The Design and Operation of CloudLab

Dmitry Duplyakin Robert Ricci Aleksander Maricq Gary Wong Jonathon Duerig
Eric Eide Leigh Stoller Mike Hibler David Johnson Kirk Webb
Aditya Akella* Kuangching Wang† Glenn Ricart‡ Larry Landweber* Chip Elliott§
Michael Zink¶ Emmanuel Cecchet¶ Snigdhaswin Kar† Prabodh Mishra†

University of Utah *University of Wisconsin †Clemson University
‡US Ignite §Raytheon ¶UMass Amherst

Given the highly empirical nature of research in cloud computing, networked systems, and related fields, testbeds play an important role in the research ecosystem. In this paper, we cover one such facility, CloudLab, which supports systems research by providing raw access to programmable hardware, enabling research at large scales, and creating a shared platform for repeatable research.

We present our experiences designing CloudLab and operating it for four years, serving nearly 4,000 users who have run over 79,000 experiments on 2,250 servers, switches, and other pieces of datacenter equipment. From this experience, we draw lessons organized around two themes. The first set comes from analysis of data regarding the use of CloudLab: how users interact with it, what they use it for, and the implications for facility design and operation. Our second set of lessons comes from looking at the ways that algorithms used "under the hood," such as resource allocation, have important—and sometimes unexpected—effects on user experience and behavior. These lessons can be of value to the designers and operators of IaaS facilities in general, systems testbeds in particular, and users who have a stake in understanding how these systems are built.

1 Introduction

CloudLab [31] is a testbed for research and education in cloud computing. It provides more control, visibility, and performance isolation than a typical cloud environment, enabling it to support work on cloud architectures, distributed systems, and applications. Initially deployed in 2014, CloudLab is now heavily used by the research community, supporting nearly 4,000 users who have worked on 750 projects and run over 79,000 experiments.

On the surface, CloudLab acts like a provider of cloud computing resources: users request on-demand resources, configure them with software stacks of their choice, and perform experiments. Much like a cloud, the testbed simplifies many of the procedures surrounding access to resources, including selection of hardware configuration, creation of custom images, automation for software installation and configuration, and more. CloudLab staff take care of the construction, maintenance, operation, etc. of the facility, letting users focus on their research. CloudLab gives the benefits of economies of scale and provides a common environment for repeatability.

CloudLab differs significantly from a cloud, however, in that its goal is not only to allow users to build applications, but entire clouds, from the "bare metal" up. To do so, it must give users unmediated "raw" access to hardware. It places great importance on the ability to run fully observable and repeatable experiments. As a result, users are provided with the means not only to *use* but also to *see*, *instrument*, *monitor*, and *modify* all levels of investigated cloud stacks and applications, including virtualization, networking, storage, and management abstractions. Because of this focus on low-level access, CloudLab has been able to support a range of research that cannot be conducted on traditional clouds.

As we have operated CloudLab, we have found that, to a greater extent than expected, "behind the scenes" algorithms have had a profound impact on how the facility is used and what it can be used for. CloudLab runs a number of unique, custom-built services that support this vision and keep the testbed operational. This includes a resource mapper, constraint system, scheduler, and provisioner, among others. CloudLab has had to make several trade-offs between general-purpose algorithms that continue to work well as the system evolves, and more tailored ones that provide a smoother user experience. The right choices for many of these trade-offs were not apparent during the design of the facility, and required experience from the operation of the facility to resolve.

The primary goal of this paper is to provide the architects of large, complex facilities (not only testbeds, but other IaaS-type facilities as well) with lessons from CloudLab's design choices and operational experiences. CloudLab is one of many facilities that serve the research community in various capacities [8, 6, 16, 33, 21, 34, 31, 35], and we aim to generalize the lessons from this specific facility. As a secondary goal, we hope that users of these facilities benefit from a closer look into the way they are designed and operated. With these goals in mind, this paper makes two contributions:

- In Section 2, we describe the CloudLab facility as it has been built and analyze its basic usage patterns and the research conducted on it. This analysis, and the dataset that goes with it, represent a contribution to the **community understanding of the practical operation of IaaS-type facilities**.

- In Section 3, we analyze specific design choices using data from the operational system, looking at some of the trade-offs inherent in the facility's design. This analysis yields **important insights about how these choices affect user behavior and point to design principles for other facilities.**

Sections 4 and 5 cover related work and conclude.

2 Development and Use of CloudLab

We begin with background on CloudLab; our goal is not a complete summary of its goals, design, and deployment, but to provide sufficient context for the analyses that follow. We then examine usage patterns: how the use of the facility has evolved over time, the availability of resources, and the types of research that are conducted on it. From these analyses, we draw lessons about user behavior and look at the implications for the design of testbeds and IaaS facilities in general.

2.1 The Deployed CloudLab Facility

The primary CloudLab hardware is hosted at three sites: the University of Utah, Clemson University, and the University of Wisconsin–Madison. Though every site supports a wide variety of hardware-agnostic experimentation, each site specializes in a different area of research. Wisconsin's hardware is designed for networking and storage work, Clemson's for analytics and high-performance workloads, and Utah's for scale-out workloads. This equipment has come online in batches as CloudLab has been built out and evolved in response to user demand. Identical nodes in the same batch are all labeled with the same *hardware type* to help users request nodes with specific properties and to enable experiments to be repeated on the same types of resources. Since its initial public availability in December 2014, CloudLab has added devices such as programmable Ethernet switches, GPUs, Infiniband, and high-disk-count servers in response to user feedback. A full description of CloudLab's hardware can be found in its manual [36].

In addition to the hardware that it owns, CloudLab is *federated* [30, 7] with several other facilities, including Emulab [39] and Apt [32]. This brings the total number of servers available to CloudLab users up to about 2,250, and for the rest of the paper we include these resources in our analysis and discussion of CloudLab's hardware.

CloudLab is operated using software developed in-house specifically for running research testbeds: its control software is directly descended from software developed for the Emulab [39], GENI [25, 32], and Apt [32] testbeds. We have extended this software to better support experimentation on clouds and have made a number of improvements (such as those documented in Section 3) based on our experience running the facility.

CloudLab provides access to its devices at the *lowest layer* possible with a *minimum* of virtualization and abstraction between users and hardware. The reason for this is twofold. First, CloudLab's goal is to support research that is not possible on public (or typical private) clouds: it allows users to modify aspects of the software stack that would be fixed on those platforms, such as the storage, virtualization, and networking layers. Second, this supports more repeatable experimentation than facilities that virtualize and share their resources, as it provides strong performance isolation between tenants, factoring out the unpredictable "background noise" that makes it harder to draw sound, scientific conclusions. CloudLab takes pains to ensure that all servers of the same hardware type have comparable performance: in prior work [22], we have developed techniques for identifying servers whose performance is not statistically representative of the whole, and we exclude such servers from the population seen by experimenters. The facility takes the principle of low-level access beyond just servers and also provides "raw" access to other types of hardware such as programmable Ethernet switches [37] and servers with many drives from which users can build their own SANs.

Experiments in CloudLab are instances of *profiles*. A profile contains a description of the hardware resources (servers, switches, etc.) that the experiment will run on, and the software needed to run the experiment (in the form of disk images, `git` repositories, and scripts to run). When a profile is *instantiated*, CloudLab selects available hardware that matches the profile's specification and provisions that hardware with the software and configuration options described in the profile. Every instance of the profile runs on a separate set of hardware resources, and many instances of the same profile can run simultaneously. The CloudLab operators provide standard profiles for popular cloud software stacks, such as OpenStack [28], as well as bare-metal profiles that load standard Linux distributions. Users can also create their own profiles, which they can share with others. A typical workflow for creating a new profile involves starting with CloudLab-provided disk images, installing custom software, and creating a hardware description meeting the experiment's needs. All experiments have an *expiration*: when they are first created, they are set to expire after a few hours. Users can then request that their experiments be *extended* to last longer; short extensions (hours to days) are granted automatically (assuming resources do not need to be reclaimed to satisfy reservations), and administrators evaluate requests for longer periods (weeks to months). When deciding whether to grant these requests, administrators look at coarse-gained *idleness* statistics, such as CPU load and network packet counts, to determine whether the user is using resources efficiently; other than this, CloudLab does not collect information about use *inside* of experiments. It is typical for there to be 200–300 experiments active on CloudLab at any point in time.

It is possible to fully script the workloads that run inside of an experiment, but in practice, most research done on CloudLab involves a great deal of development time and exploratory experiments, so most use is interactive. A key difference between CloudLab and typical cloud (as well as research and academic cyberinfrastructures such as Jetstream [33], Chameleon Cloud [21], and the Mass Open Cloud [35]) is that clouds place emphasis on *elasticity*, and therefore tends to treat ensembles of VMs working together as an *orchestration* problem. CloudLab's profiles place the emphasis instead on describing a *complete, repeatable environment*. This makes it less elastic, but makes it easier to describe entire networks and to repeat experiments in a consistent environment. We have found that some users do have initial confusion regarding this different focus, but that they tend to find it an easier way to run repeated experiments in the long run.

2.2 Hardware Overview

CloudLab Utah has a large number of servers, each with relatively modest specifications. 585 of the servers use HPE's high-density Moonshot platform, which places 45 low-power servers (Intel Xeon-D or ARM64 SoC) in each chassis. Each chassis contains two 10 Gbps switches, which effectively function as "top of rack" switches and are interconnected at 160 Gbps through a core switch. Another 200 servers connect to both a traditional Ethernet network (at 25 Gbps) and to a "layer-1" network. The latter allows control of the physical-layer topology, configurably "wiring" nodes to user-controllable Ethernet switches or directly to each other. These user-controllable Ethernet switches are allocated to one user at a time, allowing users to have full control over their configuration and even, in some cases, to program them.

CloudLab Wisconsin's goal is to reflect the type of technology and architecture found in a typical modern enterprise datacenter. All servers (which come from Cisco) are dual-socket and have a mix of spinning hard drives (HDDs) and solid state drives (SSDs). Several servers have large numbers of disks (up to 14), allowing users to build their own SAN configurations. Many are equipped with GPUs, enabling work on machine learning and applications of GPU computing to other areas, including network packet processing and other systems tasks. The network is arranged in a Clos topology.

CloudLab Clemson focuses on putting more CPU cores in each server (from Dell) and on a greater amount of RAM per core. This makes it suitable for hosting big data analytics (such as Hadoop and Spark), for running high-performance computing workloads, and for hosting large numbers of virtual machines. The Ethernet experiment network topology uses three interconnected core switches, each connected to a companion top-of-rack switch handling direct server connections. In addition to its Ethernet network, CloudLab Clemson has a 40 Gbps QDR Infiniband network used for HPC and RDMA experiments.

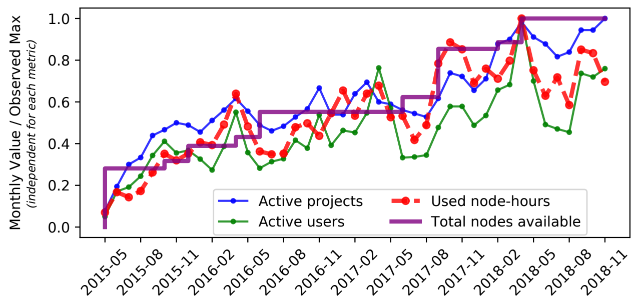

Figure 1: Growing testbed capacity and utilization. To produce a consistent scale, we divide monthly values by the all-time maximum value for each metric.

The hardware at each site has grown over time: the number of servers has increased approximately fourfold over the period covered in this paper. No hardware has yet been retired.

2.3 Usage Patterns

Figure 1 shows how CloudLab's userbase has grown along with its capacity. Starting in early 2015, when CloudLab exited its "preview" phase and became open for general use, it has grown steadily. As its capacity has increased, so too have its users: the more capacity, the more active users there are at a time, and the more projects (roughly corresponding to research groups and classes) are supported. Within the general upward trend, specific yearly cycles can be seen. CloudLab has lower usage during the summer, when there are few classes and research activity is slow. CloudLab's peak usage typically comes in the late spring: this is due to the confluence of major paper deadlines (OSDI, SOSP, SOCC, NSDI spring deadline, etc.) and end-of-year coursework.

CloudLab needs to gracefully handle periods of both high and low utilization. While we expected variation in usage over time, the extent to which it drives user behavior was somewhat surprising. Because most of CloudLab's usage is interactive, periods of low utilization are simply times when users are able to start experiments at will, without having to wait for resources to become available. We have therefore focused on improving user experience during periods of heavy utilization: as detailed in Section 3.3, we have built an *optional* reservation system which allows users to schedule resources ahead of time. Another strategy would be to incentivize users to shift their work from periods of high demand to periods of lower demand. In commercial clouds, one way of doing this is through spot pricing [1], which offers economic incentives. Because CloudLab's users do not pay to use it, economic incentives are not available; while various "virtual currency" approaches have been proposed for use in related facilities such as PlanetLab [18], none have seen widespread

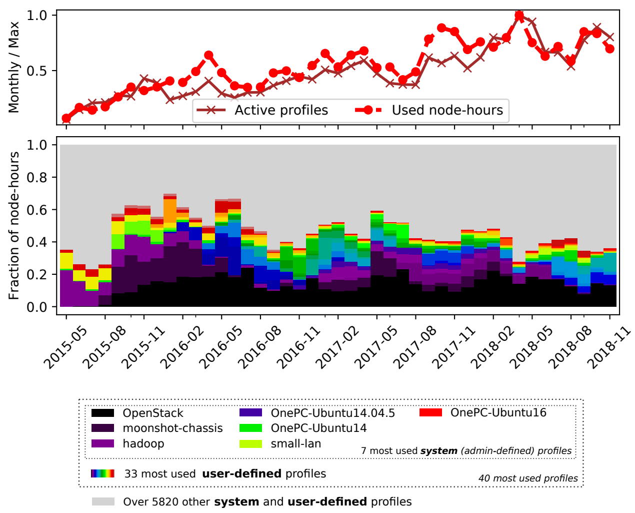

Figure 2: Evolving distribution of profile usage.

success. Demand in CloudLab's core communities appears hard to shift: Figure 1 suggests that CloudLab's users are heavily deadline-driven; it is not feasible to do coursework outside of the school-year, and not many research projects can run their experiments months before conference deadlines. If CloudLab is to "fill in" lightly-loaded periods, it will need to adopt backfill strategies [11, 23] that enable use by work that is easier to shift in time, such as the queueing systems used by high performance and high throughput computing.

Like the number of active users, the number of profiles in use at any time has grown as shown in Figure 2. The bottom half of the figure shows the forty most used profiles, as measured by the cumulative node-hours. We can see that usage patterns vary significantly by month. In some months, such as in early summer 2016, the top forty profiles constitute nearly 70% of the testbed's utilization, while in other months, such as recently, this fraction is less than 40%. The remaining node-hours are allocated by experiments representing over 5,820 other profiles. Users run experiments using up to 572 unique profiles monthly; the median is 233.

The long-tailed distribution we observe indicates that there is a large number of profiles with relatively low and infrequent use, but their combined utilization constitutes most of the user activity. This holds important implications for analysis of usage patterns and for design choices. In Section 2.5, we reflect on the fact that the testbed's major practical value is attributed to facilitating not merely a handful of common use cases, but rather a large variety of experiments in this "long tail." In quantitative analysis, it means we should use medians rather than means as the preferred measure of central tendency due to the highly skewed distributions. The analysis of other percentiles (e.g., 75th or 95th percentiles) provides complementary insights, as we discuss in Section 3.3. Furthermore, this diverse and evolving utilization distribution suggests that we cannot draw reliable conclusions about the impact of testbed's capabilities on usage patterns based solely on comparison of usage statistics from different periods of time. For example, if we compare statistics for 2017 and 2018, it would be difficult to determine the extent to which evolving usage patterns were due to changes in the system or due to the natural evolution of user interests. For the same reason, month-to-month comparisons are also unlikely to provide sufficient evidence for "before and after" analyses for system capabilities.

Fundamentally, periods of time that seem similar by one statistical measure (such as the number of active users) can look very different for other measures (such as the distribution of profile use). We posit that this is likely to be true for many IaaS-type facilities: while multi-tenancy smooths out some measures into predictable shapes (e.g., the top half of Figure 2), others are quite chaotic (e.g., the bottom half of the same figure). Taking these patterns into account when designing facilities can improve their utility and user experience.

2.4 Resource Availability

Availability and diversity of resources play critical roles in the adoption and continued use of a testbed or other IaaS facility. If users' needs frequently cannot be satisfied due to insufficient availability, those users will likely move to other facilities. Users also tend to seek out hardware with cutting-edge features and the highest performance characteristics. New hardware types have been introduced to CloudLab over time in order to satisfy both capacity and the capability requirements. Not only did the new hardware attract new users, but it also reduced contention for older, already deployed resources.

In Figure 3, we show both short- and long-term availability trends for the major CloudLab hardware types. The X-axis represents a fraction of all nodes of a particular type, and the Y-value at each point shows what fraction of the time *at least* that many nodes were available. Lines on these graphs that fall steeply signify the types that are in use most of the time, while higher curves represent more available types. For example, we can compare `d430` and `m400`: the former type is more heavily used across all three graphs.

As we saw in the previous analysis, metrics that look smooth when viewed from a high level show much more variability when we look at the details. This is important because it is often the details that influence users' experience: e.g., for an individual user, availability of the specific node type(s) needed for their experiment is important, rather than the availability of the testbed as whole. To illustrate this, we include the two monthly plots showing the variation that occurs between "slow" and "busy" months. For example, the curves for `pc3000` indicate that in January 2018 users found 80% of these nodes available for use 80% of the time. In contrast, in April of that year, there were *no* times at which 80% of these nodes were available. For several other hardware types (such as `c220g1` and `d710`), resources were even

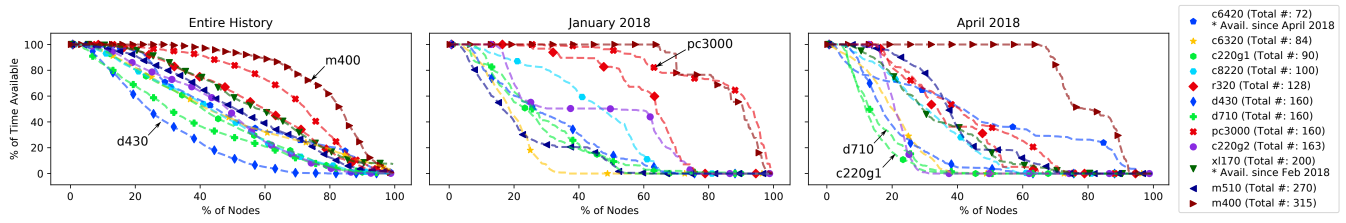

Figure 3: The availability of CloudLab resources. The left plot shows availability over the entire period of time each hardware type was available; the other two plots show availability during low-utilization (center) and high-utilization (right) months.

scarcer, and at no time are more than 30–40% of these nodes free. We also note that d430's low availability is reflected in two ways: it can be seen in these graphs, and we will discuss it in Section 3.3, where we find that users commonly make reservations to schedule access to this in-demand hardware. The curves in these figures highlight that the extent to which system changes and evolving utilization patterns impact individual hardware types varies significantly across the types.

2.5 Research Use of CloudLab

To understand the research conducted on CloudLab, we surveyed 93 papers published in 2017–2018 that used CloudLab for part or all of their experimental evaluations. Table 1 presents a categorization of the papers by the primary area of contribution, using a list of systems and related research areas. These areas are quite broad, and no one area dominates. Given that the most prevalent area, Networking, is an area where typical clouds provide very little transparency [14] and control only as an overlay [2], it is intuitive that research in this area benefits from CloudLab's greater visibility and control. Research in the second area, Security, benefits from the "closed world" of CloudLab, allowing experiments that involve attacks that would be considered hostile in a typical cloud and defenses that need to be implemented within the cloud framework itself.

In this analysis, we found two primary motivations that drove experimenters to CloudLab. The first is low-level access to hardware because of *features* that could not be developed in virtualized environments. Almost every paper in this set used some different aspect of CloudLab, such as the ability to re-configure Ethernet switches, the ability to build an HPC-like environment with root access, the SDN available on most of the CloudLab networks, the ability to monitor power use, the ability to build a complete OpenStack cloud inside the infrastructure, etc. The second motivation is the *performance predictability and isolation* that are difficult to come by in environments that use multi-tenancy on hosts and storage. When the primary metric of interest in a system is its performance, anything that adds variability requires, at a minimum, a far greater number of experiment repetitions to achieve statistical confidence [22, 17]. There is also the

Networking	30%
Security	16%
Storage	11%
Applications	10%
Computing	9%
Virtualization	8%
Databases	7%
Middleware	4%
Energy & Power	2%
Other	15%

Table 1: Research areas in 93 papers that used CloudLab.

question of the extent to which an evaluation is measuring artifacts of the platform vs. the actual system under test: while a more transparent environment does not guarantee that no system artifacts are present, it does give the experimenter more opportunities to observe, understand, and correct for these effects.

The main lesson we take from this analysis is that, as facility operators, we are constantly surprised by the uses to which users put the facility. Had we started from a position of virtualizing everything, then providing lower-level access to specific systems as needed, we think it is unlikely that we would have been able to anticipate all of the use cases found in this survey. Starting from a position of maximizing user control helps to maximize use of the facility.

3 High-Level Effects of Low-Level Decisions

We now move from examining how people use CloudLab to looking at the interactions between design decisions, operational experience, and user behavior. We have found that the *choice of algorithms* deep within the implementation of a system like CloudLab has a profound effect on the ways that users interact with the system, and even what they are able to accomplish. As a result, we have made many changes to the CloudLab facility during its lifetime; in this section, we present individual subsystems that we had to evolve based on facility's usage patterns. The high-level theme of this section is that the choices made at these low levels are not, contrary to what one might expect, simply implementation details, nor are they neutral with respect to the utility of the facility. When building an IaaS facility, designers cannot consider aspects such as resource mapping separately from user goals, requirements, and workflows. These aspects of the system must be co-designed, so that users can work *with* these subsystems rather than having to fight *against* them to get work done.

3.1 Resource Mapping

There exist several approaches to the problem of mapping user requests to physical resources. For instance, commercial clouds do not provide control over this mapping within selected instance classes; they manage the placement and consolidation for effective utilization and hide the details from the users. In contrast, Chameleon [21], which is designed as a testbed for repeatable experiments (similar to CloudLab but serving a different research community), has its users do the mapping by asking them to specify IDs of the particular servers they want to use in their requests.

CloudLab takes a unique approach where it recognizes two aspects in this mapping. It is a constraint-satisfaction problem in the sense that the user's request is a specification that must be satisfied; specifically, it resembles the subgraph isomorphism problem [10] in that both the requested and physical topologies are graphs consisting of servers, switches, etc. It is also an optimization problem, because the mapping must be done in a way that maximizes the possibilities for future mappings: it should avoid using scarce resources unless they are specifically requested or there is no available alternative. CloudLab exposes the outcomes of the mapping to the users and allows them to reuse hardware IDs if necessary.

CloudLab's mapping algorithm is derived from the one developed for Emulab [29], and uses simulated annealing to address this NP-hard problem. The advantage of using a powerful, general-purpose algorithm is that it enables the expression of complex constraints and preferences. The disadvantage, however, is that when a mapping *cannot* be found for a request, it can be difficult for users—and even administrators—to understand why. In CloudLab, we have had to evolve this system to improve the intelligibility of the responses that it provides.

The fundamental trade-off exposed here is between a *general* algorithm that makes few assumptions about the facility (and therefore is easily adaptable to new resources) and a more *specialized* algorithm that understands facility semantics and can provide actionable suggestions when a mapping fails. The general algorithm fundamentally lacks semantic information about what the user *may be trying to accomplish* and the *classes of requests that "make sense" on this particular testbed*. A mapping algorithm more tailored to a specific use case could embed such information and make assumptions about user goals.

Our response to this trade-off has been to retain the general algorithm, but to develop a set of heuristics that turn some of the more common failure modes into messages that are easier for users to understand. A major challenge in designing these heuristics is that they must be *conservative*: that is, every mapping that would have succeeded without the heuristic must still succeed. Our experience has been that it is preferable to build such heuristics *around* the mapping algorithm rather than *into* it. Building conservative checks into the *randomized* setting of the mapper itself is extremely difficult and can easily cause unexpected changes in behavior. It is easier—and more informative for the user—to build conservative checks as a *deterministic* wrapper around the mapper. We now describe some of these checks, which we have added over time in response to common error patterns and common questions from users.

In an ideal situation, all mapping errors would be explained to the user by concise, actionable error messages. In theory, the universe of possible mapping errors is so vast that not all have simple explanations. We have found that in practice, however, it is possible to catch most mapping errors with heuristics. We now describe the set of heuristics we have developed over time in response to use patterns and frequent user questions.

Looking in Table 2 at the last year (L.Y.) of mapping errors, approximately 84% of all errors are explained by the top 10 error messages, and of that top 10, only 13.5% are ones that we classify as "unhelpful." If we look at this as a percentage of all experiments, only 1.2% of all attempts to start experiments in the last year have received these four unhelpful mapper messages.

The top two messages (lines 1 and 2 in the table) together account for about half of all mapper errors, and they simply indicate a lack of free nodes (servers or user-controlled switches) at the current time. The first message indicates that there are insufficient nodes free *right now* while the second says that this would occur in the *near future* due to the reservation system described later in this section. There is a third variation on this message (line 7); this is an older version of line 1, which we updated partway through the year to clarify its meaning and provide more specific information. Note that this class of messages are *per-type*, so experiments that request, for example, both servers and user-controlled switches get specific feedback on which is the limitation. The number of available nodes is reported in order to allow the user to decide whether they would prefer to request fewer nodes or to wait until enough nodes become available. When users request specific nodes, in contrast with asking for any nodes of a selected type, they receive explicit messages indicating that those nodes are unavailable (line 10).

Other frequent errors (lines 3, 6, and 8) indicate that there is some node in the request that cannot map to anything available. Our heuristics try to report the specific reason, such as requesting too many physical interfaces, an OS image that is incompatible with the hardware type, or a specific feature (such as a GPU add-on). The distinction between lines 6 and 8 presents an interesting illustration of our use of heuristics: underneath, the mapper uses the same mechanism to handle both of these constraints (support for a particular image is considered a "'feature"). We found that the raw

Error Message	Helpful (actionable)	% of Mapping Errors L.Y.	% of Mapping Errors ALL	% of All Errors L.Y.	% of All Experiments L.Y.
1. Resource reservation violation: X nodes of type HW requested, but only Y available	✓	27.79	14.33	16.07	2.41
2. X nodes of type HW requested, but only Y available nodes of type HW found	✓	21.86	33.01	12.64	1.89
3. No Possible Mapping for X: Too many links of type Y	✓	6.64	6.96	3.84	0.58
4. No Connection	✗	5.22	2.62	3.02	0.45
5. Insufficient Bandwidth	✗	4.88	7.53	2.82	0.42
6. No Possible Mapping for X: OS 'Y' does not run on this hardware type	✓	4.74	3.50	2.74	0.41
7. Not enough nodes because of policy restrictions or existing resource reservations	✓	4.37	2.18	2.53	0.38
8. No Possible Mapping for X: No physical nodes have feature Y	✓	3.54	2.40	2.05	0.31
9. Insufficient Nodes: Unexplained	✗	3.39	2.15	1.96	0.29
10. Fixed physical node X not available.	✓	2.56	3.15	1.48	0.22

Table 2: Distribution of recorded mapping errors. "ALL" denotes the distribution of all errors recorded since October 20, 2015. "L.Y." columns refer to the percentages reported for the last year (starting on August 1, 2017).

message, however, was unhelpful and confusing to users, so we recognize the specific case of image-related mapping failures and transform the message into something that the user can act on: she needs to either pick a different image or a different hardware type.

Lines 4 and 5 are the error messages that are the least helpful to users, and they have a similar cause: the mapper is unable to find a solution that satisfies all links and LANs with the bandwidths specified in the requests. These error messages are produced directly by the simulated annealing portion of the mapper, and it is no coincidence that they are the hardest to explain. They are highly dependent on the details of the topology requested by the user and the switch topology at each CloudLab site. There are many potential actions to take in response to such failures: change the topology, reduce the bandwidth requested, try a different CloudLab site, wait for a different set of physical resources to be free, etc. In essence, the more degrees of freedom the user has with respect to reacting to a failure, the harder it is for the facility to guess which one best addresses the user's actual goals, and the more difficult it is to provide a useful message.

3.2 Interactive Topology Design Feedback

Giving users actionable messages when their profiles don't map is helpful, but it comes fairly late in the process of experiment design. Our experience has been that users can find even the "helpful" mapper errors frustrating, as they come after the user has already invested significant time. A useful analogy is to compile-time errors and syntax checking in IDEs: compiling is complex and slow, and feedback from the editor as the user writes code, while not perfect, leads to a workflow with fewer surprise errors. What we discovered was that we needed the equivalent of realtime syntax checking for network topology design, and our answer to this is CloudLab's *topology constraint system*. The biggest challenge in building it has been to design a system with a simplified model of the mapping process that does not produce a *specific mapping*, but instead checks whether such a solution *should exist*; it must do so quickly enough to run interactively in the browser.

The constraint system is used in two contexts, and has slightly different goals in each. In the first context, it is invoked as part of Jacks: CloudLab's GUI that gives users a "drag and drop" interface for constructing profiles. In this setting, its goal is to assist novice users by disabling UI options that conflict with their existing choices and to warn them when the topology they have drawn is unlikely to be instantiatable. It does not need to admit *every possible* request that can be instantiated on CloudLab (there are more sophisticated interfaces for that), but to provide an assurance that, if a topology passes at this stage, it is virtually guaranteed to succeed in mapping (assuming there are enough resources free). In the second setting, it is used at the final stage of profile instantiation, when the user selects which CloudLab cluster to run their experiment on. Here, it checks the request against each cluster and disables selection of any cluster where the request cannot be instantiated. The goal in this case is the inverse, and we must be more conservative: We want the constraint system to block instantiation if the request will *definitely* fail, but do not want to over-zealously block instantiation that *might* succeed.

The described two-phase experiment design is unique to CloudLab. On the surface, the first phase can be compared to how responsive web interfaces for clouds—e.g., Amazon EC2's dashboard and the OpenStack's Horizon dashboard [27], hide or disable infeasible configurations. At the same time, EC2 goes as far as "attaching" storage characteristics to instance classes (even though networking is actually what is being customized) when listing the storage optimized solutions among the feasible configurations. CloudLab's constraint system makes the design process more explicit by offering interactive control over all components of interconnected experiment environments. In the second phase, requests act analogously to HTCondor's classads [9]. In practice, systems like HTCondor without interactive design capabilities make working with complex configurations laborious and error-prone.

Generating and checking candidates To check constraints, we generate a set of *candidates* which are tested against a number of *groups*. A candidate is a set of node or link resource properties which we check for mutual compatibility. A group is a whitelist of acceptable combinations relating two or more resource properties. For example, a group might include all allowed combinations of hardware type and disk image. Our constraint system also supports wildcards in both candidates (for unspecified resource properties) and groups (for cases where one resource property is universally allowed). A candidate passes if it matches all groups. Our approach uses a Boolean expression in the *product of sums* form: a set of terms containing conditions that are OR-ed together, with all terms being AND-ed together.

This process is defined in terms of sets and Boolean operations as follows: for a set of candidates $X = \{x_1, x_2, \ldots, x_k\}$, we define an evaluation procedure $f(X)$ that checks all individual candidates. We define $g(x)$ for a given configuration candidate x such that the candidate must match against all groups (A, B, etc.): $g(x) = A(x) \wedge B(x) \wedge \ldots$ For each group, the candidate must match at least one condition. As an example, suppose the following table described the conditions allowed for each group:

Group relating `site`, `hardware`, and `type`:	Group relating `hardware` and `image`:
$a_1(x) =$\{`utah`, `m510`, `xen`\}$\subseteq x$	$b_1(x) =$\{`m400`, `ubuntu16-64-ARM`\}$\subseteq x$
$a_2(x) =$\{`utah`, `m400`, `pc`\}$\subseteq x$	$b_2(x) =$\{`m510`, `ubuntu16-64-STD`\}$\subseteq x$
...	...
$a_n(x) =$\{`wisconsin`, `c220g2`, `pc`\}$\subseteq x$	$b_m(x) =$\{`c220g2`, `fbsd110-64-STD`\}$\subseteq x$
$A(x) = a_1(x) \vee a_2(x) \vee \ldots \vee a_n(x)$	$B(x) = b_1(x) \vee b_2(x) \vee \ldots \vee b_m(x)$

In this case, a candidate x=\{`utah`, `m400`, `pc`, `ubuntu16-64-ARM`\} evaluates to true, as $a_2(x) \wedge b_1(x) = 1$.

In the Jacks GUI, the candidates that we generate represent the UI element (node, link, etc.) that the user has selected and the actions they *may* take on it: OS images they may select, other nodes they may connect it to, etc. Each candidate represents a different possible action, and we disable ("gray out") UI elements for candidates that do not pass ($g(x)$ evaluates to false). In the profile instantiation process, the candidates represent *all* nodes as they appear in the request, and the request may only be submitted to clusters for which *all* candidates pass ($f(X)$ evaluates to true).

Checking Constraints Quickly The set of candidates can, in practice, be quite large: in Jacks, it grows with the number of options the user can set on the node (including other nodes to connect to), and in the instantiation process, it grows with the size of the request. We have run containerized experiments with as many as 5,000 nodes. At least one candidate must be evaluated per node in a topology, and if there are LANs, the number of candidates is quadratic in the number of nodes in each LAN. The number of conditions in each group can grow even larger, as it depends in part on the product of the number of hardware types, images, sites, and other node properties. On our current system, every candidate is evaluated against at least 10,000 conditions across all groups. However, the number of groups remains small in all cases (the current number of groups in our testbed is just 7), and in practice, there are several optimizations that allow us to take advantage of the facility environment to make checks fast.

Large requests have many nodes and thus require many candidates to be tested, but many of these candidates will likely be identical. Similarly, when Jacks evaluates which items in a drop-down box are valid, there is no need to re-evaluate combinations that have already been tested on a previous drop-down box instance. Memoizing test results and culling identical candidates yields large speed improvements for our use cases. Even with memoization, every unique candidate has to be checked once, so we have optimized the evaluation of the Boolean expression as well. Naïvely testing each condition in turn using set arithmetic yields a speed that is linear on the number of conditions. Instead, we can uniquely encode conditions as entries in hash tables, and each group can be tested with an (amortized) constant-time lookup. This lookup means that testing a candidate for the first time is linear in the number of *groups* rather than the much larger number of *conditions across all groups*. Together, these optimizations reduce the complexity of the checks from $O(c \cdot g \cdot s)$ (where c is the number of candidates, g is the number of groups, and s is the size of each group) to $O(\text{unique}(c) \cdot g)$.

Impact on User Workflow CloudLab's topology constraint system is built around the idea of using a *quantitative* advantage (fast constraint checking) to provide a *qualitative* improvement in user experience. It has done so by dramatically reducing the number of submitted requests that could not possibly map—even if all resources on the testbed were available. In many situations, builders of IaaS-type facilities face a choice: to ensure that *any* request that a user makes for *any* set of resources configured in *any* way can be instantiated on the facility, or to constrain user requests in some way. While the former is attractive, it can be expensive to guarantee and can result in situations where users *can* request certain combinations but would be better off *not* doing so because these combinations do not perform well together. CloudLab's topology constraint system shows one possible path forward on the latter alternative: constrain users' requests, and give them early, interactive feedback while they design their configurations.

3.3 Reserving Resources

Until recently, resource allocation in CloudLab was done in a First-Come-First-Served (FCFS) manner. While FCFS works well for the interactive "code, compile, debug, gather results" workflow used in the systems research community, it has a number of shortcomings: it favors small experiments (whatever fits into the available resources at the time the user is active), it can be difficult to plan for deadlines (such as the paper and class deadlines seen in Section 2.3), and it can be problematic for events that must occur at a specific time (such as tutorials and demonstrations). In response to these competing needs, we have developed a *reservation system* to support these use cases while continuing to support the dominant FCFS model.

A reservation is not an experiment scheduled to run at a specific time, but a guarantee of *available resources* at that time. This allows users to run many experiments either in series (e.g., to test different scenarios) or in parallel (e.g., one experiment per student in a class). This *loose experiment-reservation coupling* is one of the key design attributes of our reservation system and the subject of much of the analysis presented in this section.

What we found in designing our reservation system was that it needed to have a fundamentally different design than the resource mapping described in Section 3.1. Resource mapping answers the question, "Given a specific request and a set of available resources, which ones should we use?" The reservation system needs to answer "Given the current schedule of experiments and reservations, would a given action (creating a new experiment, extending an existing one, or creating a new reservation) violate that schedule?" Answering this question must be fast: like the constraint system, we need the reservation system to run at interactive speeds so that we can give users immediate feedback about their ability to create or extend experiments. Our other challenge is to support *late binding* of resources: the reservation system should promise *some* set of resources in the future, but should wait until the time comes to select *specific* ones.

Our approach diverges from the scheduling schemes offered by other facilities. On Chameleon [21], users request specific servers (using server IDs) as mentioned previously; therefore, their requests require only the *early binding*, and the system trades flexibility for simplicity (presumably at the expense of utilization). In contrast, clouds do not offer control over future scheduling decisions. They provide an illusion of infinite resources, and handle all user requests interactively, at the time of submission. In High Performance Computing, solutions are built upon *job queues* where job and user priorities impact scheduling, yet making sure that exact deadlines are met in the future is a constant challenge.

We describe our design using the following terms and operations: A request for reservation r asks for N_r nodes of the specified hardware type h_r to be available within the time window $[s_r, e_r]$. Once `submit`-ed, a request typically requires approval from CloudLab staff, though small requests are auto-approved. In addition to the `approve` operation, staff can `delete` reservations, both pending and active. At any time, users can change their experimentation plans and `delete` their reservations or submit modified requests.

Late Binding Considering that CloudLab's hardware is homogeneous within each hardware type h, the reservation system does not need to decide which *specific* nodes will be counted as N_r nodes of type $h_r \in \{h\}$: any N_r such nodes will satisfy the needs of reservation r with these parameters. This increases efficiency of resource use and helps accommodate FCFS users: it does not require us to force experiments out just because the specific nodes they have allocated happen to be reserved. As long as there are *enough* free nodes for everyone who has requested them, all experiments can continue. Therefore, we spare the reservation system the task of finding exact mappings between reservations and specific nodes and implement reservation operations as node counting tasks. The "binding" occurs later, when the user instantiates their experiment(s) near or within the $[s_r, e_r]$ window. The reservation system simply ensures that the capacity is sufficient.

Checking Reservations Quickly Given the data about active experiments—node counts and their current expiration times—and parameters of approved upcoming reservations, our reservation system constructs a tentative schedule describing how the number of available nodes is expected to change over time. This schedule can be constructed in $O(n \log n)$ time (it must sort upcoming events by time), and takes $O(n)$ time to check. Here, n is the number of events, which is a sum of the number of current experiments (typically hundreds) and the number of future reservations (typically tens). Effectively, this creates a two-phase process, in which the reservation phase involves tasks that are lightweight and fast, while the laborious resource mapping phase runs as part of lengthy resource provisioning process.

This fast checking is enabled by a key design decision: reservations are *per hardware type*—we do not allow reservations for broader categories such as "any server type." While the latter would be attractive, it would also raise the time to check the schedule far above $O(n)$. In our design, we can check the schedule for each type *independently* because the sets of nodes of each type do not overlap. There is only one, binary solution at each point in the schedule: either the sum of nodes in experiments plus the reservations exceeds the total number of nodes of that type, or it does not. If we were to have overlapping sets (e.g., specific and generic types), this would create *dependencies* both between sets and across time. Each point in the schedule would have multiple potential solutions, using different numbers of nodes from each node set. Checking the solution would not only be a matter of checking the solution at each point in time, but ensuring each solution is consistent with the solutions at other time points. The combinatorial complexity that this would entail would prevent us from quickly re-calculating and checking schedules, so we accepted the tradeoff of being more rigid with respect to node types.

Enforcing Reservations The CloudLab reservation system essentially works by "accumulating" free nodes up to the

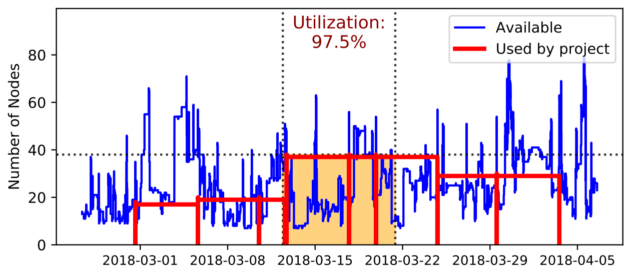

Figure 4: 38 d430 nodes reserved and used for over 9 days. The highlighted box depicts the reserved resources: the number of nodes (up to the horizontal dotted line) reserved for the reservation's time window (between the vertical dotted lines).

point at which the reservation starts. As the beginning of the reservation approaches, it scrutinizes two types of operations: creation of new experiments, and extending the duration of existing experiments. If either of these operations would overlap with the reservation and would result in there being insufficient free nodes to satisfy it, they are denied. For an illustration, refer to Figure 4, which shows a large reservation *r* requesting over 8200 node-hours worth of d430 nodes. Prior to *r*, d430's availability was insufficient for most of the preceding week (below the horizontal line). As *r*'s start approached, the admission control system began denying overlapping use, and the free nodes rose until the reservation could be satisfied. Almost immediately, the project created a large experiment (the exact size of *r*), and ran two other subsequent experiments. We can see that the final experiment outlasted the reservation: because there were no other reservations directly afterwards, the user was allowed to extend the duration of the experiment. Another interesting behavior visible in this graph is that the project was running smaller experiments before their reservation started; once it did start, they were able to double the size of their experiments.

Parameter Exploration In addition to `submit`, `approve`, and `delete`, CloudLab's reservation system supports a `validate` operation. `validate` allows users to explore potential reservations without submitting them, giving them the ability to try different times, hardware types, and reservation sizes to find configurations that fit their needs. If a validation succeeds, the user may `submit` the reservation. Taking a cue from our mapping and constraint systems, the validation procedure provides users with actionable feedback when the validation fails: messages take the form "Insufficient free nodes at Fri Sep 21 18:00:00 2018 (12 more needed)." This feedback suggests that reducing the number of nodes, shortening the reservation's duration, or moving the reservation further into the future can help the user proceed with submitting a valid reservation.

To understand how users explore different possibilities, we analyzed operations performed on our reservation system between December 6, 2017 and November 30, 2018.

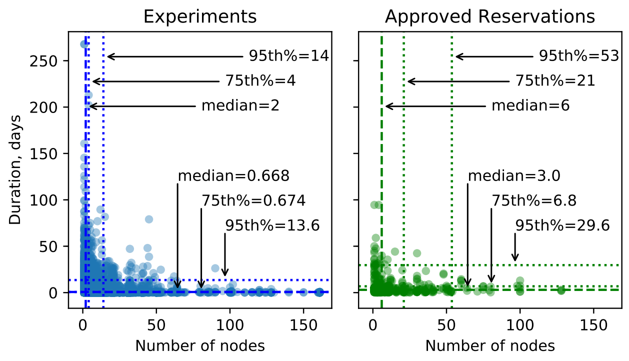

Figure 5: Comparing experiments and reservations. Empirical percentiles are labeled using '%'.

Among the 3,500 events in this dataset, there are approximately 1,800 `validate` and 900 `submit` operations. These events represent the activity of 200 users working on over 130 unique projects. Nearly 51% of reservation events are `validate` operations. On average, 2.1 validations preceded each submitted reservation (at least one is required, because users must `validate` a reservation before they `submit` it). Further analysis of the operations uncovers infrequent but revealing scenarios. For each `submit`-ed reservation, we consider `validate` operations preceding it to be indicative of a user exploring possible candidate reservations. These form a long-tailed distribution: 71% of `submit` operations were preceded by a single `validate`, and 14% by two. The remaining 15% of this distribution stretches to a maximum of 32 `validate` trials. We interpret such cases as empirical evidence for the validation procedure being sufficiently fast to allow users repeatedly *check* and *update* reservation parameters when searching for combinations that satisfy both their needs and the testbed's schedule.

Size and Duration We next compare reservations with experiments, to see whether reservations succeed in enabling larger experiments than are possible with FCFS alone. Using the same time period and reservation data as the previous analysis, we also look at records for 33,300 experiments. Figure 5 illustrates the long-tailed distributions we observe in both. Because these distributions are highly skewed, we characterize and compare them using medians (i.e., 50th percentiles), 75th, and 95th percentiles. The ratios between the pairs of the corresponding percentiles indicate that the reservations are 2.2–10.2 times larger and 3.0–5.3 times longer than experiments. We conclude that reservations do indeed enable larger experiments, though interestingly, the largest experiments were larger than the largest reservations by about 50%. Our analysis of monthly distributions also reveals that the 95th percentile for experiment durations shows significantly less volatility after we introduced the reservation system and stabilizes at its high values, around 300 hours. The same is not true of the node count statistics; the timing of the largest

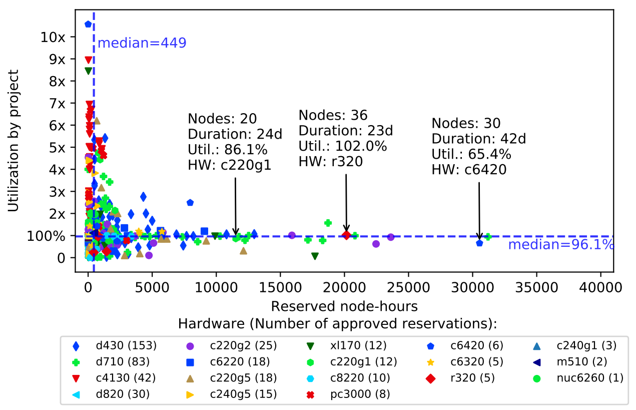

Figure 6: Utilization of reservations.

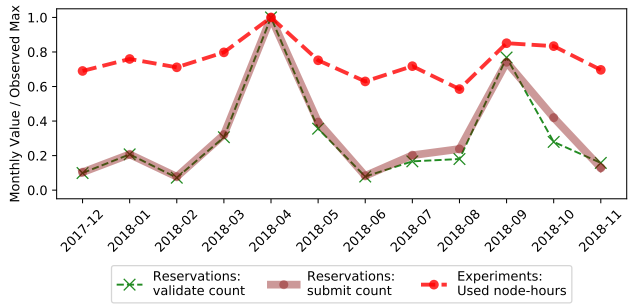

Figure 7: Use of reservations and experiments. For each metric, we divide monthly values by the all-time maximum.

spikes suggests that they are caused by testbed expansions.

Utilization CloudLab does not automatically instantiate experiments for users at the beginning of a reservation, nor does it require the end of a reservation to coincide with the end of an experiment. The current system has no direct penalties for under-using reserved nodes. This leads to an obvious utilization question: *How fully do experimenters use their reservations?* Put another way, this can be framed as a question of trust: *Can we trust users to reserve only what they need and then use what they have reserved?*

Before answering this question, we note several operational nuances that stem from the loose coupling between experiments and reservations in our design. First, we do not stop users from allocating *more* resources than they have reserved: the reservation indicates a minimum guaranteed availability, and if more are available, experimenters are free to use them. Second, if multiple experiments run on h_r hardware within $[s_r, e_r]$, we cannot distinguish experiments that are *meant* to use the studied reservation r from the ones that are run opportunistically, in addition to the planned experiments. Third, reservations are associated with *projects* (groups of users), so the user that creates the reservation may or may not be the one who actually uses it. If users in the same project coordinate their activities, one user submits a reservations on behalf of the group; otherwise, when working independently, one or multiple users submit their reservations and run planned experiments, while others run their unrelated FCFS experiments. Since the studied usage record does not allow us to distill exact user intentions, we estimate aggregate project-specific usage of hardware h_r within $[s_r, e_r]$ and view it as the *upper bound* of the intended r's utilization.

Figure 6 visualizes whole-project resource utilization for nearly 450 approved reservations. The highest point, depicting a utilization of almost 11x the quantity of resources reserved, represents a reservation where a single node was reserved for 33 hours. (The figure omits fifteen small reservations that would stretch the Y axis even further, up to 25x.) That reservation was deleted 3 hours into its time window, and, at the same time, the same or other users from the same project ran multi-node experiments up to 32 node-hours in aggregate. The result is that the project as a whole used far more resources during this time than it had reserved. Labels for several large reservations highlight instances where the utilization is near 100%. We find these and many other large experiments that conform well with the the corresponding reservations. With the median utilization for the shown instances at 96.1%, we conclude that the majority of reservations see high utilization, and we can indeed trust users to reserve what they need and use what they reserve.

In contrast, we found a fairly large number of reservations, 123 (not shown on the graph) with no identified usage. These seem to come from the cases in which the users changed their minds but did not delete their reservations, forgot about the reservations (CloudLab does send reminder emails), or, most interestingly, did run experiments but used wrong types of nodes. This final case seems to stem from misunderstandings about either how the reservation system works (the specific hardware type reserved) or the profile being used (the specific hardware type requested). CloudLab does have measures in place to encourage use of the appropriate node type: when the user has a reservation, the cluster selection box defaults to the relevant cluster, and the mapper applies a preference for nodes of the reserved type. Still, it is clear that this is an area for additional work. With the median size at 96 node-hours, however, these unused reservations add up to less than 12% of all node-hours reserved.

Reservations in Action We conclude our discussion of the reservation system by looking at how its use relates to the use of the testbed as a whole. As shown in Figure 7, rises and falls in use of the testbed (as measured by the number of node-hours used per month) are correlated perfectly with rises and falls in submission of reservations. April 2018, CloudLab's busiest month to date (previously seen in Figures 1 and 3) also saw a large spike in reservations: an astonishing 193 requests were submitted that month, or more than six per day. During that month, there were 140k node-hours of approved reservations, as compared with 724k node-hours used in general, telling us that approximately 19% of all node-hours used that month were used through reservations. During the preceding January, a lighter month, these numbers were 67k, 552k, and 12%, respectively. Another place where the effects of the reservation system appear is Table 2: if we look at the entire time period, simple resource unavailability is the top reason for mapping failures. If we look at just the last year, however, when the reservation system was more stable, better advertised, and more heavily used, node shortages due to upcoming reservations have become more common than "simple" shortages. The April spike was followed by a similar increase in usage in September 2018.

We postulate that, as the use of the testbed approaches its total capacity, (or, as the free resources approach zero), the notional value of a reservation to a user grows super-linearly. By analogy to queuing theory, as the demand rate approaches the service rate, the expected wait time approaches infinity [20]. Facing the possibility that they may have to wait an unbounded amount of time for the resources they need to become available through the FCFS system, users have far greater incentive to submit reservation requests. This results in the pattern that holds true for the aggregate and also specific hardware types. The demand for specific types of nodes fluctuates over time, and users naturally adjust, using reservations only for the types that are in high demand. Overall, our analysis confirms that the reservation system constitutes a successful "social engineering" project on the part of CloudLab in that the system did change user behavior in the desired way: they use reservations heavily during periods of high demand, but then reservations "fade into the background" when they are not needed, letting the traditional FCFS model dominate.

4 Related Work

There is a body of literature focused on design and analysis of computing testbeds. The work that has shaped the research in this area includes the studies of large-scale experimentation environments such as PlanetLab [8], Grid'5000 [6], Emulab [16], Open Cirrus [5], and PRObE [12]. There are also recent studies that examine the Jetstream [33] "production" cloud for science and engineering research, the Chameleon [21] cloud computing testbed, and the Comet [34] supercomputer, among other facilities. These facility studies describe specific needs of research communities, document major design and implementation efforts, and share the unique lessons learned in the process of deploying and operating each system. Our work complements them by describing different aspects of facility operations and yielding insights into different kinds of design decisions. Studies of relevant commercial installations with similar amounts of detail are scarce.

Another relevant theme relates to using academic and commercial cyberinfrastructures to investigate systems topics and solutions with broad applicability, including the topological issues in testbeds [15], performance and repeatability [26, 22], failure analysis [24], individual subsystems such as disk imaging [19, 4] monitoring infrastructure [38], virtualization [16], and cloud federation [13], among others. Our study complements these by focusing on the way that the *control framework* (the software that manages, assigns, and provisions resources), and the abstractions it offers affect user experience and behavior. The key difference from the related work lies in the unique facility- and user-centered scope of our analysis; none of aforementioned facilities has been studied from this angle. Additionally, this paper describes CloudLab's functionality that extends the control framework used in GENI [25, 32], Emulab [39], and Apt [32].

5 Conclusion

Testbeds for computer science research occupy a unique place in the overall landscape of computing infrastructure. They are often used in an attempt to overcome a basic impasse [3]: as computing technologies become popular, research into their fundamentals becomes simultaneously more valuable and more difficult to do. The existence of production systems such as the Internet and commercial clouds motivates work aimed at improving them, but production deployments offer service at a specific layer of abstraction, making it difficult or impossible to use them for research that seeks to work *under* that layer or to change the abstraction significantly.

The design and operation of testbeds—and other IaaS infrastructures—benefits greatly from analyzing data about how these facilities are used. In this paper, we have presented new analysis of the way that one particular facility, CloudLab, is used in practice. This analysis, and the underlying dataset (which we have made public) have shown that user behavior is highly variable, bursty, and long-tailed. In addition, algorithms that may be thought of as being "deep within" the system have large, visible effects on user experience and on user behavior. Together, these findings point towards design decisions that more carefully take user expectations and behavior into account "end-to-end" throughout the entire facility.

Data and Code

Data and code used for our analyses are available at `https://gitlab.flux.utah.edu/emulab/cloudlab-usage` with the tag `atc19`. This data covers CloudLab's resource availability and events such as experiment instantiations.

Acknowledgments

We thank the anonymous USENIX ATC reviewers and our shepherd, Dilma da Silva, whose comments helped us to greatly improve this work. This material is based upon work supported by the National Science Foundation under Grant Numbers 1419199 and 1743363.

References

[1] Amazon Web Services, Inc. Amazon EC2 Spot Instances Pricing. `https://aws.amazon.com/ec2/spot/pricing/`.

[2] Amazon Web Services, Inc. Amazon virtual private cloud documentation. `https://docs.aws.amazon.com/vpc/index.html`.

[3] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the Internet impasse through virtualization. *IEEE Computer*, 38(4):34–41, April 2005.

[4] K. Atkinson, G. Wong, and R. Ricci. Operational experiences with disk imaging in a multi-tenant datacenter. In *Proceedings of the Eleventh USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Apr. 2014.

[5] A. I. Avetisyan, R. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. O'Hallaron, M. Kunze, T. T. Kwan, et al. Open cirrus: A global cloud computing testbed. *Computer*, 43(4):35–43, 2010.

[6] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, et al. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. *The International Journal of High Performance Computing Applications*, 20(4):481–494, 2006.

[7] M. Brinn, N. Bastin, A. Bavier, M. Berman, J. Chase, and R. Ricci. Trust as the foundation of resource exchange in GENI. In *Proceedings of the 10th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, June 2015.

[8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[9] Computing with HTCondor. HTCondor: Classified Advertisements. `https://research.cs.wisc.edu/htcondor/classad/classad.html`.

[10] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[11] D. Duplyakin, D. Johnson, and R. Ricci. The part-time cloud: Enabling balanced elasticity between diverse computing environments. In *Proceedings of the Eighth Workshop on Scientific Cloud Computing (ScienceCloud)*, June 2017.

[12] G. Gibson, G. Grider, A. Jacobson, and W. Lloyd. Probe: A thousand-node experimental cluster for computer systems research. *USENIX; login*, 38(3), 2013.

[13] N. Grozev and R. Buyya. Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, 44(3):369–390, 2014.

[14] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn. Case study for running HPC applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 395–401, New York, NY, USA, 2010. ACM.

[15] F. Hermenier and R. Ricci. How to build a better testbed: Lessons from a decade of network experiments on Emulab. In *Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, June 2012.

[16] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the Emulab network testbed. In *Proceedings of the USENIX Annual Technical Conference*, June 2008.

[17] T. Hoefler and R. Belli. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, page 73. ACM, 2015.

[18] D. Irwin, J. Chase, L. Grit, and A. Yumerefendi. Self-recharging virtual currency. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*, Aug. 2005.

[19] E. Jeanvoine, L. Sarzyniec, and L. Nussbaum. Kadeploy3: Efficient and Scalable Operating System Provisioning for Clusters. *USENIX ;login:*, 38(1):38–44, Feb. 2013.

[20] L. Kleinrock. *Queueing systems: Theory*. Wiley, New York, 1975.

[21] J. Mambretti, J. Chen, and F. Yeh. Next generation clouds, the Chameleon cloud testbed, and software defined networking (SDN). In *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*, pages 73–79. IEEE, 2015.

[22] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci. Taming performance variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA, 2018. USENIX Association.

[23] P. Marshall, K. Keahey, and T. Freeman. Improving utilization of infrastructure clouds. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 205–214. IEEE Computer Society, 2011.

[24] C. D. Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer. Lessons learned from the analysis of system failures at petascale: The case of Blue Waters. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 610–621, June 2014.

[25] R. McGeer, M. Berman, C. Elliott, and R. Ricci, editors. *The GENI Book*. Springer International Publishing, 2016.

[26] L. Nussbaum. Testbeds support for reproducible research. In *Proceedings of the Reproducibility Workshop*, Reproducibility '17, pages 24–26, New York, NY, USA, 2017. ACM.

[27] Rackspace Cloud Computing. Horizon: The OpenStack Dashboard Project. `https://docs.openstack.org/horizon/latest/`.

[28] Rackspace Cloud Computing. OpenStack: Open source software for creating private and public clouds. `https://www.openstack.org/`.

[29] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *ACM SIGCOMM Computer Communications Review (CCR)*, 33(2):65–81, Apr. 2003.

[30] R. Ricci, J. Duerig, L. Stoller, G. Wong, S. Chikkulapelly, and W. Seok. Designing a federated testbed as a distributed system. In *Proceedings of the 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (Tridentcom)*, June 2012.

[31] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 39(6), Dec. 2014.

[32] R. Ricci, G. Wong, L. Stoller, K. Webb, J. Duerig, K. Downie, and M. Hibler. Apt: A platform for repeatable research in computer science. *ACM SIGOPS Operating Systems Review*, 49(1), Jan. 2015.

[33] C. A. Stewart, D. Y. Hancock, M. Vaughn, J. Fischer, T. Cockerill, L. Liming, N. Merchant, T. Miller, J. M. Lowe, D. C. Stanzione, et al. Jetstream: performance, early experiences, and early results. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, page 22. ACM, 2016.

[34] S. M. Strande, H. Cai, T. Cooper, K. Flammer, C. Irving, G. von Laszewski, A. Majumdar, D. Mishin, P. Papadopoulos, W. Pfeiffer, et al. Comet: Tales from the long tail: Two years in and 10,000 users later. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, page 38. ACM, 2017.

[35] The Mass Open Cloud Team. Mass Open Cloud Web Site. `https://massopen.cloud/`.

[36] The CloudLab Team. CloudLab hardware. `https://www.cloudlab.us/hardware.php`, 2018.

[37] The CloudLab Team. User-controlled switches and layer-1 topologies. `http://docs.cloudlab.us/advanced-topics.html#(part._user-controlled-switches)`, August 2018.

[38] A. Turk, H. Chen, O. Tuncer, H. Li, Q. Li, O. Krieger, and A. 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.

[39] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI)*. USENIX, Dec. 2002.