Lessons Learned from the Chameleon Testbed

Kate Keahey[1] Jason Anderson[2] Zhuo Zhen[2] Pierre Riteau[3] Paul Ruth[4]
Dan Stanzione[5] Mert Cevik[4] Jacob Colleran[2] Haryadi Gunawi[2] Cody Hammock[5]
Joe Mambretti[6] Alexander Barnes[5] François Halbach[5] Alex Rocha[5] Joe Stubbs[5]
[1]*Argonne National Laboratory* [2]*University of Chicago* [3]*StackHPC Ltd*
[4]*RENCI UNC Chapel Hill* [5]*Texas Advanced Computing Center* [6]*Northwestern University*

Abstract

The Chameleon testbed is a case study in adapting the cloud paradigm for computer science research. In this paper, we explain how this adaptation was achieved, evaluate it from the perspective of supporting the most experiments for the most users, and make a case that utilizing mainstream technology in research testbeds can increase efficiency without compromising on functionality. We also highlight the opportunity inherent in the shared digital artifacts generated by testbeds and give an overview of the efforts we've made to develop it to foster reproducibility.

1 Introduction

The primary goal of computer science (CS) experimental testbeds is to support CS systems research by inventing and operating a scientific instrument on which such research can be conducted. Like in any other experimental science, such instrument is a critical tool: while we can conceive of all sorts of experiments, in practice we can carry out only those that are supported by an instrument that allows us to deploy, capture, and record relevant phenomena. The objective of experiment support can be considered along two dimensions: supporting the broadest possible set of experiments for the largest possible set of experimenters. The factors that influence the former include providing state-of-the-art hardware at appropriate scales and sufficiently expressive interfaces for allocating and configuring that hardware (i.e., deploying experiments). The latter is influenced by the cost of per user and per experiment support, but also the usability and familiarity of interfaces that lower the entry barrier for most users.

In this paper, we describe Chameleon, a testbed for CS research and education, and evaluate it from the perspective of the two dimensions outlined above. Chameleon gives users access to a broad array of state-of-the-art hardware, supports deep reconfigurability and experimentation at scale as well as isolation, preventing one experiment from impacting another. Since its public availability date in July 2015, Chameleon has supported 4,000+ users working on 600+ projects.

Unlike traditional CS experimental systems such as Grid'5000 [1], Emulab/CloudLab [2, 3], or GENI [4], which have generally been configured by technologies developed in-house, Chameleon adapted the OpenStack mainstream open-source cloud technology to provide its capabilities. This has a range of practical benefits, such as familiar interfaces for users and operators (or workforce development potential for those not familiar with OpenStack, as they acquire transferable skills), the opportunity to leverage the contributions from a large development community, and the potential to contribute back to that community in turn and thus influence infrastructure used by many users worldwide. Beyond practical benefits, configuring an experimental platform as a cloud also provides a direct answer in the debate over whether CS systems research can be supported on clouds, including potentially commercial clouds. More importantly, it also provides a means of influencing that debate through direct mainstream contributions, i.e., describing how a cloud needs to be configured to support this type of research. A secondary contribution of our paper is thus an articulation of a mainstream cloud configuration that yields a platform suitable for systems research.

Perhaps the most important lesson learned from Chameleon was that testbeds generate a wealth of experimental digital artifacts compatible with the testbed – such as images, orchestration templates, or more recently, computational notebooks – that can be used to re-play experiments. Testbeds are thus "readers" for digital representations of experiments. This creates an opportunity for developing a sharing ecosystem in which users can easily share and replicate each other's experiments and thereby another dimension in which a testbed can support research. This dimension is influenced by how easily experiments can be expressed in a shareable and replicable form – and then how easily they can be discovered and published. We introduced these mechanisms in Chameleon over the last year; while they have been around for too short a time to provide a comprehensive evaluation, we will discuss both their structure and potential.

In Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20). July 2020.

2 Chameleon in a Nutshell

The Chameleon testbed consists of two operating sites: one at University of Chicago (UC) and the other at Texas Advanced Computing Center (TACC). Our approach to hardware seeks to balance scale (i.e., support for HPC and Big Data experiments) and diversity. Scale was achieved via investment in 12 Haswell racks (2 at UC, 10 at TACC) containing a mix of compute and storage nodes, one with IB interconnect. More recently, they were augmented by 3 SkyLake racks (2 at UC, 1 at TACC) and 1 CascadeLake rack at TACC. The SkyLake racks are equipped with Corsa switches enabling experimentation with Software Defined Networks (SDN). The sites are connected by a 100G network supporting experimentation with large flows. This investment in scale is supplemented by smaller clusters of nodes supporting specialized experiment types: they include four types of GPUs (M40s, K80s, P100s, and P100 with NVLINK), FPGAs, low-power nodes (ARMs, Atoms, and low-power Xeons), and memory hierarchy nodes equipped with almost a TB of RAM memory, and a range of NVMes, SDDs, and HDDs. Over 4 PB in global storage capability is additionally distributed across the sites. About a year ago a Chameleon Associate Site (contributed on a voluntary basis) was added at Northwestern with a modest allocation of nodes equipped with 100G cards, expanding Chameleon's ability to support experiments with large networking flows. A fine-grained and rigorously up-to-date description of hardware can be obtained from the Chameleon Resource Discovery services [5].

Chameleon's capabilities are designed to allow experimenters to allocate and configure these resources at multiple entry levels: users can make allocations expressed as model-based constraints, allocate by node type, or point to a specific node, either on-demand or via advance reservations. Allocatable resources range over nodes, networks, and IP addresses. Resource configuration is supported at bare metal level and supports custom kernel boot. The latter is achieved via the support of whole-disk images, which include a kernel, a partition map, and a bootloader. Reimaging takes places as a sequence of booting to a provisioning ramdisk via PXE, image transfer to node's disk via iSCSI, and a local boot. This allows kernel developers to replace the kernel as needed for experimentation; they can also use serial console access at boot time for debugging and snapshotting to save the revised image. In addition, Chameleon supports network stitching and SDN experimentation via the recently introduced Bring Your Own Controller (BYOC) [6] abstraction built on top of the Corsa DP2000 series OpenFlow switches. Users can create SDN networks that are isolated from each other and the testbed management networks by dynamically provisioned virtual forwarding contexts (VFCs), each with its own controller and subset of ports/VLANs, that are then connected to the nodes and external circuits reserved by the user.

Configuration can also be carried out at multiple entry levels: users can reimage individual nodes and then configure their experiments manually, use orchestration capabilities to render complex (potentially distributed) experiments that can be automatically and repeatedly deployed, or use Jupyter notebooks in combination with one or both of these mechanisms. Monitoring is supported partly by OpenStack Gnocci service that has been augmented to provide capabilities such as e.g., fine-grained power monitoring. The CHameleon Infrastructure (CHI), which implements these capabilities, is built primarily on top of the mainstream open-source OpenStack platform [7] (with extensions and contributions from our team), but also integrates resource representation, versioning and management tools from the Grid'5000 project [1], and network management tools from the ExoGENI [8] project. The implementation of CHI has been described in detail in [9].

Chameleon projects are given allocations of 20,000 SUs (one SU is a node hour) for six months; this aims to strike a balance between what might represent a reasonable amount of time needed to obtain a result (a very variable measure!) and 1% of 6 months' capacity of the initial testbed deployment (i.e., if everybody was using their allocation at the same time the testbed could support no more than 100 projects in that amount of time). Allocations can be recharged (more SUs) or renewed (longer time). In addition, user's leases on the system (i.e., the length of time for which resources can be allocated) are limited to at most 7 days; these can also be extended either programmatically or via interactions with Chameleon operators.

Figure 1 provides a rough summary of Chameleon's growth in hardware and utilization, as well as users and projects. Over the almost four and a half years of its operational life Chameleon has supported 4,331 users across 655 unique projects representing a broad array of research and educational uses. The growth of Chameleon usage has been steady since the project start, with about 7 new projects per month in the first years of operations, accelerating to eleven new projects per month in 2019. Incremental hardware investments have been keeping up with that growth. We also observe a similar trend to that already reported by [3] where usage is lower during summer and winter breaks and peaks during the semester and important conferences such as the supercomputing conference series.

3 Most Experiments for Most Experimenters

We now evaluate the key decisions made in Chameleon from the perspective of supporting the most experiments for the most experimenters. To evaluate the former, we will look at whether the hardware and capabilities we provide are sufficient to support experiments in our community; for the latter we will look at quantitative measures such as the numbers of users and experiments the tested can simultaneously support and the types of projects it attracted.

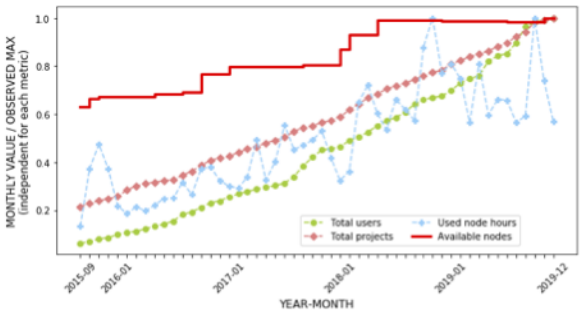


Figure 1: Increasing usage and capacity of the testbed over time. Monthly values are normalized to the maximum observed value over time. Scalings are independent for each metric.

3.1 Experiments

In this section we describe and evaluate both our strategy for hardware configuration and the technical decisions we made to provide access to this hardware such that it supports the broadest possible set of experiments. Unless we specify otherwise, the data reflects period between 09/01/15 to 12/31/19.

3.1.1 Hardware

Where resources are limited (as they inevitably are in academic testbeds) it is important that hardware investments strike a balance between scale, i.e., support for large-scale experimentation, such as HPC, and diversity, i.e., support for a broad range of resources on which different research questions can be tried. Given these considerations, our hardware investment strategy was to build up scale at the beginning of the project (the original 12 rack Haswell deployment) and then introduce diversity gradually with an eye towards the types of projects that were requested the most, and the type of resources that were utilized the most. We also held back a "strategic reserve" of hardware investment to develop the testbed as innovative hardware solutions emerged.

To understand how well we satisfy the need for scale we looked at the size of lease requests for the testbed in general, and the large Haswell partition in particular. By far the most leases on the testbed are requests for a single node, constituting 67.19% of testbed leases (63.24% of Haswell leases) with only 5% of requests exceeding 10 nodes (11% for Haswell). The largest lease on the testbed was 120 nodes. Overall, this means that our investment in scale did indeed broaden the set of supported experiments while also allowing us to support more simultaneous leases.

To understand how well we support the need for diverse hardware types, we show in Figure 2 node availability against the percentage of time that it was available, first for the overall time since resource installation, and then during the busiest and least busy month as measured by highest and least utilization of the testbed as a whole (Cascade Lake was installed very recently and is not shown). By far the most used resource types are the four GPU types, with the two newer GPUs (GPU P100) more in demand than the two older ones (GPU K80 and M40), followed closely by the memory hierarchy nodes. On the other end of the spectrum, ARMs, Atoms, and low-power Xeons—targeting specialized power experiments—are used the least. The large-scale resources, Haswells and Skylakes, as well as the FPGAs, occupy the middle.

While these categorizations are roughly consistent across specific time periods and overall, some important details are different. For example, the availability of the GPU P100 NVLINK is less in both most and least used month than the overall availability graph would suggest; this is due in significant part to the fact that resources are typically much less utilized immediately after they are introduced as the knowledge of their availability has yet to be absorbed. In general, periodic usage patterns might fluctuate due to specific conference deadlines or teaching/workshop use.

One lesson learned from our experiences is the need for adaptation. Overall, introducing diversity to the testbed gradually allowed us to make changes to respond to community demand, shaped by the evolving nature of the research needs. For example, based on the response to our early K80 and M40 deployments driven by emergent interest in machine learning, and combined with less demand for a low-power processors, we re-budgeted some of our planned investment between those categories and also invested our "strategic reserve" into more GPUs. This meant that we were able to provide ample support for low-power experimentation while keeping up with emergent demands. Because of periodic fluctuations it is hard to formulate recommendations on when such change should be considered; based on our experiences it is likely to occur when usage moves to the area of the graph between the large-scale and memory hierarchy resources.

3.1.2 Capabilities

Resource Description. Much of the usefulness of the testbed relies on the manner in which users gain access to hardware. To do that, users have to describe the resources they need. Commercial clouds offer a variety of "instances", sometimes with vaguely described properties (e.g., "high I/O bandwidth"). Experimental testbeds allow users to choose a specific hardware type and sometimes seek to ensure that all servers of the same type have comparable performance [3,10]. In contrast, we take the view that performance variability is a fact of life and often a research topic in itself. Our approach therefore is to allow users to choose resources on a range of levels: from a model description expressed as a set of constraints (e.g., "memory of at least X" or "X nodes situated on the same rack"), through describing hardware type (e.g., a Skylake node), or by referring to a specific node.

Analyzing Chameleon lease requests made between 09/16

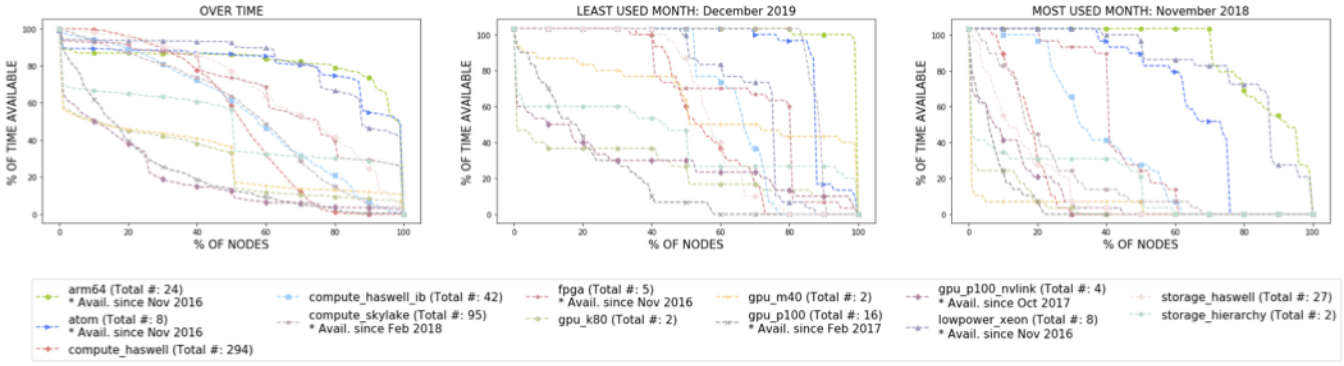


Figure 2: The usage of Chameleon hardware types expressed as resource availability plotted against the percentage of time in which it was available for (a) overall availability since resource installation, (b) the least and (c) most utilized month.

and 11/19, we find that the majority (89.24%) were created using a single constraint (the rest are leases using either no constraint or multiple constraints; 9.5% and 1.26%, respectively). Of the single constraint leases, 90.18% were created by specifying the hardware type and only 3.38% specify node uid (a specific node). However, when we look at single-constraint leases that are created more than 7 days in advance, the percent of leases with node type constraint drops to 59.91%, while the percent of leases with uid increases to 18.45%, which shows that users who need a specific node are willing to wait for it rather than replace it with something else.

Overall, a hardware type is clearly the most requested quality. Specific nodes are requested relatively infrequently, though some experimenters do need them and model-based descriptions based on high-level constraints are rare. While it is often tempting to think that a model-based description is the ideal, the following anecdote illustrates the limitations of this approach. Using a model-based request ("memory greater than X"), one of our users was assigned an ARM node; this led to a difficulty since although this was a correct match for the experimental model, the user's tooling did not work on ARMs. In subsequent conversation with our support staff the user was advised to browse our discovery services, found the Chameleon memory hierarchy nodes, and concluded that with those nodes he would be able to design a more ambitious experiment. We derive two lessons from this: first, models are not all that is needed for determining the right experimental resources (logistical concerns need to be taken into account as well); second, resource discovery phase is an essential part of an the experimental workflow and critical to taking full advantage of the testbed.

Allocatable Resources. Another matter of interaction with the testbed consists of being able to obtain resources in a timely manner. Commercial clouds use the metaphor of an "endless resource" always available on-demand – in practice no resource is of course endless even in commercial clouds (e.g., the current initial limit at AWS is 256 VCPUs [11]) – though some are sufficiently large. Thus, the ability to gracefully deal with availability limitations is important, particularly in academic clouds where we try to maximize small scale resource investment.

To provide such ability we introduced the abstraction of an allocatable resource described in [12]. In brief, an allocatable resource allows users to manage a resource allocation in terms of both time and resource assigned to the allocation (i.e., when an allocation starts and ends, and well as how many nodes belong to it). In particular, the ability to manage the start time is sometimes referred to as "advance reservations" and is a generalization of on-demand availability provided by commercial clouds (i.e., on-demand is an advance reservation with start time set to "now"). The resources can be of various types and can be managed to fulfill different conditions; in Chameleon currently the managed allocatable resources consist of nodes, VLANs, and public IP addresses. The implementation of this capability and its contribution to OpenStack was initiated by the Chameleon team.

As [12] demonstrates, allocatable resources, and advance reservations in particular, are useful in providing access to scarce resources: the scarcer the resource, the more likely users are to make an advance reservation to ensure availability, and the longer in advance this reservation is likely to be made. In figure 3, we provide a more detailed demonstration of this concept: we scatter plot all leases made for three types of resources: Haswell compute nodes at TACC (largest partition), Skylake compute nodes at UC (largest partition), and our GPU P100 cluster (16 nodes), noting the number of nodes requested and the reservation lead time.

We see that the GPU P100 nodes (one of the most utilized resources per Figure 3) have by far the longest advance reservation lead times even though only very few users reserve more than one node. On the other hand, Haswell@TACC, used for experiments at scale, show a significantly higher proportion of leases with multiple nodes (with the max being 85). While many of them are created with some lead time, it was

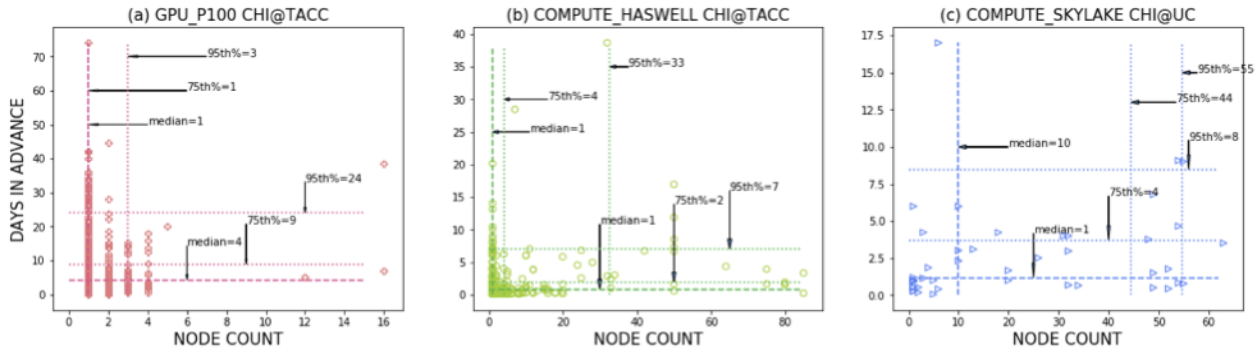


Figure 3: Node counts vs. advance reservation lead time of advanced leases for (a) GPU_P100, (b) Haswell, and (c) Skylake

possible to create some large leases on-demand; they are correlated to summer use (low utilization) and no leases with size in the 95th percentile were available after 2016 as the testbed became popular. This trend is even more pronounced when looking at the Skylake nodes, which are a scarcer resource than Haswells (64 versus 278 for largest partition) but also support experiments at scale. Advance reservations are thus useful in managing two types of resource scarcity: very scarce resources (e.g., GPU P100) will require high lead times, but relatively abundant resources (e.g., Haswells) can become scarce if a large reservation is requested.

The end time of a resource reservation can also be extended programmatically, although according to our policies this can only take place within 48 hours of lease expiration and of course only if the resource is not reserved by another user (a policy exception can also be requested via the help desk.). This last consideration limits the practical usefulness of this feature in the case of scarce resources, as it is likely that they will have been reserved by the time the extension window becomes active. For this reason, programmatic extension requests have only been successful in relatively few cases in practice, e.g., during the last year at UC only 5.4% leases got extended in this way; though half of them more than once.

Separation of allocation and configuration. Unlike commercial clouds where resource allocation and image deployment are one operation, Chameleon separates them to allow users to map different images to an allocation. This capability proved relatively popular with experimenters: 22.07% of the allocations had more than one instance deployed, and roughly half of those (9.17% of all the allocations) had more than one unique instance (i.e., associated with a different image) deployed; the average number of instances deployed within one allocation is 1.45 (with max being 12) and of unique instances is 1.12 (with max being 10).

Networking. Network isolation is an important property in that it allows non-standard IP configuration, potentially disruptive services, or security experiments that analyse or intentionally attack other nodes on the isolated network. Similarly to many of the base Chameleon features, we implemented this via standard OpenStack services; within each geographic site, users can create Chameleon networks that are isolated within unrestricted VLANs (i.e., no firewalls) and logically connect any number and type of nodes.

Chameleon's "Bring Your Own Controller" (BYOC) [6] capability extends these isolated networks by providing direct user control of network flows and configuration via a standard or customized OpenFlow 1.3 controller. Though OpenStack did not support BYOC out of the box, its modular design enabled us to implement the feature as a custom Neutron plugin, which enables users to specify the IP and port of the user's OpenFlow controller (whether provisioned on Chameleon or externally). Chameleon uses Corsa DP2000 series switches to dynamically create isolated OpenFlow 1.3 network slices in hardware operating at full performance (10 Gbps node ports and a 100 Gbps uplink); this is in contrast to an approach which provides coarse control of whole OpenFlow switches from a static pool of hardware as implemented by CloudLab. BYOC enables many experiments from basic hands-on educational experiences with OpenFlow to advanced networking experiments that optimize performance of network traffic or identify and remedy security breaches by analysing low-level traffic behavior. Despite the fact that BYOC networking is new and targeted at highly specialized and advanced users, there have been already been 11 unique projects (representing 4% of active projects over the time period) that have deployed OpenFlow experiments on the Chicago site alone.

Networking experiments on Chameleon are not limited to the Chameleon testbed alone. Users can create dedicated layer 2 circuits between Chameleon networks and external facilities such as ExoGENI, campus laboratories, public clouds, and other Chameleon sites; creating dynamic connections of this type is often called "stitching" [4]. ExoGENI provides a dynamic stitching service that connects a wide collection of participating facilities, including Chameleon. Chameleon users can create isolated stitched links between their networks (including BYOC networks) and ExoGENI and can extend those links across ExoGENI to remote facilities. In addition, multiple stitched links can be connected to a single Chameleon

network, enabling user-controlled wide-area multi-path routing experiments. To date, 22 unique projects (representing 8% of active projects over the feature's life time) that rely on network stitching have created 920 stitched links between remote facilities using ExoGENI.

Orchestration. Once a lease is created, users can configure it using Chameleon provided images, create their own (often, but not always, derived from Chameleon provided images), or use complex appliances [13], representing concepts such as a cluster, a cloud, or a networking experiment, that can be deployed "with one click" and then repeated in future deployments, similar to CloudLab profiles [3]. These experiments are configured using images in conjunction with Heat orchestration templates [14] that define how to deploy and contextualize [15] them to create the desired integrated environment and processes. Using Heat, an active Chameleon topology can be modified (by e.g., adding or removing nodes, altering MTUs on the network, or changing a post-boot step for a particular node) through changes to the underlying Heat template; the orchestration system applies the delta without forcing re-creation of the entire topology. The choice to decouple allocation and configuration made this functionality easier: because a set of resources is allocated explicitly to a user for a time period, any topological or software/firmware configuration can vary without breaking the researcher's assumptions about the underlying hardware.

Consistent with our decision to make the testbed available at various levels of access, the use of orchestration is optional for Chameleon users. Using orchestration/Heat provides repeatability at the cost of an additional up-front investment (i.e., developing an orchestration template) and thus tends to be used in later stages of a project when experimental configuration is settled on. Since users often develop orchestration templates by modifying existing ones, the Chameleon project provides 3 complex appliances (images+Heat template) and another 14 individually-supported complex appliances are hosted on our appliance catalog; the most popular are MPI bare-metal cluster (MPICH3), Ryu OpenFlow Controller, and OpenFlow - QuickStart appliance. To date, 81 Chameleon projects have used Heat, among those 20 were in systems, 17 in education, and 12 in networking, with the rest ranging over a variety of topics including security, power management, and others. The usage data since 10/16 when this feature was introduced show a steady upward trend in orchestrated deployments: 94 (2017), 155 (2018), and 405 (2019), though more recently users were increasingly using Jupyter notebooks or scripting for orchestration. Overall, while orchestration is an advanced feature and thus the uptake is slow, it is proving a useful tool to express a range of experiments.

Configuring complex experiments, even when automated via orchestration, can still take significant time, as packages need to be installed, configuration scripts run, and tests executed. Thus, while separating allocation and configuration proved a successful decision, users often ask for functionality

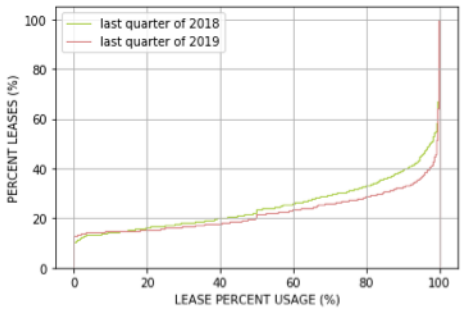


Figure 4: The cumulative distribution functions (CDFs) of lease percent usage for the last quarter of 2018 and 2019.

that effectively recombines these actions, i.e., automatically triggers the deployment of an orchestrated experiment when a user's advance reservation comes into effect. To provide it, we introduced the automated deployment feature [16] in 01/19 that automatically triggers the deployment of experiments orchestrated with Heat. So far it has been little used while still being often requested; this likely points to the need for more energetic education efforts as well potentially to the need of extending this capability to other orchestration methods, in particular the increasingly popular Jupyter-based approach.

Managing User Behavior. Perhaps the most significant challenge of operating Chameleon, common to all academic cloud resources, is ensuring fair sharing of the resource. Unlike in HPC datacenters, where actual resource use is tied to the submission of a specific program (such that if the program fails the resource grant is withdrawn), access to cloud resources is given out on an open-ended basis. Commercial clouds create incentive to use no more than is needed by charging for the duration of access. Academic clouds, such as Chameleon, seek to provide a similar incentive via allocation policies (see Section 2); this is generally less successful since users can recharge or renew their allocation relatively easily. In fact, we found this measure to be inadequate on its own early in the project as users created leases to "put a hold" on resources that then went unused, significantly reducing testbed capacity for others. This led us to introduce the (extensible) 7 day lease limit described earlier which improved fair sharing at the cost of imposing extra overhead on experiments that legitimately need more than 7 days. However, we still see leases on the testbed that merely hold resources without using them.

To manage this situation, we introduced a policy whereby users are expected to start using their lease within a certain amount of time from deployment. This policy is enforced by a "lease reaper" (deployed in 09/19) that monitors the use of a lease, sends a reminder to users not using their leases, and terminates them if still unused after a certain period of time.

Figure 4 shows a comparison of lease usage for the last quarter of 2018 and last quarter of 2019 (before and after

the lease reaper was introduced). While it is natural that resources in a lease may be unutilized for some time (e.g., between deployment of different images), we see that in 2018 (without lease reaper) a large percentage of leases is underutilizing resources to a significant extent. However, in 2019 (with lease reaper) the situation improved: it went from 40.79% to 45.95% fully-used leases and from 66.92% to 71.19% 80%-used leases. More sophisticated ways of ascertaining if a lease is actually being used are likely to tighten the gap between allocated and used leases further, but at the same time become open to "gaming" by users emulating lease usage via artificial means, reducing their effectiveness. Thus, striking the right balance of incentive and access to promote fair sharing in academic environments is still an open question.

Summary. On the whole, we found that the key design decisions we took, whether by introducing new hardware or new capabilities, led to expanding the set of experiments available to our user community, and were thus quickly embraced. While we still receive new feature requests, they are increasingly smaller and come less often. At the same time, a significant lesson learned in the process of operating Chameleon is that no research testbed is ever complete because the set of desired experiments is constantly expanding. As the research frontier advances emergent research opportunities create the need for new scientific instruments – or new features in existing scientific instruments – to support their exploration. While setting aside a strategic reserve in hardware served us well, the extent to which this phenomenon drove development was surprising: not only did we need to adapt the system to leverage new opportunities in hardware (such as e.g., providing the BYOC capability on top of the Corsa switches), we needed to develop abstractions (such as e.g., the allocatable resources) to integrate those extensions in the experimental workflow. The most significant types of experiments that Chameleon does not support yet are thus in emergent topics – for example, on the intersection of Internet of Things (IoT) and cloud computing as well as machine learning.

3.2 Experimenters

Supporting as many users as possible will be influenced by two factors: how many users a testbed can sustain by managing the cost of users and experiments and how many users it can attract by adapting itself to the needs of different communities; this section will discuss how well Chameleon was able to achieve both.

Isolation and Automation. Providing an appropriate level of isolation captures an important trade-off: it should be fine-grained enough to divide the testbed efficiently between multiple experiments – but also coarse-grained enough to satisfy the isolation needs of a specific experiment. Since the granularity often goes with the level of isolation a specific mechanism provides [12] we must be careful to not sacrifice the required level of isolation; at the same time we want to

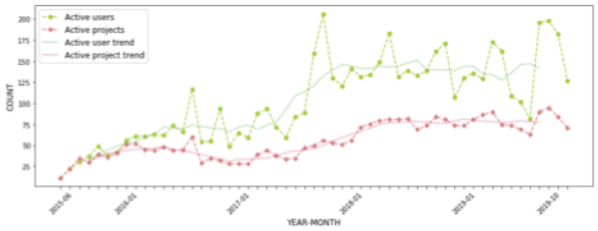


Figure 5: The numbers of active users and projects over time. The trend lines average the number of users/projects over 6 month period.

serve as many users as possible. In Chameleon, we navigate this trade-off by configuring the bulk of the testbed with CHI while setting aside two racks (originally three) provided as a standard OpenStack/KVM cloud. This finer-grained system isolation that this alternative cloud provides means that multiple user VMs can be deployed on one node instead of allocating a whole bare metal node (though it does not provide the performance isolation that a bare metal offers). Because of this efficiency, we were also able to offer a different policy: users can make open-ended deployments on the KVM partition while CHI (bare metal) leases are limited as to 7 days at most. Last but not least, it is more suited to less-experienced users.

Our data indicate that 209 Chameleon projects (22.94% of all Chameleon projects) used our KVM partition at least once. Among those projects, 12.92% are projects in CS education. About 26.16% of our total number of users used the partition, most of them assigned to an educational project. Most VMs (71.52%) are deployed for an hour or less and only 3.18% leverage the ability to claim testbed resources for more than one week. The median daily count of deployed VMs is 344 (with max/min of 1490/29); each of those would have likely occupied a bare metal node otherwise. All those statistics point to significant educational use; given the number of users and projects overall this seems a good investment for what currently constitutes only 16% of our total Haswell system and even smaller fraction of the overall testbed.

Supporting Volume. While isolation method determines the unit of sharing, the largest factor in the ability to support as many users as possible is automation since it lowers the per-user and per-experiment cost. We noted earlier that Chameleon is a production testbed, i.e., a testbed that supports production services that yield individual/breakable testbeds. Consistent with this definition, we define Chameleon testbed functions as only those experiments that are accessible to users in an automated manner (while we also support experiments requiring manual intervention from operators and special requests, we consider them support functions, not testbed functions). We now examine indicators of how much user volume the testbed can support.

We first asked how many active users the testbed supported

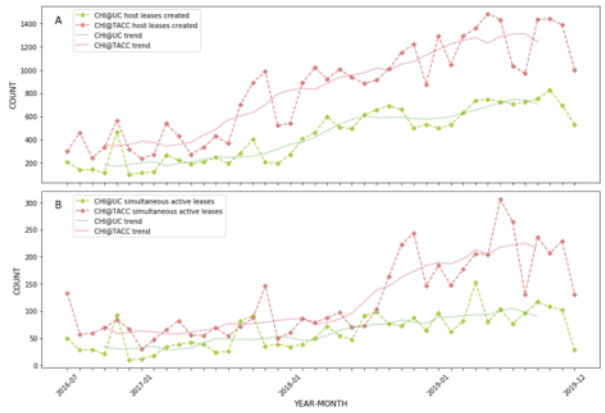


Figure 6: (A) Total leases created each month. (B) Maximum simultaneous active leases by month. The trend lines average the number of leases over 6 month period.

overtime. We see that numbers of active users follow the general pattern of slow and busy months (semesters versus breaks) that we have already seen in Figure 5. However, although we saw that the cumulative number of users was rising, it is interesting to note that the average number of active users and projects grew significantly about two years after the testbed has became available (fall of 2017). It is hard to pinpoint this to any one reason but possibilities include a lag that it takes for a new testbed to become established, the incremental introduction of features that broadened the set of supported experiments, and our first Chameleon User Meeting held just before the trend increase. At peaks, the testbed supported about 200 active users; this is twice as much as the lower bound that our allocation policy is based on (Section 2); luckily not all users are living up to their allocations!

We then asked how many leases and simultaneous leases users were able to create on a per month basis. The result for unique leases is shown in Figure 6 (A). We see that the trends are consistent with the number of active users reflecting the usage patterns in a similar way and picking up around the same time, though the growth trend is still continuing. The result for simultaneous leases (i.e., for each month, we found the max number of leases happening at the same time) is shown in Figure 6 (B); we see that the testbed has sustained up to 300 simultaneous ongoing experiments, but the trends picked up about a year later than trends for active users and experiment counts; it is clear that the testbed is now becoming more saturated.

Support Cost. Another metric important in the discussion of any testbed is support costs, specifically the time effort required by the team. Chameleon users have submitted 3,167 technical help desk tickets, averaging roughly 13 tickets every week and less than 1 ticket per user. On average users receive a reply within 15 hours and their issue is completely resolved within 2 weeks. These trended down over time: in 2019 the average response time was 16 hours, while the resolution time was 6 days. The costliest tickets concern hardware failures, which are often difficult to diagnose and/or require ordering new parts and performing maintenance. Cutting-edge or non-standard hardware and firmware also pose problems for support staff, as documentation might not be extensive (or even exist) and having expertise at hand is unlikely. In all cases, an ounce of prevention is worth a pound of cure: we have deployed or implemented an operational model integrating early detection (e.g., daily/hourly "happy path" tests of common user flows, live-monitoring with Prometheus [17], and a catalogue of alertable issues and resolution steps) and automated remediation (in the form of "hammers", i.e., bots that periodically check for and fix irregularities in testbed usage and performance). We additionally automate most common operator tasks, such as building new base images, deploying patches or configuration changes to the control plane, and taking nodes in or out of maintenance. Details of Chameleon operations has been published in [18].

Another indicator of cost in a system where users are given significant privileges is the cost of security management. We employ a range of standard security practices designed to make the system more secure: project PIs are vetted and assume responsibility for users on their projects; we provide base images configured and maintained by our team with reasonable security defaults and SSH key pairs for authentication; the management network is isolated from tenant networks; and we use intrusion detection system (IDS) alerts across the entire deployment. Since the system went public we've had a number of security incidents; most are caused by users either unknowingly or deliberately shirking best practices, e.g., using images with a known administrative password or using outdated software with known exploits – the latter sometimes to be compatible with benchmarks and other research software. The most typical types of exploits result in activities such as distributed denial of service (DDoS) attacks (~40% incidents) or bitcoin mining (~36% incidents). Those are typically identified via IDS systems operated on sites and trigger well-defined security procedures in response usually involving user/PI cooperation. So far, we've only had one incident involving malicious users in the first quarter of 2020 which shows that our PI vetting methods work well on the whole. Considering the nature of attacks to date the greatest improvement would probably be effected by more user training in techniques such as setting up bastion host for when insecure researchware has to be used as well as general training in operational security.

Community. Chameleon users come from 168 different institutions, the vast majority of which are US colleges and universities from 40 states and Puerto Rico, including 11 minority serving institutions. While most of the research projects we support are in computer science, 54 identify themselves as being from outside of computing disciplines, primarily in life sciences, astronomy and other fields. By analyzing project

abstracts we see that roughly 12% of all projects relate to cybersecurity, 20% are involved in machine learning, 10% are in edge computing or IoT applications, 5% are doing research work relating to containers (scheduling, virtualization, or performance), 2% are investigating software-defined networking. Several hundred grants are reported by users as the source of funding; the vast majority of these are from NSF, and within those the vast majority are spread among all divisions within CISE. About 5% of grants are supported by the DOE, DARPA, or the Air Force Office of Sponsored Research. A handful of projects are supported by industry, and several more have international support.

Publications are a complicated metric to track as they tend to be a lagging indicator; many happen after the allocation is complete, when there are few incentives for users to report them to the project. Further, the primary way we capture publication counts is through self-reporting when users seek a renewal of their allocation; while this is an incomplete method it still represents a useful lower bound. Through 2019, users have self-reported about 275 publications relating to their work on Chameleon, with many projects still active. There are 75 referred journal articles among these, with close to 200 conference publications. The spread of publications is fairly wide, with no clear concentrations in particular conferences or journals. As would be expected for a testbed, growth over time is dramatic and lagging: of the 75 journal papers, only 3 were published in the first project year, 8 in the second, 10 in the third, 23 in the fourth, and 31 have been published so far in the fifth year with several months remaining.

In addition to research usage, there is substantial educational use of the system. 45 projects support classroom instruction, often multiple classes over multiple semesters; all but four of these projects support courses in CS departments at 41 different schools. In total, the education projects have used about 9% of the total time available on Chameleon to date (roughly 675,000 node-hours), and they represent about 9% of the total projects – so an average classroom project uses about 15,000 node-hours, matching the usage of a typical research project.

One measure of the satisfaction of the user community is persistence. Projects are initially allocated time on the system for 6 months, and at the end of the year they must seek a renewal or extension. Of the projects that have reached the end of their initial year of allocation, about three-quarters of all projects have sought to renew their allocations, indicating that they find value in the use of the system. Many projects seek multiple renewals – in fact thirty-three of the original projects from the first year have been renewed multiple times and counting!

4 Building a Testbed on Top of Mainstream Cloud Implementation

When the Chameleon project started, we were presented with the unique opportunity of building the testbed on top of the then maturing cloud infrastructure: the first version of the OpenStack Ironic component [19], implementing bare metal reconfiguration (which we knew would be an indispensable capability of the system) has been released a few months prior to the project start, and while not yet an official part of the system has already been used in some bare metal deployments. The chance to base a testbed for cloud computing research on a mainstream open source implementation held out many possibilities, but will it be enough to support all the experiments that needed to be supported in the way they needed to be supported? After thorough evaluation of the system and development of a few alternative risk-mitigating strategies we decided that capabilities we needed were there—or were within reach in that they could be developed by our team. This section presents an analysis of the advantages and cost of taking this approach.

One practical benefit of using OpenStack is that it provides familiar interfaces to users and operators. The 2018 OpenStack User Survey [20] (most recent) included 858 OpenStack deployments across 441 organizations and 63 countries; of those organizations, 13% were categorized as academic or research-oriented. Those include major scientific institutions such as CERN [21], NeCTAR [22], and NASA JPL [23], and a formal Scientific Special Interest Group (SIG) [24] for OpenStack's use in science domains has existed since 2016 [25]. Since the introduction of the OpenStack Administrator certification three years ago, 3,000 individuals in 77 countries have taken the test [26]. All this not only creates a base of familiarity with OpenStack for users and operators – but also ensures that such familiarity is a transferable skill and thus valuable for workforce development. Finally, the Net Promoter Score [27] of 41 reported by the survey (up from 25 in 2017) indicates that the OpenStack environment continues to improve in terms of usability and that users enjoy the experience overall.

Another benefit of working with a mainstream platform is the ability to leverage and adapt the work of a large community which helps keep our development and operations costs down. Over 8,400 individuals have contributed code to OpenStack, with 1,000-2,500 contributors participating in each major release [28]. Leveraging their contributions, over the lifetime of the project we were able to offer our community new key features such as whole disk image boot, multi-tenant networking, serial console integration, support for non-x86 architectures, and user-customizable firewalls—simply by upgrading to a new OpenStack version. Future capabilities already possible due to upstream contributions include self-service BIOS customization and detachable remote storage; both have been common user requests. From the

operator's perspective, deploying and managing Chameleon was made simpler and more reliable by the Kolla project, which provides a packaging of OpenStack as Docker containers [29] and a set of highly-configurable Ansible provisioning scripts [30] to orchestrate the setup and maintenance of the deployment. Further, ~6,000 individuals have been involved in reviewing all code changes [31]: thus, by using a mainstream infrastructure we also benefit from a built-in large-scale quality control mechanism. No less valuable has been the access to the existing documentation and support systems within the community: the openstack-discuss mailing list [32] sees between 500-1000 messages each month, the OpenStack Q&A forum [33] has over 18,000 answered questions, and on many occasions we were able to get a workarounds or patches for bugs within days simply by filing a ticket to the official tracker.

The flip side of leveraging contributions of others is the opportunity to contribute to and shape a mainstream infrastructure. On a practical level, this magnifies our investment in the infrastructure and our broader impacts as any new features and additions we make to OpenStack are also impacting communities beyond the testbed. Because its hardware resources have always been constrained relative to the demands of its user community, Chameleon required a system for allocating and managing resources (including advance reservations), which our team implemented by reviving and significantly improving the OpenStack Blazar project. This attracted the interest of others and we were subsequently able to partner on development with contributors from NTT (Japan) and DOCOMO Euro-Labs (Germany) who use the component in Software Defined Networking applications. As a result of this collaboration, Blazar became an official top-level OpenStack component in 09/17 and has been included in each OpenStack releases since Queens. Other significant new features developed by the Chameleon team include bare metal snapshotting and enablement of slice creation via integration with ExoGENI stitching implementation; in addition, we made many smaller contributions in the form of bug fixes and patches.

These advantages are offset by some costs. Since it solves a complex problem, OpenStack is complex; thus operating, and in particular extending it, requires both development skills and deep expertise in the underlying systems and concepts, putting pressure on operator hours and level of skill. The most problematic manifestation of this complexity in our experience used to be OpenStack upgrades; this has improved significantly with tools that aid operators in these tasks, such as the aforementioned Kolla project. The size and structure of the open source community imposes its own overhead and requires commitments both in process and in time: patches must be reviewed, meetings must be attended, changes must be formally proposed and approved, and documentation and tests must be written. These commitments are the price of admission to an open-source community that, in our experience, ultimately returns the investment many times over in the form of support, debugging, development, and partnership.

Looking beyond practical benefits, building on top of a mainstream infrastructure helps settle a point of intellectual interest in that it provides a direct answer to the question: can clouds support CS system experimentation? While different clouds will of course be configured differently, Chameleon represents a configuration that satisfies this condition; we describe this configuration in this paper but it is also expressed in practical form via code, recipes, and settings that are publicly available and suitable for replication whether in academia or commercially. Like the Chameleon interfaces we support, that recipe has multiple "entry points": users may elect to simply install OpenStack with our contributions—they may elect to replicate the Chameleon configuration in every detail—or they may choose something in between. We facilitate this by providing a packaging of Chameleon Infrastructure (CHI) that contains not only OpenStack but also extensions including Grid'5000 and ExoGENI additions, as well as an operational model we developed that makes clouds of this type cost-effective to provide. This packaging, called CHI-in-a-Box [34], has recently been installed at Northwestern University and augmented Chameleon capabilities by providing modest but unique networking hardware.

5 Fostering Replicability and Sharing

Perhaps the most important lesson learned from Chameleon is that testbeds provide not only a platform for instruments but also generate shareable digital artifacts such as images, orchestration templates, datasets, tools, notebooks, and others. Those artifacts typically represent either a complete experiment or an important part of one and can be used to reenact it on the testbed on which it was created. For example, over the lifetime of the Chameleon testbed, users created 120,000 disk images and 31,000 orchestration templates that can be used for such purpose. This presents an opportunity: since the generated artifacts can be used to repeat experiments, sharing them should allow others to repeat experiments, introduce variation, or extend experiments more easily. We posit that fostering that sharing will contribute to the overall goal of providing a scientific instrument to advance CS systems research by both reducing time to discovery and providing a more fertile ground for sharing of ideas. The question arises how specifically experiments should be represented and structured—and then how specifically they should be shared.

Chameleon has supported a variety of mechanisms to aid repeatability over its lifetime. First, the Chameleon hardware is versioned, which allows users to easily identify any changes which would introduce variation. Users can also version the images they configure, and publish them in a catalog. Since this still requires a user to keep track of which appliances were deployed on which testbed version, we introduced a system, called Chameleon's Experiment Précis [35], which captures all the distributed events generated by a user in the testbed,

and presents him or her with a summary (a précis) of their experiment. Then, working with an accurate and impartial record of their work, the user can filter or preview the events to include only the relevant ones. The précis data can be used to generate a description of the experiment in English, or potentially an actionable description of the experiment in the form of orchestration templates or commands that will reproduce the experiment. Generating Heat orchestration templates in this way is in fact the objective of an OpenStack Flame [36] tool. Overall, the Experiment Précis is somewhat analogous to a shell's "history" command with the critical difference that it captures distributed rather than local events – though its output is less "actionable".

This raises the question of what an actionable representation of an experiment should ideally look like. Orchestration systems such as profiles in CloudLab or Heat in Chameleon require adhering to a strict machine-readable syntax, often formulating that syntax in a declarative text file, or providing a layer of indirection that allows the user to work in a higher-order language. Furthermore, these systems are transactional, either fulfilling the topology or not, making complex configurations difficult to develop and iterate upon. Proposed workflow systems [37] experience similar challenges [38]. They fundamental problem from the user perspective in these cases is that a user must invest extra time to make an experiment reproducible. This leads to the "reproducibility dilemma": the user needs to choose whether to invest time into making an experiment replicable or continuing with other research. Ideally, a system that represents an experiment would allow the experimenter to develop it gradually and interactively reflecting the often meandering creative process, support experimental "story-telling" for human as opposed to just machine users, and—true to our philosophy—be an open source project in mainstream use.

In researching solutions to this problem, we considered computational notebooks such as Wolfram Mathematica [39] and Jupyter Notebooks [40], which combine expository text, executable code, and presentation of results in one human-readable, interactive document. Jupyter's newfound ubiquity across the research landscape and its extensible architecture provided fertile ground for exploration; to leverage it, we integrated Jupyter and Chameleon with the intent to reduce the gap between designing an experiment and sharing it. As a result, users can log in to Chameleon Jupyter server with their testbed credentials; these credentials are implicitly bound to the user's isolated Jupyter Notebook server, allowing the user to call Chameleon APIs (via the CLI interface or Python APIs) directly within their notebook's code blocks. This allows a notebook to completely re-create a pre-existing testbed topology. We additionally mounted Chameleon's object store as a virtual drive in the Jupyter application to allow shared storage and therefore collaboration between users. The user's Jupyter server comes pre-installed with several libraries that aid interfacing with an experiment on Chameleon [41] (e.g., creating a lease and launching an instance and then executing commands remotely via SSH). As we've improved the Jupyter interface to Chameleon it has seen an increasing share of testbed usage; over the last year 10% of our monthly active users have been using Jupyter as their interface to the system.

While the notebook may be an appropriate way to package an experiment without disrupting the flow of research, the challenge of properly disseminating and discovering these artifacts remains. At the end of 2019, we implemented the first version of a Sharing Portal [42] to allow users to publish and discover these notebook-based artifacts. Users can publish a set of files directly from the Jupyter interface via a custom UI extension; the files are compacted into an archive and published to CERN's Zenodo [43] for long-term storage, where they are also assigned a DOI for citation. The Sharing Portal then maintains a reference to the published artifact along with helpful metadata such as tags and documentation. Other Chameleon users can search for artifacts within the portal and "re-play" them on Chameleon with one click: an ephemeral Jupyter server tied to the artifact is dynamically provisioned, including additional software dependencies defined by the publisher. Users can version their artifacts by publishing a new set of files and creating a new version (and DOI) on Zenodo. Though too early in its lifetime to provide a quantitative analysis of its impact, we expect to continue investing in this area going forward.

6 Related work

Some of our design decisions in Chameleon were informed by our earlier work on experimental testbeds including FutureGrid [44, 45] and ExoGENI [8], as well as our long standing close collaboration with Grid'5000 [1]. In particular, we are indebted to the latter two projects for not only providing insight but also specific capabilities that were directly integrated into the testbed; the stitching capabilities in the case of ExoGENI and the resource representation and related tooling in the case of Grid'5000. At the same time, Chameleon represents a significantly different approach from any of them in many ways, most prominently in that it is configured for cloud computing research (unlike ExoGENI), supports bare metal reconfiguration (unlike FutureGrid), and is based on a mainstream infrastructure (unlike Grid'5000).

We additionally leverage the experience gained by the rich history of CS testbeds, ranging from those specializing in networking (PlanetLab [46], GENI [4], Emulab [2], OneLab [47], CENI [48]) and wireless/IoT (ORBIT [49], FIT [50], CityLab [51]) to systems (CloudLab [3]) and security (DeterLab [52]); Chameleon complements these systems in that it largely focuses on a different area of research and thus supports different types of experiments. The most similar testbed to Chameleon is CloudLab (itself another NSFFutureCloud testbed). We differ from CloudLab primarily in specific design decisions we made in building the system, many of which

are described in this paper as well as in our early emphasis on reproducibility and sharing by integration of tools like Experiment Précis and Jupyter notebooks. The most significant difference however is that we built Chameleon on top of a mainstream cloud infrastructure for reasons described above.

Clouds are being increasingly used in science and many of them elect to use OpenStack, e.g., Jetstream [53], Aristotle [54], Comet [55], Bridges [56], and NeCTAR [22] all represent different configurations of the system. The most significant difference from Chameleon is that these are relatively standard cloud configurations, designed primarily to support domain science applications rather than CS experimentation, and differ from Chameleon significantly on key features such as bare metal reconfiguration. However, there are significant commonalities on the operations side ensuring that we are able to leverage their contributions, with the OpenStack SIG being one avenue of communication. An interesting recent addition is the CloudBank project [57], which will provide tools, training, and credits for CS research on virtualized commercial clouds; since we provide similar services via our KVM cloud, we look forward to working with this project.

Reproducibility of CS experiments [58] is another area in which our contributions relate to other work in the field. Several projects have used Jupyter notebooks as a mechanism for encapsulating and reproducing research [59, 60]. Managed "Notebook-as-a-Service" platforms e.g., CodeOcean [61], WholeTale [62], and Nextjournal [63] have further elevated the profile and utility of Jupyter for this purpose. Workflow solutions such as Popper [37] aim to aid reproducibility in a different way and are more relevant to our efforts on Experiment Précis. Chameleon differs from all of these examples in that it integrates reproducibility tools in the context of a testbed, allowing users to leverage a commonality of platform to replicate not only the process of experimentation, but also the requisite hardware configurations and topologies.

7 Conclusions

Chameleon represents a unique testbed in that it expresses the capabilities needed for CS research in terms of mainstream cloud functionality. This is an important step to understand how such capabilities may be supported more ubiquitously, with more discernment, and in a more cost-effective manner. This paper discusses the specific design decisions, extensions, and configurations that we chose in order to do so, and evaluates them within a framework that seeks to establish how they influenced the set of supported experiments and how they influenced the community of users the testbed was able to support. Our contribution is accompanied by software that was contributed or integrated with a mainstream open-source cloud implementation (OpenStack) so that a cloud of this type – or its evolutions or variations – can be supported by anyone.

The most important part of our experience is the insight that though originally created to support the most experiments for the most experimenters, testbeds have also become both a generator and an essential platform for sharing digital representations of experiments. While our understanding of digital sharing ecosystem still evolves, and is likely to evolve for some time, we proposed some approaches that our user community has found useful and we look forward to contributing to this area in the future.

Acknowledgments

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation. This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.

Availability

Traces from Chameleon CHI and KVM have been publically available at [64, 65] for the last couple of years and used in a variety of resource management publications. Not all the data used in this paper is reflected in those traces though we are currently discussing revisions to both the content and format. All the code described here is open source.

References

[1] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec. Adding Virtualization Capabilities to the Grid'5000 Testbed. In I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, editors, *Cloud Computing and Services Science*, volume 367 of *Communications in Computer and Information Science*, pages 3–20. Springer International Publishing, 2013.

[2] 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. *ACM SIGOPS Operating Systems Review*, 36(SI):255–270, 2002.

[3] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, et al. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 1–14, 2019.

[4] M. Berman, J.S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. GENI: A Federated Testbed for Innovative Network

Experiments. *Computer Networks*, 61:5 – 23, 2014. Special issue on Future Internet Testbeds – Part I.

[5] Chameleon Resource Discovery. https://www.chameleoncloud.org/hardware/.

[6] M. Cevik, P. Ruth, K. Keahey, and P. Riteau. Wide-area Software Defined Networking Experiments using Chameleon. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 811–816. IEEE, 2019.

[7] OpenStack. https://www.openstack.org/.

[8] I. Baldin, J.S. Chase, Y. Xin, A. Mandal, P. Ruth, C. Castillo, V. Orlikowski, C. Heermann, and J. Mills. ExoGENI: A Multi-Domain Infrastructure-as-a-Service Testbed. In McGeer et al. [4], pages 279–315. Special issue on Future Internet Testbeds – Part I.

[9] K. Keahey, P. Riteau, D. Stanzione, T. Cockerill, J. Mambretti, P. Rad, and P. Ruth. Chameleon: a Scalable Production Testbed for Computer Science Research. In Jeffrey Vetter, editor, *Contemporary High Performance Computing: From Petascale toward Exascale*, volume 3 of *Chapman & Hall/CRC Computational Science*, chapter 5, pages 123–148. CRC Press, Boca Raton, FL, 1 edition, May 2019.

[10] 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)*, pages 409–425, 2018.

[11] Using New vCPU-based On-demand Instance Limits with Amazon EC2. https://aws.amazon.com/blogs/compute/preview-vcpu-based-instance-limits/.

[12] K. Keahey, P. Riteau, J. Anderson, and Z. Zhen. Managing Allocatable Resources. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 41–49, July 2019.

[13] C.P. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, M.S. Lam, M. Rosenblum, et al. Virtual Appliances for Deploying and Maintaining Software. In *LISA*, volume 3, pages 181–194, 2003.

[14] OpenStack Heat. https://docs.openstack.org/heat/latest.

[15] K. Keahey and T. Freeman. Contextualization: Providing one-click virtual clusters. In *2008 IEEE Fourth International Conference on eScience*, pages 301–308. IEEE, 2008.

[16] Chameleon Automated Deployment. https://chameleoncloud.readthedocs.io/en/latest/technical/complex.html#automated-deployment.

[17] Prometheus. https://prometheus.io/.

[18] K. Keahey, J. Anderson, P. Ruth, J. Colleran, C. Hammock, J. Stubbs, and Z. Zhen. Operational Lessons from Chameleon. In *Proceedings of the Humans in the Loop: Enabling and Facilitating Research on Cloud Computing*, pages 1–7. 2019.

[19] OpenStack Ironic. https://docs.openstack.org/ironic/latest.

[20] The 2018 OpenStack User Survey Report. https://www.openstack.org/user-survey/2018-user-survey-report/.

[21] OpenStack Operator Spotlight: CERN. https://superuser.openstack.org/articles/openstack-operator-spotlight-cern/.

[22] NeCTAR Cloud. https://nectar.org.au/research-cloud/.

[23] NASA's JPL powers planetary exploration with Red Hat OpenStack platform. https://www.redhat.com/en/about/press-releases/nasa%E2%80%99s-jet-propulsion-laboratory-powers-planetary-exploration-red-hat-openstack-platform.

[24] OpenStack Scientific SIG. https://wiki.openstack.org/wiki/Scientific_SIG.

[25] OpenStack Scientific Working Group Launches at OpenStack Summit Austin. https://superuser.openstack.org/articles/openstack-scientific-working-group-launches-at-openstack-summit-austin/.

[26] Mirantis Partners with OpenStack Foundation to Support Upgraded COA Exam. https://www.globenewswire.com/news-release/2019/10/17/1931470/0/en/Mirantis-Partners-With-OpenStack-Foundation-to-Support-Upgraded-COA-Exam.html.

[27] Net Promoter Score. https://www.netpromoter.com/know/.

[28] Stackalytics: Total Commits. https://www.stackalytics.com/?metric=commits&release=all.

[29] Kolla. https://docs.openstack.org/kolla/latest/.

[30] Kolla-ansible. https://docs.openstack.org/kolla-ansible/latest/.

[31] Stackalytics: Total Reviews. https://www.stackalytics.com/?metric=marks&release=all.

[32] Openstack-discuss Mailing List Archives. http://lists.openstack.org/pipermail/openstack-discuss/.

[33] Ask OpenStack. https://ask.openstack.org.

[34] CHI-in-a-Box. https://github.com/ChameleonCloud/chi-in-a-box.

[35] S. Wang, Z. Zhen, J. Anderson, and K. Keahey. Reproducibility as Side Effect. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'18 Poster)*. IEEE Press, 2018.

[36] OpenStack Flame. https://opendev.org/x/flame.

[37] I. Jimenez, M. Sevilla, N. Watkins, C. Maltzahn, J. Lofstead, K. Mohror, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. The Popper Convention: Making Reproducible Systems Evaluation Practical. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1561–1570. IEEE, 2017.

[38] M.A. Sevilla and C. Maltzahn. Popper Pitfalls: Experiences Following a Reproducibility Convention. In *Proceedings of the First International Workshop on Practical Reproducible Evaluation of Computer Systems*, page 4. ACM, 2018.

[39] S. Wolfram. The Mathematica Book. *Assembly Automation*, 1999.

[40] Project Jupyter. https://jupyter.org/.

[41] Python-chi: Chameleon Python library. https://github.com/chameleoncloud/python-chi.

[42] M. King, J. Anderson, and K. Keahey. Sharing and Replicability of Notebook-Based Research on Open Testbeds. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC'19 Poster). IEEE Press*, 2019.

[43] Zenodo. https://zenodo.org.

[44] G.C. Fox, G. von Laszewski, J. Diaz, K. Keahey, J. Fortes, R. Figueiredo, S. Smallen, W. Smith, and A. Grimshaw. Futuregrid: a Reconfigurable Testbed for Cloud, HPC, and Grid Computing. In *Contemporary High Performance Computing*, pages 603–635. Chapman and Hall/CRC, 2017.

[45] G. von Laszewski, G.C. Fox, F. Wang, A.J. Younge, A. Kulshrestha, G.G. Pike, W. Smith, J Vöckler, R.J. Figueiredo, J. Fortes, et al. Design of the Futuregrid Experiment Management Framework. In *2010 Gateway Computing Environments Workshop (GCE)*, pages 1–10. IEEE, 2010.

[46] L. Peterson, A. Bavier, M. E Fiuczynski, and S. Muir. Experiences Building Planetlab. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 351–366, 2006.

[47] S. Fdida, T. Friedman, and T. Parmentelat. OneLab: An Open Federated Facility for Experimentally Driven Future Internet Research. In *New Network Architectures*, pages 141–152. Springer, 2010.

[48] Y. Xu, X. Lu, and Y. Zhang. The Development of China's Next Generation Network and National Service Testbed. *development*, 14:3, 2015.

[49] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-generation Wireless Network Protocols. In *IEEE Wireless Communications and Networking Conference, 2005*, volume 3, pages 1664–1669. IEEE, 2005.

[50] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, et al. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 459–464. IEEE, 2015.

[51] J. Struye, B. Braem, S. Latré, and J. Marquez-Barja. CityLab: A Flexible Large-scale Multi-technology Wireless Smartcity Testbed. In *Proceedings of the 27th European Conference on Networks and Communications (EUCNC), 18-21 June 2018, Ljubljana, Slovenia*, pages 374–375, 2018.

[52] J. Mirkovic and T. Benzel. Teaching Cybersecurity with DeterLab. *IEEE Security & Privacy*, 10(1):73–76, 2012.

[53] C.A. Stewart, T.M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, S. Tuecke, G. Turner, et al. Jetstream: a Self-provisioned, Scalable Science and Engineering Cloud Environment. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, page 29. ACM, 2015.

[54] R. Knepper, S. Mehringer, A. Brazier, B. Barker, and R. Reynolds. Red Cloud and Aristotle: Campus Clouds and Federations. In *Proceedings of the Humans in the Loop: Enabling and Facilitating Research on Cloud Computing*, pages 1–6. 2019.

[55] H. Kim and M. Parashar. CometCloud: An Autonomic Cloud Engine. *Cloud Computing: Principles and Paradigms*, pages 275–297, 2011.

[56] S. Bridges. Cancer Genomics Cloud.(June 2017). *Retrieved June*, 1:2017, 2017.

[57] CloudBank. https://www.cloudbank.org/.

[58] D.G. Feitelson. From Repeatability to Reproducibility and Corroboration. *ACM SIGOPS Operating Systems Review*, 49(1):3–11, 2015.

[59] S.R. Piccolo and M.B. Frampton. Tools and Techniques for Computational Reproducibility. *GigaScience*, 5(1):30, 2016.

[60] T. Kluyver, B. Ragan-Kelley, F. Pérez, B.E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J.B. Hamrick, J. Grout, S. Corlay, et al. Jupyter Notebooks - a Publishing Format for Reproducible Computational Workflows. In *ELPUB*, pages 87–90, 2016.

[61] Code Ocean. https://codeocean.com/.

[62] A. Brinckman, K. Chard, N. Gaffney, M. Hategan, M.B. Jones, K. Kowalik, S. Kulasekaran, B. Ludäscher, B.D. Mecum, J. Nabrzyski, et al. Computing environments for reproducibility: Capturing the "Whole Tale". *Future Generation Computer Systems*, 94:854–867, 2019.

[63] Nextjournal. https://nextjournal.com.

[64] Chameleon CHI Traces, November 2019. https://zenodo.org/record/3709794#.XsgpCRNKiL8.

[65] Chameleon KVM Traces, November 2019. https://zenodo.org/record/3709958#.XsgoqRNKiL8.