The circle-ended red line indicates the hourly profit that this vendor can make if some portion of the nodes are rented as bare-metal nodes to the Real-Time workload depicted in Figure 6 and the remaining nodes are again virtualized and rented in the spot market. For the Real-Time workload, we assume that the hourly cost of a node is the bare-metal renting price of a 3.4 GHz Xeon E3-1270 server with 8GB of RAM from IBM SoftLayer.

As seen in Figure 7, mixing Real-Time or HPC services along with virtualized services provides higher profits for the cloud provider, even with the overhead caused by switching state. e.g., in this example, ∼30% more profit is obtained by the HPC + Spot combination and ∼50% more profit is obtained by the Real-Time + Spot combination, compared to a pure Spot case. We should note that due to high variability of the server demands of HPC workloads, the HPC profits suffer from the provisioning overhead. This shows that the ability provided by HIL that lets us mix virtualized workloads with bare-metal workloads is invaluable.

8. Related work

The Hardware Isolation Layer model brings the philosophy of resource isolation without abstraction to the data center. Exokernel first applied this idea to segmenting the resources of a single machine; enabling highly customizable per-application library OSes and demonstrated performance advantages for specific applications [19]. Applying this model to the data center is simpler; placing nodes on isolated networks is easier than developing models to partition memory, compute and storage. HIL similarly enables specialized per-application provisioning systems.

NoHype [33] also partitions the physical resources of a machine, but exploits virtualization hardware support to run any operating system. The fundamental goal is to avoid security issues associated with increasingly complex hypervisors. HIL, similarely, is very simple, with a very small trusted computing base. NoHype like solutions may be attractive in a HIL environment in environments without sufficient hardware support to protect firmware from malicious guests, see 6.5.

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 [5, 6, 8, 10, 12, 13, 22, 24, 41, 43, 53, 55]. The sheer diversity of these projects suggest that it is unlikely that any one system will solve all use cases, and there is value in a HIL-like layer to alow 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 [7], 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.

9. Future Work

HIL is an evolving platform, and a number of important enhancements are under development or in design at present.

HIL provides network isolation via 802.1Q VLANs; however the limited VLAN namespace will pose problems at full data-center scale. Isolation based on the VXLAN protocol will provide a much larger namespace, with no change to the existing switch driver model. More scalable alternatives such as Layer 3 isolation and SDN-based flow-level isolation via a custom switch controller are under investigation.

In Table 4 we see that pre-OS node boot times vary greatly, taking over two minutes for some platforms; since provisioning nodes may require multiple reboot cycles, moving resources between projects may incur high costs on such systems. Work is underway on a *fast provisioning system*, where nodes boot from (and shut down to) a minimal boot-loader environment optimized for fast network booting [52, 54].

Although HIL is currently in use by our own team, for it to succeed it must be broadly deployed across a variety of use cases. It is challenging to convince administrators of heavily used production services to deploy HIL. We are currently in two proof-of-concept collaborations with other teams in the MGHPCC: ATLAS, where we are using idle HIL nodes to augment their HPC cluster on-demand. The second is an incremental deployment of HIL in a subset of the Engaging-1 [29] cluster, allowing resources to be moved on demand between the HPC cluster and the MOC.

Security in systems managed by HIL is in part dependent on node security: if malicious users are able to compromise systems at a low (i.e. firmware) level, this may be leveraged into attacks on any future users of the same resources. Although progress has been made in some hardware areas (e.g. disk firmware [3]), features such as local IPMI access common on commodity servers continue to present risks. We have just started a project with the USAF, MIT LL and TwoSigma to (1) extend HIL to meet federal security, isolation, and other compliance requirements, and (2) deploy it in the MOC with hardware that meets these same requirements. This infrastructure would be available for federal, state, and municipal agencies when needed and otherwise would be made available to industry segments having these same security needs.

Another HIL-related project is developing a Mesos-like scheduler to move resources between services using HIL to increase utilization. It is based on a distributed model where each service is responsible for determining its load and resources requirements; work to date has demonstrated significant increases in efficiency.

Finally, we are developing an architecture for enabling multiple networking providers to offer differentiated services between HIL-controlled PODs [57], where providers may differ in bandwidth, latency, SLA, price, security, and other attributes. In this model, users can determine which flows will use which networking provider, allowing choice of hardware resources in not only compute nodes but in the intra-data center network as well.

10. Conclusions

In this paper we have introduced HIL as a fundamental isolation layer for the data center. HIL is designed to be a layer under different hardware and virtual provisioning systems, enabling strong isolation between the different services while reducing the burden of deploying new services and enabling resources to be moved between services for efficiency. We have shown that HIL can be used for bare-metal self-service deployment of a Hadoop environment. We have discussed integration into the SLURM scheduler as well as richer self-service bare metal deployment tools such as MaaS, Ironic, and Emulab. We also presented results of a trace-driven economic simulation showing the value of moving capacity between different services. We described the architecture of HIL and a current realization of that architecture. The Massachusetts Open Cloud (MOC) uses HIL in its production deployment, in use by over one hundred users on a daily basis since January 2016.

Acknowledgements

We gratefully acknowledge Ian Denhardt and George Silvis III for their significant contributions to HIL. Also, Yue Zhang, Apoorve Mohan, Ravisantosh Gudimetla and Minying Lu for their assistance in the evaluation. We are also grateful to the many other contributors to HIL, including Zhaoliang Liu, Ryan Abouzahra, Jonathan Bell, Jonathan Bernard, Rohan Garg, Kyle Hogan, Andrew Mohn, Kristi Nikolla, Abhishek Raju, Ritesh Singh, Ron Unrau and Valerie Young. Thanks to Cisco and Dell for providing the servers used as part of the evaluation.

Partial support for this work was provided by the MassTech Collaborative Research Matching Grant Program, National Science Foundation awards 1347525 and 1149232 as well as the several commercial partners of the Massachusetts Open Cloud who may be found at http://www.massopencloud.org.

References

[1] ATLAS. https://atlas.web.cern.ch/Atlas/Collaboration/.

[2] Massachusetts Open Cloud. http://www.massopencloud.org/.

[3] Maximize Security, Lock Down Hard Drive Firmware with Seagate Secure Download & Diagnostics. Technology paper TP684.1-1508US, Seagate Technology Llc, August 2015.

[4] Amazon. Amazon Elastic MapReduce (Amazon EMR). https://aws.amazon.com/elasticmapreduce/, 2015.

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

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

[7] Jonathan Appavoo, Volkmar Uhlig, and Amos Waterland. Project kittyhawk: Building a global-scale computer: Blue gene/p as a generic computing platform. *SIGOPS Oper. Syst. Rev.*, 42(1):77–84.

[8] Sam Averitt, Michael Bugaev, Aaron Peeler, et al. Virtual computing laboratory (VCL). In *Proceedings of the International Conference on the Virtual Computing Initiative*, pages 1–6, 2007.

[9] Henri Bal and the DAS-2 team. The grid workloads archive: GWA-T-1 DAS2. http://gwa.ewi.tudelft.nl/datasets/gwa-t-1-das2.

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

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

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

[13] Jeffrey S Chase, David Irwin, Laura E Grit, Justin Moore, and Sara Sprenkle. Dynamic virtual clusters in a grid site manager. In *Int'l Symp. on High Performance Distributed Computing*, 2003.

[14] Clusto. Clusto cluster mangement tool. github.com/rongoro/clusto, last accessed 11 Aug 2015.

[15] Dan Farmer. All the IPMI that's fit to print. http://fish2.com/ipmi/, June 2014.

[16] Dell, HP, Intel Corporation, and NEC Corporation. Intelligent platform management interface. http://www.intel.com/content/www/us/en/servers/ipmi/ipmi-specifications.html.

[17] Peter Desnoyers, Jason Hennessey, Brent Holden, Orran Krieger, Larry Rudolph, and Adam Young. Using Open Stack for an Open Cloud Exchange (OCX). In *Int'l Conf. on Cloud Engineering*, March 2015.

[18] A Elmore, J Duggan, M Stonebraker, M Balazinska, U Cetintemel, et al. A demonstration of the BigDAWG Polystore System. *Critical care medicine*, 8(12), 2015.

[19] Dawson R Engler, M. Frans Kaashoek, and James Otoole. Exokernel: an operating system architecture for application-level resource management. In *ACM SIGOPS Operating Systems Review*, SOSP '95, pages 251–266, New York, NY, USA, 1995.

[20] Eugen Feller, Lavanya Ramakrishnan, and Christine Morin. On the performance and energy efficiency of Hadoop deployment models. *BigData Conference*, pages 131–136, 2013.

[21] Kurt B Ferreira, Patrick Bridges, and Ron Brightwell. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *ACM/IEEE Conf. on Supercomputing*, 2008.

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

[23] Apache Software Foundation. Apache Bigtop. http://bigtop.apache.org.

[24] Massimo Gaggero and Gianluigi Zanetti. *HaDeS: a Scalable Service Oriented Deployment System for Large Scale Installations*. Consorzio COMETA, February 2009. 251-257.

[25] W. Gentzsch. Sun Grid Engine: towards creating a compute power grid. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001. Proceedings*, pages 35–36, 2001.

[26] Yaroslav O Halchenko and Michael Hanke. Open is not enough. Let's take the next step: an integrated, community-driven computing platform for neuroscience. *Frontiers in neuroinformatics*, 6, 2012.

[27] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, et al. Lest we remember: Cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98, May 2009.

[28] Red Hat. Introduction to Foreman OpenStack Manager. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux_OpenStack_Platform/4/html/Installation_and_Configuration_Guide/chap-Foreman_Overview_and_Installation.html. available from access.redhat.com/documentation, Oct 2015.

[29] Christopher Hill. Engaging-1 Facility Notes, January 2014. github.com/christophernhill/engaging1.

[30] IBM. Ibm biginsights for apache hadoop. www.ibm.com/software/products/en/ibm-biginsights-for-apache-hadoop, 2015.

[31] Internap. Bare-metal agileserver. http://www.internap.com/bare-metal/, 2015.

[32] Alexandru Iosup and the PDS group. The grid workloads archive: GTA-T7. http://gta.st.ewi.tudelft.nl/datasets/gta-t7/.

[33] Eric Keller, Jakub Szefer, Jennifer Rexford, and Ruby B. Lee. Nohype: Virtualized cloud infrastructure without the virtualization. In *Proceedings of the 37th Annual International Sym-*

posium on Computer Architecture, ISCA '10, pages 350–361, New York, NY, USA, 2010. ACM.

[34] Jeff Kelly. Hadoop-nosql software and services market forecast, 2014-2017. http://wikibon.com/hadoop-nosql-software-and-services-market-forecast-2013-2017, 2014.

[35] Michael R Lamprecht, David M Sabatini, Anne E Carpenter, et al. Cellprofiler: free, versatile software for automated biological image analysis. *Biotechniques*, 42(1):71, 2007.

[36] Chunjie Luo, Jianfeng Zhan, et al. Cloudrank-d: benchmarking and ranking cloud computing systems for data processing applications. *Frontiers of Computer Science*, 6(4):347–362, 2012.

[37] Rich Miller. WoW's back end: 10 data centers, 75,000 cores. http://www.datacenterknowledge.com/archives/2009/11/25/wows-back-end-10-data-centers-75000-cores/, November 2009.

[38] Mirantis. Mirantis OpenStack Reference Architectures. docs.mirantis.com/openstack/fuel/fuel-5.1/reference-architecture.html, Aug 2015.

[39] TP Morgan. HP Project Moonshot hurls ARM servers into the heavens. *The Register, November*, 1, 2011.

[40] Juniper Networks. Junos Pulse Application Acceleration Service. Technical Report 1000286-005-EN, Juniper Networks, October 2011. available from fr.security.westcon.com.

[41] Yushi Omote, Takahiro Shinagawa, and Kazuhiko Kato. Improving agility and elasticity in bare-metal clouds. In *ASPLOS*, 2015.

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

[43] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Comput. Commun. Rev.*, 33(1):59–64, January 2003.

[44] Rackspace. Rackspace cloud big data onmetal. http://go.rackspace.com/baremetalbigdata/, 2015.

[45] Albert Reuther, Jeremy Kepner, William Arcand, et al. LLSuperCloud: Sharing HPC systems for diverse rapid prototyping. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6. IEEE, 2013.

[46] Mohammed Saeed, Mauricio Villarroel, Andrew T Reisner, Gari Clifford, Li-Wei Lehman, George Moody, Thomas Heldt, Tin H Kyaw, Benjamin Moody, and Roger G Mark. Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II): a public-access intensive care unit database. *Critical care medicine*, 39(5):952, 2011.

[47] Dan Schatzberg, James Cadden, Orran Krieger, and Jonathan Appavoo. A Way Forward: Enabling Operating System Innovation in the Cloud. In *USENIX Conf. on Hot Topics in Cloud Computing*, HotCloud'14, pages 4–4, Berkeley, CA, USA, 2014. USENIX Association.

[48] Softlayer. Big data solutions. http://www.softlayer.com/big-data, 2015.

[49] Michael Stonebraker and Ugur Cetintemel. "one size fits all": An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 2–11, Washington, DC, USA, 2005. IEEE Computer Society.

[50] Ata Turk, Hao Chen, Ozan Tuncer, Hua Li, Qingqing Li, Orran Krieger, and Ayse K Coskun. Seeing into a Public Cloud: Monitoring the Massachusetts Open Cloud. In *USENIX Workshop on Cool Topics on Sustainable Data Centers (CoolDC 16)*, Santa Clara, CA, 2016. USENIX Association.

[51] Ata Turk, Ravi S. Gudimetla, Emine Ugur Kaynar, Jason Hennessey, Sahil Tikale, Peter Desnoyers, and Orran Krieger. An experiment on bare-metal bigdata provisioning. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, 2016. USENIX Association.

[52] Ata Turk, Ravi S. Gudimetla, Emine Ugur Kaynar, Jason Hennessey, Sahil Tikale, Peter Desnoyers, and Orran Krieger. An Experiment on Bare-Metal BigData Provisioning. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, 2016.

[53] Devananda Van der Veen et al. Openstack ironic wiki. https://wiki.openstack.org/wiki/Ironic.

[54] R. Wartel, T. Cass, B. Moreira, E. Roche, M. Guijarro, S. Goasguen, and U. Schwickerath. Image Distribution Mechanisms in Large Scale Cloud Providers. In *Int'l. Conf. on Cloud Computing Technology and Science (CloudCom)*, pages 112–117, November 2010.

[55] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, et al. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, December 2002.

[56] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, number 2862 in Lecture Notes in Computer Science, pages 44–60. 2003.

[57] Da Yu, Luo Mai, Somaya Arianfar, Rodrigo Fonseca, Orran Krieger, and David Oran. Towards a network marketplace in a cloud. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, Denver, CO, June 2016. USENIX Association.

[58] ZDNet. Facebook: Virtualisation does not scale. http://www.zdnet.com/article/facebook-virtualisation-does-not-scale/, 2011.