Strategies to Improve Operation and Usage of Bare-Metal Clusters
(Thesis Proposal)

Apoorve Mohan

Department of Electrical and Computer Engineering
College of Engineering
Northeastern University

1 Overview

Although virtualized infrastructure can satisfy the requirements of various organizations, most performance and security-sensitive organizations demand to run their workloads directly on bare-metal servers[1]. Such organizations are not willing to tolerate the performance unpredictability or security risks (e.g., side-channel attacks) due to co-location, and performance overhead or vulnerabilities due to a large code base of the complex virtualization stack. Moreover, organizations setting up virtualization-based cloud platforms (to offer public services or private use), or running workloads that require direct and exclusive access to hardware components that are either difficult to virtualize (such as InfiniBand, RAID, FPGAs, GPUs), also require bare-metal access.

To run their workloads, such performance and security-sensitive organizations invest enormous sums of money to buy or rent infrastructure, and set-up and manage bare-metal clusters.

It is essential for organizations to efficiently operate and use these bare-metal clusters in order to maximize their returns on the investment. However, despite the intense research in the area of bare-metal cluster operability and usage, there exist the following issues that lead to inefficient operation and usage of bare-metal infrastructure:

- **Imaging and Elasticity:** Existing general-purpose bare-metal provisioning techniques deploy the intended software stack to the local storage present on the bare-metal server. Provisioning bare-metal servers to the local storage of bare-metal servers slows down the ability to rapidly add/remove bare-metal servers to a cluster.

- **Introspectability:** Provisioning bare-metal servers to the local storage requires running an agent/client on them to introspect the deployed software stack. However, the integrity of the introspection agent/client remains questionable. Moreover, the complexity of managing these agents/clients is linearly correlated to the scale of bare-metal servers in the cluster.

- **Aggregate Resource Usage:** Today, organizations host (co-locate) multiple bare-metal clusters running heterogeneous workloads in their data centers, which are known to exhibit

[1] A bare-metal server is a physical server designed to run dedicated services. The operating system is installed directly on to the server, eliminating layers and delivering better performance. Organizations run dedicated servers (i.e., cluster of servers) in their private data centers or a shared data center, or rent them from a service provider.

heterogeneous resource utilization patterns. These resource utilization patterns tend to complement each at various time intervals and provides an opportunity for short-term bare-metal multiplexing of bare-metal servers between these co-located clusters for improving the aggregate resource usage for the organization. Traditionally, the co-located bare-metal clusters are siloed[2] from each other, which makes it hard to exploit the complementary utilization patterns via short-term bare-metal multiplexing and improve the aggregate resource usage.

- **Job Co-scheduling:** Job co-scheduling is common in throughput-oriented batch clusters. However, co-scheduling jobs that are not compatible, i.e., that contend for the same system resources can result in degradation. There have been several efforts that try to predict colocatable jobs based on offline profiling. However, this requires a priori knowledge about the applications. Furthermore, in the case of misprediction, existing batch schedulers cannot adapt to the situation.

Over the last decade, organizations have realized the advantages of having a data-driven enterprise, which has lead to a massive global increase in the number of computing facilities to process all that data. The surge in the number of data centers across the globe, thus calls for urgent resolution of the aforementioned bare-metal cluster operability and usage issues.

This thesis focuses on general-purpose and workload-agnostic strategies to improve imaging and elasticity, introspectability, aggregate resource usage, and co-scheduling of batch jobs in bare-metal clusters.

> **Thesis Statement:** Classically, diskless provisioning of bare-metal servers has been used to optimize for infrastructure cost, reduce server-level power consumption, and increase the reliability of compute-severs. This thesis demonstrates that diskless provisioning of bare-metal servers can as well be leveraged to improve imaging and elasticity, introspectability, and aggregate resource usage in bare-metal clusters. To further improve the efficiency of bare-metal clusters, this thesis proposes an adaptive co-scheduling mechanism for single-node batch jobs.

2 Background

This section reviews the existing bare-metal provisioning approaches (Section 2.1), different software introspection mechanisms, and systems (Section 2.2), how bare-metal clusters are set up and hosted in consolidated data centers (Section 2.3), and previous work on job co-scheduling in high performance computing (HPC) and throughput-oriented commodity data centers (Section 2.4).

2.1 Bare-Metal Provisioning

Bare-metal provisioning systems can be broadly classified into two categories, *diskful* and *diskless*, based on where the image is hosted after the bare-metal servers are provisioned.

Diskful provisioning systems provisions an intended software stack (i.e., the operating system and applications) on the local storage present of the bare-metal servers. The standard provisioning

[2] A bare-metal silo is a network-isolated partition (cluster) of bare-metal servers. A software stack dedicated for executing a particular workload is provisioned on the local storage of the bare-metal servers in a silo. The size of a silo (i.e. the number of bare-metal servers) is determined through careful capacity planning.

tools used in many bare-metal deployments are diskful. A rich set of open-source and commercial provisioning products such as Emulab [17], OpenStack Ironic [66], Crowbar [65], Canonical Metal-as-a-Service (MaaS) [20], Razor [68], and Cobbler [22] are available for automated diskful provisioning of bare-metal systems. Chandrasekar and Gibson [21] provide a comparative analysis of commonly used diskful provisioning systems.

Diskful provisioning systems can be further divided into two types. The first type of solution automates the manual installation process of the operating system and desired applications on the local disks (e.g., Foreman [31]). As they follow a step-by-step installation process, these solutions generally take the longest to provision.

The second type of solution copies a pre-installed image, containing the operating system and applications, onto the local disk over the network (e.g., OpenStack Ironic [81]). The size of such pre-installed images can be tens of gigabytes. Transferring them can overwhelm the network and making them persistent on the local storage of a bare-metal server still requires hundreds of seconds assuming standard hard disks are used.

Both solutions have a lower bound on the time before a server is ready to use, due to the need to reboot twice, once via the Preboot Execution Environment (PXE) to enter the installer, and once to boot into the freshly installed system. Moreover, when using diskful provisioning systems, re-provisioning a bare metal system requires formatting the local disks and then installing/copying the new system. If saving the existing disk state is desirable, the contents of the disk have to be copied away, which again requires hundreds of seconds and further increases the re-provisioning cost.

In general, diskful provisioning systems and the automation tools that employ these provisioning systems (e.g., Ironic, Canonical Metal-as-a-Service (MaaS), Foreman) are designed for setting up long-running bare-metal systems. They consider the high startup delays to be tolerable, assuming that the servers they provision will run continuously for long times. To support fast provisioning and reduce the boot time of diskful provisioning systems, Omote et al. [64] propose BMCast, an OS deployment system with a special-purpose de-virtualizable Virtual Machine Manager that supports OS-transparent quick startup of bare-metal instances.

A fundamental problem with all diskful provisioning systems, is that any modifications to the image are stored on the disks attached to the bare-metal server. This means, for example, that a user cannot easily release and re-acquire servers to match their needs since any state on the local disk is lost when the user releases the servers. Some of the rich functionality users take for granted in virtualized clouds is also not available in these environments. For example, a user cannot snapshot the disk of a physical server and then clone it to boot additional servers. Perhaps most importantly, if a physical server fails, the user cannot easily start up another server from the same disk image or use the disk to diagnose the failure; any state stored on the local disk of the server is inaccessible as long as the server is down. Moreover, in diskful provisioning systems, the local disk-hosted boot drives become a single point of failure. If the disk containing the boot drives fails, the bare-metal server becomes unusable until the disk is fixed or replaced and data recovery can be daunting in such cases.

On the other hand, *diskless provisioning systems* keep the provisioning image resident on a network-accessible remote logical disk that appears like a local disk to bare-metal systems. This method of provisioning historically has been used with diskless workstations [36, 46, 74] and HPC systems [25, 35, 69] to boot multiple servers from a single image. The Linux BIOS [55] proposes using Linux as a boot loader to network-boot other operating systems, but that approach requires making changes to the Linux kernel used as a BIOS. Furthermore, diskless provisioning is heavily used in virtualized systems [24, 49, 61].

2.2 Software Introspection Mechanisms and Systems

Introspection Mechanisms: Vulnerabilities are weaknesses that can be exploited by a malicious entity to perform unauthorized actions. Heartbleed (OpenSSL-based) [29], Shellshock/Bashdoor (Unix Bash shell-based) [33], and GHOST (Linux glibc-based) [40] are examples of the most infamous software vulnerabilities seen over the last decade. Various tools such as FlawFinder [83], RATS [70], ITS4 [82], and Foster [32] have been developed to detect software vulnerabilities. These tools employ techniques such as pattern matching, lexical analysis, data flow analysis, taint analysis, model checking, fault injection, fuzzing testing, etc., to detect vulnerabilities [43].

Malware is ill-intentioned software designed to conceal its identity and cause damage to the computer system that it runs on. Types of malware include viruses [75], rootkits [26], keyloggers [30], etc. Tools such as `chkrootkit` (for detecting the presence of rootkits) [4], Linux Malware Detect (Linux system scanner to detect threats in shared hosted environments) [9], and ClamAV (an antivirus engine for detecting trojans, viruses, etc.) [5] are examples of well-known malware detection tools. Such tools employ techniques such as anomaly-based detection, specification-based detection, and signature-based detection to identify the existence of malware in a system [39].

Introspection Systems: With the increase in the number, complexity, and code-base size of deployed software systems, the number of threats that can lead to security breaches has increased as well [54, 73, 84]. Thus, introspection has become a key requirement for organizational IT deployments [16,41,58,59,63,86,89] and large-scale introspection systems have been developed to address these requirements.

Introspection systems can be classified into two types: *Agent-based* and *Agentless* introspection systems. In agent-based introspection systems, the introspection agent/software runs on each server and the agent periodically executes the desired introspection mechanisms (e.g., ClamAV [5], chkrootkit [4], Linux malware detect [9], etc.) on the server to forward the introspection results to a centralized statistics collection system. Amazon Inspector [16], IBM BigFix [38], Symantec Endpoint Protection [23], and Tanium Threat Response [80] are examples of agent-based introspection systems.

Agent-based introspection has many drawbacks such as being amenable to intrusion and creating interference, and these issues have been previously reported in connection with production deployments [1]. For virtualized deployments, agentless introspection systems such as OpVis [63], Anchore [11], Clair [3], AquaSec [7], and Twistlock [8] have been proposed to overcome these drawbacks. These solutions exploit the non-intrusive capabilities available in virtualized/containerized systems that enable snapshotting of the target filesystems/images. Specifically, the open-source tool OpVis [63] snapshots virtualized instances at the host-level and introspects the snapshots using filesystem tree introspection techniques such as Columbus [57]. Anchore [11] and Clair [3] use image-scanning capabilities to introspect container images. AquaSec [7], and Twistlock [8] offer proprietary agentless container introspection solutions. These systems are designed for virtualized system introspection, whereas this work focuses on bare-metal system introspection.

There exist a few studies that focus on agentless introspection of bare-metal servers that are provisioned to Storage Area Network (SAN) targets. Banikazemi et.al. [18] proposes intrusion detection techniques at the SAN target level. Unfortunately, the introspection they do at the SAN-level leads to interference within the SAN controller's I/O path [15]. IDStor [15] avoids this problem through a network-based intrusion detection approach. It intercepts every iSCSI request between the server and the SAN target, re-creates the filesystem state identifying an inverse mapping between blocks and the inodes of files (external to the SAN), and introspects the re-created filesystem state. However, it cannot detect threats caused by software that has not yet been accessed by the server.

Moreover, IDStor approach is not sustainable as it requires developing special-purpose software to identify block-inode inverse mapping for every filesystem type. Unlike these approaches, the M2 design separates the image-control and introspection services and thus avoids interference. Furthermore, it can view the entire filesystem, and can detect vulnerabilities located in the unused parts of the image. Finally, M2 is not specific to SAN solutions, which require specialized hardware to operate.

2.3 Bare-Metal Siloing

Large organizations have realized the benefits of setting up consolidated data centers over separate, relatively smaller, computing facilities for each of its sub-organizations. The set of such organizations is not tiny, and it includes federal agencies, large corporations, academic institutions, etc. Utah Data Center (also known as the Intelligence Community Comprehensive National Cybersecurity Initiative Data Center) is an example of a compute facility shared by various intelligence agencies. Similarly, large corporations such as IBM, Microsoft, Google, etc. host sub-organizations/teams working on different products/services.

Typically, such organizations have complex hierarchical structures, where a top-level entity allocates an infrastructure budget to each of its sub-organizations, and the sub-organizations either set-up a dedicated bare-metal cluster, or rents infrastructure from another sub-organization offerings Infrastructure-as-a-Service on its bare-metal cluster.

The size of these bare-metal clusters is determined by the budget allocated to a sub-organization – which in turn depends on factors such as the value proposition of the workloads they host, historical utilization patterns, quality-of-service requirements of the hosted workloads, etc. The sub-organizations are required to optimize their allocated budgets. For example, the size of a cluster hosting a cloud-like service should be chosen to support peak utilization demands and not to turn down any incoming users – leading to low average utilization of the available bare-metal servers. On the other hand, a bare-metal cluster hosting a high-performance computing platform is usually limited by the allocated budget – thus resulting in long queues of jobs waiting for available resources to free up.

2.4 Job Co-scheduling

In HPC centers, people have intensively studied how to co-schedule separate jobs on the same CPU cores of a computer [19, 34, 42]. In particular, if one job emphasizes CPU-intensive computations, while a second emphasizes RAM-intensive computations (e.g., writing sequentially to a large in-memory array), and a third emphasizes I/O, there are substantial savings to be obtained by co-scheduling the three jobs to use the same CPU cores simultaneously.

Similarly, throughput-oriented commodity data centers are highly interested in co-locating jobs. This area of work has seen a recent resurgence due to the presence of many-core computers running a mixture of low-latency server applications (e.g., web searches, commercial transactions, etc.), and background "best-effort" jobs [27, 28, 45, 51–53, 77–79, 88, 90, 91]. In this case, the low-latency applications have top priority. Background best-effort jobs use available spare cycles but, are killed when the low-latency applications require additional CPU cycles.

Checkpointing has frequently been proposed and tested as a means of eliminating the need for a backfill queue in batch job queues [62]. However, such work takes care to use checkpointing only on a limited subset of the batch jobs. Liu et al. propose the use of checkpointing for improving turnaround times for batch queues [50].

Job scheduling for batch queues (resource managers) has been and continues to be intensively studied [34, 42, 48, 60, 62, 67, 71, 72, 76]. In particular, co-scheduling or co-location of jobs is also actively studied [19, 34, 42, 52].

The issue of co-scheduling or co-location is of special importance in modern data centers. For example, there have been a series of papers with authors from Google concerning co-location of latency-sensitive applications (such as web searches) with background "best-effort" computations [45, 51–53, 77–79, 88, 90].

3 Operation and Usage Problems in Bare-Metal Clusters

Despite the intense research, the following problems still exist due to the way bare-metal clusters are operated and used today:

3.1 Poor Operability

Imaging and Elasticity: Existing bare-metal cluster provisioning solutions install the operating system and application into the server's local storage. This installation process incurs long startup delays (tens of minutes to hours) and high networking costs to copy large disk images. Moreover, because user state is local to the server, these solutions lack rich functionality of virtual solutions including checkpointing/cloning of images, releasing and re-acquiring servers to match demand, and fast recovery from server failures.

A number of industry and research projects have attacked the performance and functionality challenges of provisioning bare-metal servers [14, 17, 20, 64–66, 68]; automating the bare-metal provisioning process, reducing the management overhead, and improving the performance of copying the image to the server's disk. For example, Omote et al. [64] proposed a lazy copy approach that copies the OS image in the background after the operating system is booted using a remote disk. While sophisticated techniques like this can reduce some of the user-visible provisioning time, all these approaches end up eventually transferring the boot image to the local disk, and hence still incur overhead to copy the image and have the functionality problems discussed above.

Introspectability: Traditionally, to ensure and verify the sanity of the deployed bare-metal servers, security solutions have been installed as agents inside systems [23, 38, 80]. These agents run just like any other application on the system while implementing their respective security functions. These agents periodically scan the file system, memory, and running processes and compare their findings against databases of known vulnerabilities[3] or malware, and send reports to the system administrator summarizing their findings. Over the decades, these practices of implementing security functions through local agents have become a standard in data centers.

[3] Non-profit organization such as Mitre Corporation [6], National Institute of Standards and Technology [10] maintain and constantly update open-source databases of vulnerable software and software configurations for public use.

This model of agent-based software introspection has three important shortcomings. *First*, security agents themselves may become vulnerable and lead to new attack vectors into one's system, as reported through a recent "DoubleAgent" attack that turns one's antivirus into malware or/and hijacks the system [13]. Therefore, there is an increasing need to ensure the integrity of the agents themselves and to sandbox their executions. *Second*, an agent needs to be installed separately on every new system, and thus their operational and maintenance cost increases linearly with the scale of the systems and becomes challenging at massive scale. Although there exist automation engines (e.g., Puppet [12], Ansible [2]) that can ease these maintenance overheads, these automation engines, in turn, require running their respective agents on the system. This brings us to the *third* point. While performing periodic security inspection, these agents consume system resources (e.g., CPU cycles, memory, network resources, etc.), which may impact the performance of the primary workloads that the systems are catering to. While the system may tolerate or avoid these overheads, this may, in turn, lead to performance instabilities or resource inefficiencies [87].

Aggregate Resource Usage: Consolidating different sub-organizations in a data center and setting up fixed-sized siloed bare-metal clusters for each of them leads to poor bare-metal resource efficiency on aggregate in such consolidated data center. The workloads running on these bare-metal clusters are known to exhibit heterogeneous resource demands (e.g., diurnal patterns, long job queues, etc.). However, statistical time-multiplexing of underutilized bare-metal servers is still not recommended – as the potential gains from such short-term multiplexing are marginal – leading to poor aggregate resource usage. This is primarily due to the high overhead to context switch the bare-metal servers between the distributed frameworks managing the workloads running on these bare-metal silos – due to high bare-metal (re)-provisioning overheads and the requirement for manual intervention in many cases.

3.2 Inefficient Resource Usage

Node-Level Usage: Batch-style many-core workloads for single-process jobs appear both in traditional HPC and in the Cloud (e.g., through IaaS). Yet application writers are notoriously poor at predicting how many CPU cores will be needed, and they often declare a requirement for more cores than they need. This happens especially when a job execution may pass through several phases, with different resource requirements in each phase – resulting in wasted CPU cycles.

Existing work in this area concentrates on predicting performance for a known workload with known characteristics in order to decide questions of co-scheduling (during batch job submission for HPC clusters [19, 34, 42]) or co-location (for corporate data centers, to co-locate known base applications with background "best-effort" jobs [45, 51–53, 77–79, 88, 90]). But both large HPC centers and the Cloud process diverse workloads whose characteristics are unknown in advance, In the real-world case of diverse workloads, previous systems cannot recover when they select a bad set of actively running jobs.

4 Proposed Research

4.1 Diskless Provisioning

It is proposed that bare-metal servers should be provisioned from images that contain the operating system and applications (i.e., remote-mounted boot drives). Upon boot, the operating system and application libraries are cached in the memory of the bare-metal servers, thus reducing traffic to

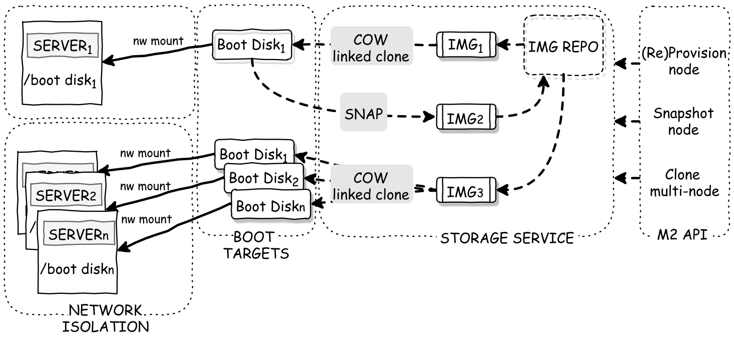

Figure 1: M2 conceptual design.

the mounted drives. Images of provisioned bare-metal instances are hosted on fast and reliable distributed storage.

Diskless provisioning enables to rapidly release and then acquire bare-metal servers – improving the elasticity of bare-metal clusters. It also allows rapid recovery of failed servers by booting another server with the remote disk of the failed server. Furthermore, using a fast and reliable distributed storage for hosting the bare-metal instance images enables transparent and fast snapshotting and cloning – easing the imaging process for bare-metal servers.

Next, diskless provisioning of bare-metal systems enables system-agnostic remote access to the live state of the bare-metal systems disk (i.e., installed operating system, application packages, configuration files, etc.). Using the seamless imaging capabilities achieved by diskless provisioning, the latest state can be snapshotted. The latest snapshot can then be mounted remotely and introspected – thus eliminating all the problems experienced by an agent-based introspection.

Lastly, the improved imaging and elasticity in bare-metal clusters enabled by diskless provisioning, in conjunction with a network configuration and isolation mechanism, and fault-tolerance mechanisms can reduce the overhead to context switch bare-metal servers between co-located clusters. With low context switching overheads, gains from short-term bare-metal multiplexing can be maximized – thus improving the aggregate resource usage in consolidated data centers.

4.2 Adaptive Co-Scheduling

Inspired by network congestion control mechanisms, an approach for adaptive co-scheduling of single-node batch jobs is proposed. The idea is to monitor the on-going performance of the co-scheduled jobs periodically, and dynamically adjust the co-scheduled set of jobs (i.e., remove, replace, add, or reduce jobs in the co-scheduled set) to improve the aggregate performance of the co-scheduled set. The performance of each co-scheduled job is monitored periodically by dumping the CPU performance counters. Jobs are suspended and resumed using an application-transparent checkpoint/restart mechanism to adjust the co-scheduled set. A side effect of our adaptive approach is that it leads to out-of-order micro-scheduling of the queued batch jobs, which may cause some jobs being starved for execution. The proposed approach cures this deleterious side-effect by incorporating an aging policy.

Unlike existing batch scheduling solutions, which either do not co-schedule jobs or try to predict the jobs that can be co-scheduled based on offline profiling, our approach continuously strives to adapt and achieve performant execution of the co-scheduled jobs – and improve the overall throughput of the batch cluster.

5 Approach

This section presents the approach chosen to address the bare-metal cluster operability and usage problems described in Section 3. Section 5.1 presents an approach to address the imaging and elasticity problem. Section 5.2 shows how the provisioned software stack on bare-metal servers can be introspected non-intrusively. Section 5.3 discusses a way to improve aggregate resource usage in data centers hosting bare-metal clusters running heterogeneous workloads, and Section 5.4 proposes to solve the job co-scheduling problem seen in batch clusters.

5.1 M2: Malleable Metal as a Service

5.1.1 Overview

This section proposes an operating system agnostic diskless provisioning mechanism for bare-metal servers which can improve imaging and elasticity of bare-metal clusters. Below is an overview of the features that are enabled by diskless provisioning of bare-metal servers and how they help in improving imaging and elasticity in bare-metal clusters:

Rapid provisioning: To improve elasticity, it is critical that a bare-metal provisioning system minimizes the startup overhead so that even short-lived deployments with life-spans of a few hours can use bare-metal servers efficiently. If servers take tens of minutes to deploy, the "effective" utilization of the data center resources will decrease.

Rapid snapshotting, re-purposing, re-provisioning: Ability to quickly snapshot the operating system and applications, releasing a server when unused, and being able to quickly provision a server using a previous snapshot. These features also enable the system to offer "elasticity" to the applications it hosts.

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

Support for multi-tenancy: Existing provisioning tools assume that all the hardware will be accessed by the same administrator, and the same entity manages all of the hardware available in the data center. However, in a shared data centers, the provisioning system has to ensure performance isolation and security across its users (or sub-organizations) during and after provisioning.

5.1.2 High-Level Design

Considering the list of features presented above, this section proposes Malleable Metal as a Service (M2), a diskless provisioning system for bare-metal clusters. This section presents a high-level design for M2:

In order to offer rapid provisioning, the use of diskless bare-metal provisioning mechanism is opted. By using diskless provisioning, M2 will not need to copy the entire image to the bare-metal server, and it can save the overhead to install images and applications to local disks once an image is prepared. M2 can rapidly start running applications by only fetching the necessary OS and application libraries before start-up, and further required packages will be fetched on-demand as they are used. Note that the standardization of technologies such as iSCSI has allowed diskless provisioning to be used with commodity servers and clients over any layer-3 network.

Furthermore, servers should be network-booted from images residing on a distributed storage. M2 can serve the images from centralized high-performance storage systems using multiple disks to improve boot time. Many modern distributed storage systems support capabilities such as *copy-on-write (COW)*, *de-duplication* and *linked-cloning* [85], which are beneficial for capabilities such as rapid cloning and snapshotting.

Note that in diskless provisioning, clients are always dependent on uninterrupted access to the centralized storage, and as the number of clients increases, the storage infrastructure has to be adequately scaled to support the increased load. Network connectivity and availability also plays a critical role in the performance of diskless provisioning systems. However, with the advances in faster, cheaper and redundant networking (e.g., Clos networks [37]) and storage solutions (e.g., Solid State Drives), datacenter applications increasingly lean towards disaggregating storage services to make full use of the capacity of their infrastructures [47]. Diskless provisioning approaches are very much aligned with this trend.

Figure 1 presents the conceptual design for M2 and some of the functionalities it offers. As seen in the figure, M2 stores images (user or M2provided) in an image repository. When a provisioning API call is made for a bare-metal server with a given image, a linked-clone of that image is created, followed by network isolation of the requested bare-metal server and mounting of the clone on the bare-metal server. When the server network-boots, it only fetches the parts of the image it uses, which significantly reduces the provisioning time. M2 also supports provisioning multiple servers in parallel from a single image by simply performing parallel provisioning calls.

As seen in Figure 1, API call of M2 for creating disk snapshots enables users to create restore-points by saving the current state of the image to the repository and tagging the saved image with a unique identifier. Using linked-cloning and COW, M2 can offer rapid snapshotting.

5.2 NiBi: Non-Intrusive Bare-Metal Introspection

5.2.1 Overview

Non-Intrusive Bare-Metal Introspection (NiBi) is a mechanism to introspect the deployed software stack of bare-metal servers without the need to run an agent on them. To achieve this at a cluster-scale, NiBi ensures that the following requirements are never violated:

Separation of Introspector and Introspected: The integrity of security the introspectors is critical. In order to establish confidence in their security assessment (about the introspected system), the introspectors should be impervious to compromises. In case of an agent-based system, the agents (i.e., the security introspectors) execute as just another application on the system, potentially alongside any malicious software and exposes themselves to compromise [13], doing more harm than good. Moreover, the agents could also transitively get influenced by the existing vulnerability in the system. For instance, some trojan malware could override the *ps* utility on the system to hide specific processes from reporting, and then any monitor using that utility to detect suspicious processes on the system will become ineffective [44]. Hence, it is vital that introspectors are separate from the introspected system, i.e., the integrity of the entity performing the security introspection should be verifiable independently from the introspected system.

Interference-less Introspection: Periodic introspection should not have an impact on the performance of the workloads running on the introspected systems. Agent-based solutions require the co-location of the agent with the running workloads. When the agent periodically executes, it contends with the running workloads for resources potentially causing jitter in the system and impacting the performance of the running workloads [1]. This impact can be mitigated by either (i) reserving exclusive resources for the agent on each server, or (ii) running introspection when the

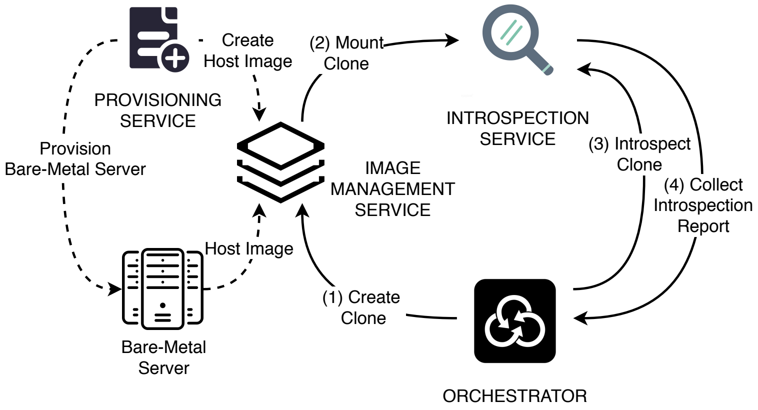

Figure 2: NiBi Design

server is under-utilized. However, both of these approaches have side-effects, since either (i) the reserved exclusive resources for the agent will remain idle when the agent is not running (leading to resource inefficiency), or (ii) there may be large time windows between introspections, and malicious agents can exploit this feature to avoid introspection. It is preferable to employ an introspection mechanism that does not have a performance impact on the workloads running on the introspected systems.

At-Scale Introspection: It is essential to keep the operational and maintenance cost associated with security assessment in bare-metal clusters to a minimum. In the existing agent-based model, a separate instance of security software is required to be installed on every server. The complexity of managing them is linearly correlated to the scale of bare-metal servers. Common-routine administrative tasks such as monitoring the health of agents, collection of assessment reports, and rolling of agent updates to all servers become challenging in this model as the scale grows. Furthermore, when operating at-scale, the networking between introspected systems and the reporting center becomes more complex as well. Therefore, there is a need for a solution wherein the bare-metal servers in the cluster can scale independently, and the cost of implementing their security introspection can be contained and managed efficiently.

5.2.2 High-Level Design

Given the list of requirements presented above, NiBi will be designed with a micro-service-based architecture for performing remote introspection of remotely provisioned[4] bare-metal servers. NiBi design includes four micro-services. Figure 2 presents the proposed NiBi design where the four services are: (a) Provisioning Service, (b) Image Management Service, (c) Introspection Service, and (d) Orchestration Service.

Provisioning Service: This service provisions bare-metal servers from the remote-mountable image disks made available by the Image Management Service. For newly provisioned systems, the service calls the Image Management Service to create a new host image, and then the bare-metal server is provisioned from that host image. A user can release a server, and later the user can restart the server by pointing it to the same image. This is called *re-provisioning.* If the system is being

[4] Provisioned over the network to a remote disk typically residing on a distributed file system.

re-provisioned, the Provisioning Service uses the existing image hosted at the Image Management Service for re-provisioning.

Image Management Service: This service hosts the remote-mounted images for bare-metal servers and provides APIs to snapshot or clone a provisioned bare-metal server's image rapidly. The remote provisioning service heavily relies on the image management service to perform its activities, but NiBi mainly uses the cloning capabilities of this service to clone the host image of the bare-metal server to be introspected.

Introspection Service: This service first mounts a clone created by the Image Management Service for the system to be introspected and then scans the mounted image to check for vulnerabilities. It maintains a database of known vulnerabilities. This database is populated and periodically updated by querying vulnerability databases (open- or closed-source based on availability). The introspection service can run rootkit analysis and/or software vulnerability analysis over the mounted volume. Note that introspection operations that need memory analysis are not immediately supported with this design.

Orchestration Service: This service controls the remote introspection workflow among the different components. It also offers an interface for users to introspect the bare-metal servers they own. Users can submit requests to this interface for introspecting the bare-metal servers they control. This allows them to control and change the introspection periodicity on demand, and hence, it allows them to control the cost of introspection of their systems.

As shown in Fig. 2, when performing a remote introspection, the Orchestration Service: (1) creates a clone of the provisioned bare-metal server's remote disk via the Image Management Service; (2) then mounts the filesystem present on the clone; and (3) invokes the Introspection Service to perform vulnerability analysis on the mounted image. Finally, (4) it collects the vulnerability analysis results and reports the data back to the user that initiated the introspection.

5.3 BareShala: Improving Bare-Metal Resource Efficiency in Consolidated Data Centers

BareShala aims to provide a scalable solution for improving aggregate resource efficiency for an organization consolidating its resources in a data center. To achieve this, BareShala proposes performing rapid multiplexing of bare-metal servers between the deployed clusters by the consolidating organization – where each cluster represent a sub-organization – based on ongoing cluster utilization.

Below are five fundamental design principles behind BareShala:

a) To aggregate resource inefficiency in a consolidated data center, the underutilized bare-metal servers that are allocated to a cluster are re-provisioning for a cluster that can benefit from additional resources. Bare-metal servers are prevented from oscillating between clusters during infrequent sharp utilization shifts using thresholds based on high-water and low-water mark strategies.

b) The co-located bare-metal clusters participating in BareShala can individually benefit by dynamically providing updated SLAs and constraints to BareShala as their infrastructure requirements or utilization are changed. Utilization statistics of a clusters' allocated servers' are also made available to BareShala. BareShalauses both these information to optimize from global resource efficiency.[5]

[5] Given the hierarchical structure in such consolidating organizations, each sub-organization optimizes their al-

Algorithm 1 Meta-algorithm (core algorithm for each phase): different policies produce different variations

```
while True do
    Initialize and measure ideal performance for any
        recently queued job(s) and place in the batch-pool
    Execute a specific policy based on
        current system state
    Start new production epoch
    while not end of production epoch do
        Periodically check for completed job(s)
        if any job(s) completed then
            Replace from pool of jobs
        end if
    end while
    # Production epoch finished; begin re-balancing
    Get performance of all running job(s)
    Process end of production epoch (re-balancing)
    Update system state (e.g., new phase)
end while
```

c) Each cluster must support preemption of its allocated bare-metal servers. Whenever a bare-metal server is preempted from a cluster, it is treated as a server failure by the distributed software framework running orchestrating the cluster. As part of their SLAs, clusters can specify its non-preemptible bare-metal server to the bare-metal multiplexing entity. With preemption in-place, BareShala can ensure that bare-metal servers are timely moved between clusters – thus ensuring cluster SLAs are not violated.

d) To reduce preemption overhead, bare-metal servers should be statlessly provisioned i.e. the software stack running on the bare-metal server is provisioned on a remote storage. Stateless provisioning removes the hassle to scrub the disk of the server to prevent any data theft, and speed-up the process to verify a servers integrity [56] – while ensuring performance[6].

e) A bare-metal server's network settings (at the networking infrastructure end) and out-of-band management are controlled programmatically – enabling BareShala to create isolation between non-trusting clusters, and ensure that no in-memory state from a previous cluster resides on the server before it is allocated to another cluster (by power cycling a server).

5.4 Batchpool: Recycling Lost CPU Cycles: Managing Multiple Many-core Jobs

5.4.1 Overview

This section proposes a model for co-scheduling multiple jobs on a many-core computer. The model is called Additive Increment/Full Decrease (AIFD). This model treats the execution of a job

located budgets by dynamically adjusting their SLAs. Whereas, the upper-level organization uses the utilization statistics and SLAs provided to multiplex bare-metal servers between the clusters of the sub-organizations to optimize for aggregate data center efficiency.

[6] Improved operating system caching mechanisms helps in achieving bare-metal performance. [?]

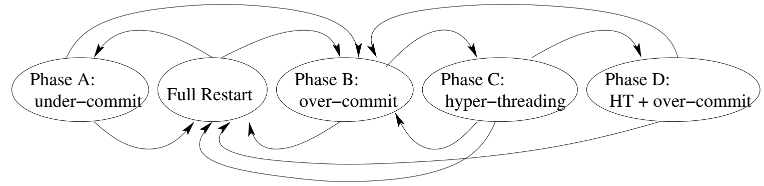

Figure 3: The five phases of AIFD ("Full Decrease" corresponds here to "Full Restart".)

through alternating phases of *production epochs* and short *re-balancing periods.* A *production epoch* is a fixed time period during which jobs are executed. At the end of execution, the re-balancing period AIFD: (i) acquires measurements of the performance (throughput) of the jobs that just executed; and (ii) makes decisions whether to start/re-start some jobs from the batch pool, stop (checkpoint) some currently running jobs, switch from one phase of AIFD to a different phase, and so on.

The details of the decisions made in the re-balancing period are what distinguish one AIFD phase from another. In particular, when throughput is measured in a re-balancing period, the actual throughput of the workload for that epoch is measured, and compared to an ideal throughput. A tunable threshold ratio of actual to ideal throughput determines if performance during the last epoch is considered good or bad. A "good" performance indicates that one may add another job from the batch pool into the currently running job mix. A "bad" performance indicates that one may "step back" to a previous job mix, or change to a different phase, or invoke a full decrease (i.e., begin again with an entirely new job mix chosen at random from the batch pool).

Next, is a meta-algorithm, Algorithm 1, which, in principle, can describe any of the four phases. The only difference in each of the phases is the policy chosen during re-balancing.

5.4.2 High-Level Design

The high-level design of an AIFD-based batch queue and execution of each of its four AIFD phases is presented below. Note that the system may transition among phases according to Figure 3.

If the system is beginning or has just completed a "Full Decrease" as part of AIFD, then a random subset of available jobs from the batch pool is chosen. Recall that the batch pool consists of checkpoint image files, and the system is either executing new jobs or restarting previously executing jobs.

AIFD departs from traditional batch queues in allowing for a variable level of over-commitment and a variable combination of jobs. After an initial production epoch, one must decide whether to attempt to raise the throughput or to fall back to a different combination of jobs (full decrease). If the ratio of actual throughput to ideal throughput over the last epoch is above a tunable threshold, then the over-commitment level is raised (an additional job is launched/restarted). If the ratio of actual to ideal falls below the desired threshold, then either one reverts to the previous epoch, or else a full decrease is incurred in the AIFD scheme. (See Figure 3.)

During a production epoch, each of the co-scheduled jobs is run toward the end of the production epoch (or toward completion if it completes during the epoch). A CPU performance counter for cumulative CPU instructions is measured both at the beginning and the end of a production epoch. (This is the only low-level hardware information used.) This allows one to determine the *actual throughput*, as opposed to the *ideal throughput* incurred if a job were to run in isolation.

Next, are the policies for the four AIFD phases.

Full Decrease: At the beginning of AIFD and in some instances during the execution of AIFD, there is no prior indication of what might be a set of compatible jobs to be co-scheduled. In such cases, a random set of jobs from the “batch pool” is selected as a new job mix. The total number of threads in the job mix must be less than or equal to the total number of CPU cores. (Informally, this is referred to as an over-commit ratio of 1.0.) After a full decrease, after the first production epoch at an over-commit of 1.0, AIFD will attempt to run in under-commit mode by removing one job at random from the current job mix. If the absolute throughput (irrespective of the ideal throughput) improves, then the system enters Phase A (under-commit). Otherwise, the job mix at over-commit of 1.0 is restored, and the system enters Phase B (over-commit).

Phase A (under-commit): Phase A is concerned with the case of an over-commit level less than 1.0. The number of executing threads is less than the number of CPU cores. (See Figure 3.)

Phase B (over-commit): The concept of a degradation ratio plays a vital role in phases B and D, where phase D refers to hyper-threading plus over-commit. During a re-balancing period, the total actual throughput of all co-scheduled jobs combined is compared against the total ideal throughput of all of the co-scheduled jobs. The result of this comparison determines either an increase or a decrease for the next production epoch. The ratio of the actual throughput to the ideal throughput determines a *degradation ratio* during the latest production epoch.

If the degradation ratio is larger than the configured degradation threshold (implying a small degradation), then an additive increment is chosen for the next production epoch, based on adding a new job from the batch pool at random. (Note that degradation ratios are less than one, and a degradation ratio of 1.0 implies no degradation from the ideal throughput. Hence, a larger degradation ratio implies smaller degradation.)

If the degradation ratio is smaller than the previously chosen degradation ratio (implying a substantial degradation), then a decrease is chosen for the next production epoch. The action taken during a decrease may consist of a “step back” or a “full decrease”.

Note that a given job mix will typically reach a “plateau” in which the over-commit level oscillates by attempting to add a random job, and then falling back to the previous job mix.

Phase C (hyper-threading HT): When, in Phase B, it is no longer possible to increase the level of over-commit, then an attempt is made to enable hyper-threading. Hyper-threading is turned on for the next production epoch. If the degradation ratio improves, then there is a transition from Phase C to Phase D (HT+over-commit). If the degradation ratio becomes worse, then the system reverts to Phase B. (Note that a standard technique of using Linux’s CPU core affinities is employed to enable or disable hyper-threading dynamically. The computer is configured to enable hyper-threading, and the DMTCP checkpointing package provides a plugin that can selectively set the core affinities of a process to the first half of the virtual cores (i.e., the physical cores), or all virtual cores.)

Phase D (HT+over-commit): After entering Phase C (hyper-threading: HT), the system immediately enters Phase D at the next production epoch. On each successive production epoch, if the degradation ratio is larger than the configured threshold, then an additive increase is triggered by adding a new job from the batch pool at random, similarly to Phase B. Similarly to Phase B, if the degradation is worse, then a “step down” (removal of one job), or else a “full decrease”, is triggered.

6 Schedule of Milestones

Anticipated completion dates of milestones in the proposed thesis work are given below:

1. **Imaging and Elasticity**: Done
2. **Introspectability**: End of October, 2019
3. **Aggregate Resource Usage**: End of January, 2020
4. **Node-Level Usage**: Done
5. **Thesis Proposal**: Beginning of September, 2019
6. **Thesis Writing**: First draft: End of February, 2020
7. **Thesis Defense**: End of March, 2020

References

[1] Amazon. summary of the october 22,2012 aws service event in the us-east region. https://aws.amazon.com/message/680342.

[2] Ansible is Simple IT Automation. https://www.ansible.com/.

[3] Automatic container vulnerability and security scanning for appc and docker. https://coreos.com/clair/docs/latest/.

[4] checkrootkit. http://www.chkrootkit.org/.

[5] Clam AntiVirus. https://www.clamav.net.

[6] Common Vulnerability Exposures. https://cve.mitre.org/.

[7] Container security - docker, kubernetes, openshift, mesos. https://www.aquasec.com/.

[8] Docker security and container security platform. https://twistlock.com.

[9] Linux Malware Detect. https://www.rfxn.com/projects/linux-malware-detect/.

[10] National Vulnerability Database. https://nvd.nist.gov/.

[11] Open source tools for container security and compliance. https://anchore.com.

[12] Puppet: Deliver better software, faster. https://puppet.com/.

[13] The clever 'DOUBLEAGENT' attack turns antivirus into malware. https://www.wired.com/2017/03/clever-doubleagent-attack-turns-antivirus-malware/.

[14] OpenQRM: http://www.openqrm.org/, 2018.

[15] Miriam Allalouf, Muli Ben-Yehuda, Julian Satran, and Itai Segall. Block storage listener for detecting file-level intrusions. In Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on, pages 1–12. IEEE, 2010.

[16] Amazon. Amazon Inspector. https://aws.amazon.com/inspector/.

[17] D.S. Anderson, M. Hibler, L. Stoller, T. Stack, and J. Lepreau. Automatic online validation of network configuration in the Emulab network testbed. In IEEE International Conference on Autonomic Computing, pages 134–142, USA, 2006.

[18] Mohammad Banikazemi, Dan Poff, and Bülent Abali. Storage-based intrusion detection for storage area networks (sans). In Mass Storage Systems and Technologies, 2005. Proceedings. 22nd IEEE/13th NASA Goddard Conference on, pages 118–127. IEEE, 2005.

[19] Major Bhadauria and Sally A McKee. An approach to resource-aware co-scheduling for CMPs. In Proc. of the 24th ACM Int. Conf. on Supercomputing, pages 189–199. ACM, 2010.

[20] Canonical. Metal as a service (MAAS), 2015.

[21] Ashok Chandrasekar and Garth Gibson. A comparative study of baremetal provisioning frameworks. Technical Report CMU-PDL-14-109, Parallel Data Laboratory, Carnegie Mellon University, 2014.

[22] Cobbler. Cobbler.

[23] Symantec Corp. Symantec Endpoint Protection. `https://www.symantec.com/smb/endpoint-protection`.

[24] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: High availability via asynchronous virtual machine replication. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, pages 161–174, USA, 2008.

[25] David Daly, Jong Hyuk Choi, José E Moreira, and Amos Waterland. Base operating system provisioning and bringup for a commercial supercomputer. In Parallel and Distributed Processing Symposium, (IPDPS), pages 1–7. IEEE, 2007.

[26] Michael Davis, Sean Bodmer, and Aaron LeMasters. Hacking Exposed Malware and Rootkits. McGraw-Hill, Inc., 2009.

[27] Christina Delimitrou and Christos Kozyrakis. Paragon: QoS-aware scheduling for heterogeneous datacenters. ACM SIGARCH Computer Architecture News, 41(1):77–88, 2013.

[28] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and QoS-aware cluster management. ACM SIGPLAN Notices, 49(4):127–144, 2014.

[29] Zakir Durumeric, Frank Li, James Kasten, Johanna Amann, Jethro Beekman, Mathias Payer, Nicolas Weaver, David Adrian, Vern Paxson, Michael Bailey, et al. The matter of heartbleed. In Proceedings of the 2014 conference on internet measurement conference, pages 475–488. ACM, 2014.

[30] Aaron Emigh. The crimeware landscape: Malware, phishing, identity theft and beyond. Journal of Digital Forensic Practice, 1(3):245–260, 2006.

[31] Foreman. Foreman provisioning and configuration system.

[32] Jeffrey Scott Foster and Alexander S Aiken. Type qualifiers: lightweight specifications to improve software quality. PhD thesis, Citeseer, 2002.

[33] T Fox-Brewster. What is the shellshock bug? is it worse than heartbleed. The Guardian, 2014.

[34] Eitan Frachtenberg, Dror G Feitelson, Fabrizio Petrini, and Juan Fernandez. Adaptive parallel job scheduling with flexible CoScheduling. Parallel and Distributed Systems, IEEE Trans. on, 16(11):1066–1077, 2005.

[35] Baris Guler, Munira Hussain, Tau Leng, and Victor Mashayekhi. The advantages of diskless hpc clusters using nas. Technical Report Dell Power Solutions, 2002.

[36] C.K. Haun, C.H. Prouse, J. Sokol, and P.M. Resch. Providing a reliable operating system for clients of a net-booted environment, June 15 2004. US Patent 6,751,658.

[37] Scott Hogg. Clos networks: What's old is new again, Jan 2014.

[38] IBM. IBM bigFix: A collaborative endpoint management and security platform. .

[39] Nwokedi Idika and Aditya P Mathur. A survey of malware detection techniques. Purdue University, 48, 2007.

[40] Red Hat Inc. GHOST: glibc vulnerability (CVE-2015-0235). `https://access.redhat.com/articles/1332213`.

[41] Lina Jia, Min Zhu, and Bibo Tu. T-vmi: Trusted virtual machine introspection in cloud environments. In Cluster, Cloud and Grid Computing (CCGRID), 2017 17th IEEE/ACM International Symposium on, pages 478–487. IEEE, 2017.

[42] Yunlian Jiang and Xipeng Shen. Exploration of the influence of program inputs on CMP co-scheduling. In Euro-Par 2008 — Parallel Processing, pages 263–273. Springer, 2008.

[43] Willy Jimenez, Amel Mammar, and Ana Cavalli. Software vulnerabilities, prevention and detection methods: A review1. Security in Model-Driven Architecture, page 6, 2009.

[44] Stephen T Jones, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Vmm-based hidden process detection and identification using lycosid. In Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pages 91–100. ACM, 2008.

[45] Melanie Kambadur, Tipp Moseley, Rick Hank, and Martha A Kim. Measuring interference between live datacenter applications. In Proc. of the Int. Conf. on High Performance Computing, Networking, Storage and Analysis (SC'12), page 51. IEEE Computer Society Press, 2012.

[46] Y. Klimenko. Technique for reliable network booting of an operating system to a client computer, October 26 1999. US Patent 5,974,547.

[47] Ana Klimovic, Christos Kozyrakis, Eno Thereska, Binu John, and Sanjeev Kumar. Flash storage disaggregation. In Proceedings of the Eleventh European Conference on Computer Systems, page 29. ACM, 2016.

[48] Dalibor Klusáček, Hana Rudová, and Michal Jaroš. Multi resource fairness: Problems and challenges. In Job Scheduling Strategies for Parallel Processing (JSSP), pages 81–95. Springer, 2014.

[49] R.L. Lewis. Virtual disk image system with local cache disk for iscsi communications, August 2 2005. US Patent 6,925,533.

[50] Feng Liu and Jon B Weissman. Elastic job bundling: An adaptive resource request strategy for large-scale parallel applications. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'15). ACM, 2015.

[51] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In Proc. of the 42nd Annual Int. Symp. on Computer Architecture, pages 450–462. ACM, 2015.

[52] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In Proc. of the 44th annual IEEE/ACM Int. Symp. on Microarchitecture, pages 248–259. ACM, 2011.

[53] Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. Contention aware execution: Online contention detection and response. In Proc. of the 8th annual IEEE/ACM Int. Symp. on Code Generation and Optimization, pages 257–265. ACM, 2010.

[54] T Mccabe. More complex= less secure. McCabe Software, Inc, page 12, 2014.

[55] Ron Minnich, James Hendricks, and Dale Webster. The linux bios. In Proceedings of the 4th Annual Linux Showcase and Conference. USENIX, 2000.

[56] Amin Mosayyebzadeh, Gerardo Ravago, Apoorve Mohan, Ali Raza, Sahil Tikale, Nabil Schear, Trammell Hudson, Jason Hennessey, Naved Ansari, Kyle Hogan, Charles Munson, Larry Rudolph, Gene Cooperman, Peter Desnoyers, and Orran Krieger. A secure cloud with minimal provider trust. In 10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18), Boston, MA, 2018. USENIX Association.

[57] Shripad Nadgowda, Sastry Duri, Canturk Isci, and Vijay Mann. Columbus: Filesystem tree introspection for software discovery. In Cloud Engineering (IC2E), 2017 IEEE International Conference on, pages 67–74. IEEE, 2017.

[58] Shripad Nadgowda and Canturk Isci. Drishti: Disaggregated and interoperable security analytics framework for cloud. In Proceedings of the ACM Symposium on Cloud Computing, SoCC '18, pages 528–528, New York, NY, USA, 2018. ACM.

[59] Shripad Nadgowda, Canturk Isci, and Mustafa Bal. DÉjÀvu: Bringing black-box security analytics to cloud. In Proceedings of the 19th International Middleware Conference Industry, Middleware '18, pages 17–24, New York, NY, USA, 2018. ACM.

[60] Florian Negele, Felix Friedrich, and Bernhard Egger. On the design and implementation of an efficient lock-free scheduler. 2015.

[61] Michael Nelson, Beng-Hong Lim, Greg Hutchins, et al. Fast transparent migration for virtual machines. In USENIX Annual technical conference, general track, pages 391–394, USA, 2005.

[62] Shuangcheng Niu, Jidong Zhai, Xiaosong Ma, Mingliang Liu, Yan Zhai, Wenguang Chen, and Weimin Zheng. Employing checkpoint to improve job scheduling in large-scale systems. In Job Scheduling Strategies for Parallel Processing (JSSPP), pages 36–55. Springer, 2013.

[63] Fábio Oliveira, Sahil Suneja, Shripad Nadgowda, Priya Nagpurkar, and Canturk Isci. Opvis: extensible, cross-platform operational visibility and analytics for cloud. In Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track, pages 43–49. ACM, 2017.

[64] Yushi Omote, Takahiro Shinagawa, and Kazuhiko Kato. Improving agility and elasticity in bare-metal clouds. In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systemsi, ASPLOS, pages 145–159, Turkey, 2015. ACM.

[65] OpenCrowbar. The Crowbar project, 2015.

[66] Openstack. Ironic.

[67] Suraj Prabhakaran, Marcel Neumann, Sebastian Rinke, Felix Wolf, Abhishek Gupta, and Laxmikant V Kale. A batch system with efficient adaptive scheduling for malleable and evolving applications. In Parallel and Distributed Processing Symposium (IPDPS'15), pages 429–438. IEEE, 2015.

[68] Puppetlabs. Provisioning with Razor.

[69] K Salah, Raed Al-Shaikh, and Mohamad Sindi. Towards green computing using diskless high performance clusters. In Network and Service Management (CNSM), 2011 7th International Conference on, pages 1–4. IEEE, 2011.

[70] CERN Computer Security. Rough Auditing Tool for Security. `https://security.web.cern.ch/security/recommendations/en/codetools/rats.shtml`.

[71] Dongyou Seo, Myungsun Kim, Hyeonsang Eom, and Heon Y Yeom. Bubble task: A dynamic execution throttling method for multi-core resource management. In Job Scheduling Strategies for Parallel Processing (JSSP), pages 1–16. Springer, 2014.

[72] Ohad Shai, Edi Shmueli, and Dror G Feitelson. Heuristics for resource matching in Intels compute farm. In Job Scheduling Strategies for Parallel Processing (JSSP), pages 116–135. Springer, 2014.

[73] Yonghee Shin and Laurie Williams. Is complexity really the enemy of software security? In Proceedings of the 4th ACM workshop on Quality of protection, pages 47–50. ACM, 2008.

[74] D. Sposato. Method and apparatus for remotely booting a client computer from a network by emulating remote boot chips, October 8 2002. US Patent 6,463,530.

[75] Peter Szor. The art of computer virus research and defense. Pearson Education, 2005.

[76] David Talby and Dror G Feitelson. Improving and stabilizing parallel computer performance using adaptive backfilling. In Parallel and Distributed Processing Symposium, 2005. Proc. 19th IEEE Int., pages 84a–84a. IEEE, 2005.

[77] Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. The impact of memory subsystem resource sharing on datacenter applications. In Computer Architecture (ISCA), 2011 38th Annual Int. Symp. on, pages 283–294. IEEE, 2011.

[78] Lingjia Tang, Jason Mars, Wei Wang, Tanima Dey, and Mary Lou Soffa. Reqos: Reactive static/dynamic compilation for QoS in warehouse scale computers. In ACM SIGPLAN Notices, volume 48, pages 89–100. ACM, 2013.

[79] Lingjia Tang, Jason Mars, Xiao Zhang, Robert Hagmann, Robert Hundt, and Eric Tune. Optimizing Google's warehouse scale computers: The NUMA experience. In High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th Int. Symp. on, pages 188–197. IEEE, 2013.

[80] Tanium. Platform for endpoint management and security.

[81] Devananda Van der Veen et al. Openstack Ironic Wiki.

[82] John Viega, Jon-Thomas Bloch, Yoshi Kohno, and Gary McGraw. Its4: A static vulnerability scanner for c and c++ code. In Computer Security Applications, 2000. ACSAC'00. 16th Annual Conference, pages 257–267. IEEE, 2000.

[83] David A Wheeler. Flawfinder, 2011.

[84] Dan Williams, Ricardo Koller, and Brandon Lum. Say goodbye to virtualization for a safer cloud. In 10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18), Boston, MA, 2018. USENIX Association.

[85] WMware.com. VMware Workstation 5.0 Understanding Clones.

[86] Hyun wook Baek, Abhinav Srivastava, and Jacobus Van der Merwe. Cloudvmi: Virtual machine introspection as a cloud service. In Cloud Engineering (IC2E), 2014 IEEE International Conference on, pages 153–158. IEEE, 2014.

[87] Wei Yan and Nirwan Ansari. Why anti-virus products slow down your machine? In Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on, pages 1–6. IEEE, 2009.

[88] Hailong Yang, Alex Breslow, Jason Mars, and Lingjia Tang. Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers. In ACM SIGARCH Computer Architecture News, volume 41, pages 607–618. ACM, 2013.

[89] Fangzhou Yao and Roy H Campbell. Cryptvmi: Encrypted virtual machine introspection in the cloud. In Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on, pages 977–978. IEEE, 2014.

[90] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. CPI^2: CPU performance isolation for shared compute clusters. In Proc. of the 8th ACM European Conf. on Computer Systems (EuroSys'13), pages 379–391. ACM, 2013.

[91] Yunqi Zhang, Michael A Laurenzano, Jason Mars, and Lingjia Tang. Smite: Precise QOS prediction on real-system SMT processors to improve utilization in warehouse scale computers. In Proc. of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pages 406–418. IEEE Computer Society, 2014.