Arbiter: Dynamically Limiting Resource Consumption on Login Nodes

Dylan Gardner
dylan.gardner@utah.edu
Center for High Performance Computing
University of Utah
Salt Lake City, Utah

Robben Migacz
robben.migacz@utah.edu
Center for High Performance Computing
University of Utah
Salt Lake City, Utah

Brian Haymore
brian.haymore@utah.edu
Center for High Performance Computing
University of Utah
Salt Lake City, Utah

ABSTRACT

In a high-performance computing (HPC) environment, shared resources are often the most capricious and unreliable. The performance of each process is affected by the behavior of all connected users. This applies particularly to the login nodes of HPC clusters, which are used by multiple people at a time. Often, policies govern their use, typically limiting users to small, lightweight tasks such as scripting, compiling, submitting batch jobs, and staging data. If a user is behaving poorly, such as using a significant portion of CPU time or consuming a large fraction of the total memory, processes using the same resources can be slowed or run out of memory. Policies on login nodes that prohibit such behavior, however, may not be enforced automatically because of technical limitations.

Arbiter is a service that overcomes such limitations by using cgroups, a feature of the Linux kernel, to monitor and limit usage. It can both enforce default limits—to ensure the server remains responsive—and penalize users who are consuming resources immoderately while still allowing for brief testing of workloads better suited to computational resources. Arbiter tracks the total memory and CPU usage of a user, reduces the quantity available for a period of time if usage is excessive, and sends emails to inform users of policies and potentially impactful behaviors. Throttling performed by Arbiter encourages users to use computational resources for intensive tasks (through the batch system), thereby supporting the reliability and responsiveness of login nodes.

KEYWORDS

cgroups, systemd, monitoring, throttling

ACM Reference Format:
Dylan Gardner, Robben Migacz, and Brian Haymore. 2019. Arbiter: Dynamically Limiting Resource Consumption on Login Nodes. In *PEARC '19: Practice and Experience in Advanced Research Computing, July 28–August 1, 2019, Chicago, IL.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3332186.3333043

PEARC '19, July 28–August 1, 2019, Chicago, IL

ACM ISBN 78-1-4503-7227-5/19/07…$15.00
https://doi.org/10.1145/3332186.3333043

1 INTRODUCTION

Login nodes are the gateways to computational resources at the Center for High Performance Computing (CHPC), among other research computing sites, and provide an environment for lightweight tasks such as analyzing data, transferring small sets of data, and submitting jobs to a scheduler. Such nodes are a limited resource and abuse, often by new users who are not aware of policies for node usage or users who are trying to circumvent the scheduler queue, can cause performance degradation for other users. Excessive memory or CPU usage can significantly reduce the responsiveness of login nodes and interrupt the workflows of others.

System administrators have several tools available to limit the impact an individual user can have on a server. Control group (cgroup) [6] controllers can be set to limit access to resources. Such controls are present on CHPC login nodes to reserve memory for the system and prevent failure when users deplete the available memory. This alone, however, cannot police usage on the user level; it is still possible for a subset of users on the node to consume memory and CPU cycles in excess, thereby affecting other users connected to the same node.

Arbiter therefore not only limits per-user resource usage with the use of cgroups but also allows for the setting of a threshold or trigger level above which a user is tracked. When users remain above the trigger levels for a determined period of time, they are placed in a "penalty" state. In addition, Arbiter enforces tiered penalty states if a user is repeatedly identified as being in violation of the policies with increased throttling for a longer period of time. In contrast to other existing solutions, these dynamic resource limits have a greater impact on users who repeatedly abuse interactive nodes. This approach is shown in Figure 1, where throttling (right) reduces the system load further than static quotas (center).

When a user violates the administrator-defined policies, the Arbiter service notifies the user and CHPC staff of excessive CPU or memory usage. This facilitates discussions of acceptable usage on login nodes and helps train new users who are unfamiliar with HPC cluster environments and usage policies.

2 BACKGROUND

First released in the Linux kernel 3.6, cgroups is a feature of the Linux kernel that allows groups of processes to be monitored and limited. Resource controllers allow administrators to control and monitor the specific resources of a cgroup, such as CPU or memory.

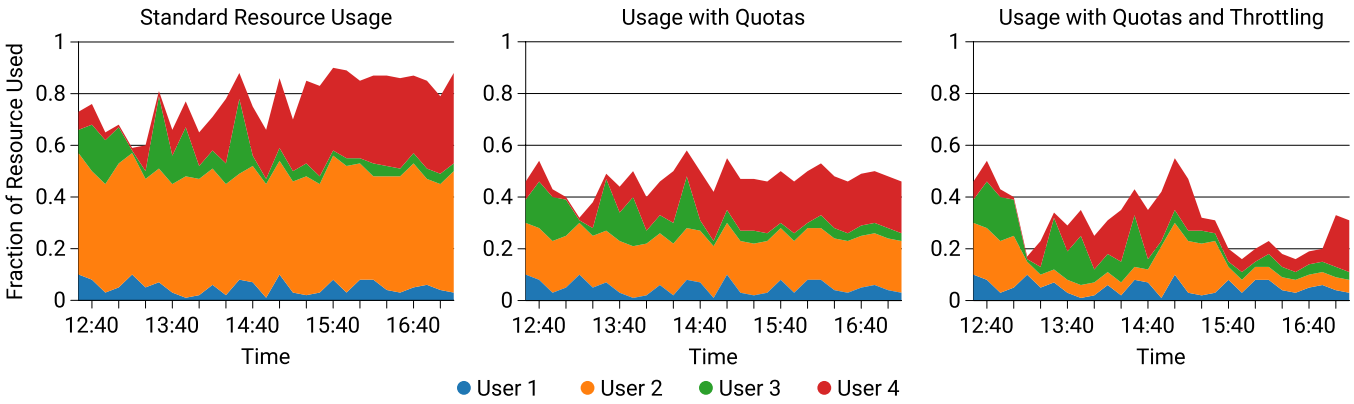

Figure 1: An example of resource consumption on a node with multiple users. The left plot shows usage without quotas or throttling: users are not limited. The middle plot shows the same usage scenario with default cgroup quotas applied, preventing one user from monopolizing the resource. The plot at right shows both default quotas and throttling for excessive usage, further reducing overall node utilization.

2.1 systemd cgroup Hierarchy

Since systemd v205, systemd has integrated cgroups into its notion of a unit (slices, services, and scopes) in a custom hierarchy at /sys/fs/cgroup. Units (such as services in system.service) in system.service can be limited and resource consumption accounted for. A series of slices called user-$UID.slice (this is a child of user.slice, which applies to all users) contain the processes of specific users and are automatically created when users log in and destroyed when they log out with no remaining sessions. This configuration makes it trivial to limit and track a user's behavior. By default, the slices do not have resource accounting enabled in controllers like CPU and memory, but these can be turned on for a specific user with `systemctl set-property user-$UID.slice CPUAccounting=true MemoryAccounting=true`. Turning accounting on for one cgroup also implicitly turns accounting on for parent and sibling cgroups.

2.2 Previous Work

Several HPC sites use cgroups to implement limits on user behavior on shared resources. Such limits are often static and are intended only to limit the impact individual users can have on a node; few solutions implement user accounting or dynamic resource quotas.

One example is Maves and John's *cgroups_py* [5], which is used to apply limits on the resources of a given user with the systemd user-$UID.slice. The service enforces limits when the user is consuming excessive CPU time but is based on a simple usage threshold. The limits are static: users receive the same quotas regardless of individual usage patterns. The maximum CPU time available to each user is limited based on the number of users exceeding a minimum fraction of CPU time. This solution reduces the resources available to all users on the server without considering individual impact or past behavior. In addition, the program enforces limits on the fraction of system memory an individual can use, but this limit is set statically and cannot be set after quotas are initialized.

A predecessor to the Arbiter service has been running on login nodes at CHPC for several years. This first version identified excessive usage on login nodes and notified involved users and system administrators. The service collected information about users' processes and sent emails when policy violations were detected. It did not, however, throttle resources and enforcement of policies required intervention from system administrators. The messages sent by the service identified processes that may have a significant impact, but this information was collected instantaneously as the message was sent and did not show behavioral patterns over time.

Existing solutions for monitoring nodes and limiting a user's impact do not provide detailed information about the processes a user was running. Arbiter's continual collection of usage metrics for each user helps justify throttling and allows users to determine which workloads are suitable for login nodes and which should be submitted to computational resources through the scheduler.

3 IMPLEMENTATION

The improved Arbiter sets default limits on new cgroups; collects CPU and memory information, processes the information into an understandable format; evaluates whether each user's usage is violating the usage policy; and, if in violation, takes action on the user by setting lower usage limits and sending emails. Each step is completed on a configurable refresh interval.

3.1 Collecting Information

In order to collect information on cgroups, Arbiter pulls information from the cgroup hierarchy at /sys/fs/cgroup. When a new user logs in, a slice is created under each of the controllers that are enabled for it: /sys/fs/cgroup/cpu/user.slice/user-$UID.slice for CPU and /sys/fs/cgroup/mem/user.slice/user-$UID.slice for memory (in cgroups v1). Information is then pulled from each of the slices' controllers files: cpu.usage_percpu for CPU usage and memory.usage_in_bytes for memory usage.

The information pulled from cgroups cannot be used directly; Arbiter performs calculations to determine a user's impact. For example, the CPU information pulled from the cpu.usage_percpu file is the CPU time the user has used since the creation of his

or her cgroup. The difference in the number of clock ticks over a period of time is used to approximate usage during that same time. The memory information is instantaneous and must be averaged with multiple measurements to achieve an accurate result. Because average memory consumption is used to evaluate user impact, Arbiter collects information more frequently than it evaluates users (averages are calculated over the refresh interval).

To provide users with a clearer picture of what processes contribute to excessive usage, Arbiter collects information on individual processes. Metrics are collected for each process identification number (PID). The list of PIDs is read from cgroup.procs file in the cgroup slice, but the usage of PIDs must be collected from files in the /proc directory. This per-PID usage is not as accurate (it is polled less frequently), so process usage is scaled to the cgroup usage when it is reported. Short-lived processes, which start and end between collection intervals, may not appear in plots and tables, but the overall usage remains accurate because of this mapping process.

3.2 Evaluating Users

Arbiter determines whether a user has violated the policies for usage on the node by evaluating the metrics it collects on each controller. To evaluate metrics, Arbiter calculates a *badness score*, a representation of how close the user is to a policy violation, where a maximum score of 100 is a violation and a minimum score of 0 indicates that the user's usage has not exceeded the threshold on acceptable CPU and memory usage. The change in the badness score is calculated at each refresh interval and added to the previous badness score. This summation over intervals is used so the rate of change in the badness score is proportional to resource usage (unacceptably high usage would yield a higher per-interval badness score, leading to penalties more rapidly than lower but still unacceptable usage). The change to the badness score B for a given resource r in a given time interval is

$$\Delta B_r = \begin{cases} \eta \frac{u_r}{Q_r} & u_r \geq C_r \\ -\mu\left(1 - \frac{u_r}{C_r}\right) & u_r < C_r, \end{cases}$$

where η is the maximum increase and μ the maximum decrease in badness score for a given unit of time, u_r is the fraction of the resource being used by all processes, Q_r is the resource quota, and C_r is the threshold below which the usage is not considered to be impactful.

Where $u_r \geq C_r$ (usage is above acceptable usage), the badness score increases with time; when $u_r < C_r$, it decreases, albeit more slowly. The net effect is that the user can run programs with resource consumption above his or her thresholds for a short time without being penalized. The constants η and μ are determined by setting a time for which users can run programs consuming all of the resources available to them; η is the ratio of Arbiter's refresh rate (how frequently the program is run) to the time required to be penalized at maximum capacity and μ is an arbitrary value (such that $\eta > \mu$). The algorithm for determining a user's badness score is depicted for an example of resource usage in Figure 2. Resource consumption above the threshold increases the badness score quickly; below the threshold, the score decreases slowly. The badness score, crucially, decreases even when the user is not connected to the node; this situation is not depicted on the plot.

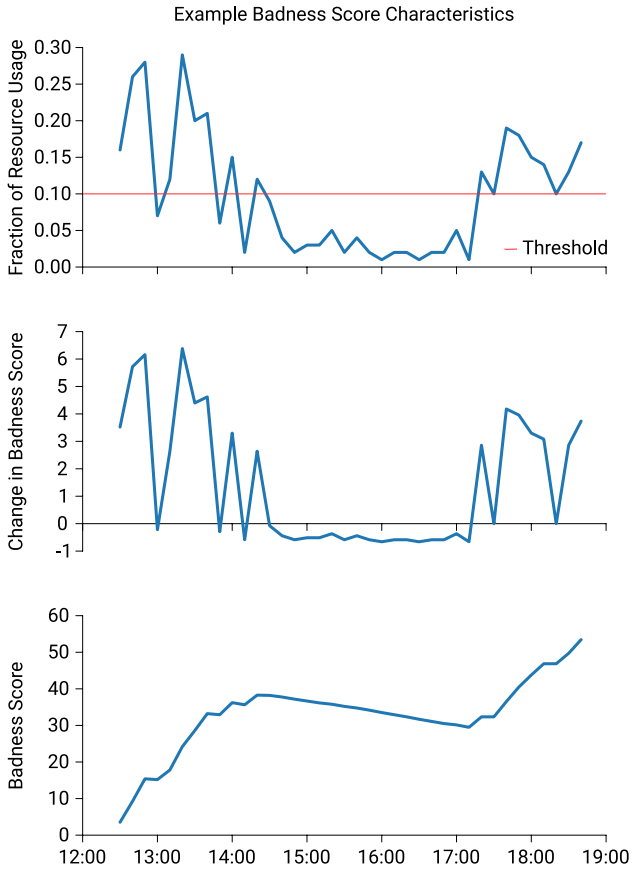

Figure 2: An example of the badness score resulting from user behavior on the node. Only one resource is shown. The change in the user's score is determined by distance from the threshold.

Some programs' CPU usage is not considered when evaluating a user's impact on the node (based on a whitelist of programs). Programs typically encouraged on login nodes, such as compilers and data transfer tools, are ignored when calculating impact on CPU time to avoid penalizing users for legitimate usage on the node. Memory, however, is considered for all processes, as it is a finite resource and cannot be throttled like CPU time. By default, Arbiter does not operate on users with a user ID below a threshold. In addition, other users can be made exempt from penalties. This prevents system accounts from being throttled.

3.3 Tracking a User's Status

To keep track of what limits should be applied to a user, Arbiter has several *statuses*. Statuses contain default resource limits, a threshold defining unacceptable usage, and other attributes like a status-specific process whitelist. The statuses are defined in the configuration file. A user has only one status at a time (their *current status*). He or she also has a *default status*, which is used as a reference for

resource limits when returning from a penalty state. Initially, before Arbiter has taken any actions or set any limits, a user's current status is their default status. This default status is determined by matching a user's user ID or group ID with statuses defined by administrators. If no status contains the attributes, the user's status is set to a default. Statuses allow for the categorization of users (CHPC, for example, has a *normal* status for most users, an *admin* status with an extended whitelist for administrators, and an *invincible* status that allows for unlimited usage). Arbiter uses statuses to define limits when users are penalized; throttling is based on the default status of a user and a fraction of resources for each penalty state.

3.4 Applying Limits

When a user's current status changes, the limits associated with the new status are applied to the user's cgroup slice. Limits on CPU are set by writing a quota on the CPU time available to a user to the cpu.cfs_quota_us file in the CPU hierarchy. Users may utilize the CPUs on the server until this limit is reached; beyond this point, the resources will be unavailable for a short time. After this time, the user can again use CPU time in accordance with his or her default status. In effect, the processes are slowed by running intermittently. Arbiter does not attempt to restrict users' processes to specific cores. Brief testing across all the cores on a server is possible, which allows users to evaluate message-passing algorithms as they would work on computational servers.

The memory allocated to a given user is limited by setting a limit (in bytes) directly to the memory.limit_in_bytes file in the memory hierarchy. The memory limit cannot, however, be set when the current memory usage exceeds the quota being set. If a user surpasses his or her memory quota once they have been successfully set, the kernel will attempt to reclaim memory; if this is not possible, an out-of-memory routine (called the *out-of-memory killer*) "is invoked to select and kill the bulkiest task in the cgroup" [2].

Arbiter makes an effort to limit a user's memory to the minimum possible value. If a memory quota cannot be set, the limit itself is incrementally increased and the new, higher value is applied instead. Arbiter checks users' memory quotas against the expected value each time it runs and sets the quotas again if there are discrepancies. If processes relinquish memory (if they run out of memory and are killed, for instance), the quota can be set nearer the user's limit. While quotas on memory usage might not be possible when a user is first throttled, the process of continually checking and updating the cgroup limits ensures the quota will be as close to the expected value as possible.

3.5 Default Limits

To prevent users from suddenly abusing the login nodes and affecting the work of others, Arbiter sets default limits on each user. This prevents an individual user from using excessive CPU and memory.

There are two ways that administrators can apply such default limits. A script can be integrated into the Linux Pluggable Authentication Modules (PAM) stack to set a fixed default limit when a user logs in or, as at CHPC, Arbiter can automatically detect new users every refresh interval and apply default limits then (using the `--status-on-creation` flag). The latter requires fewer changes to the system and allows for dynamic default limits based on a user ID or group ID, making it possible to set higher or lower default limits for certain users, at the administrator's discretion. The downside of the latter approach is that users can potentially abuse resources for a short time when they first log in, before Arbiter runs and the default limits are applied. In practice, this is unlikely: it generally takes time for resource consumption to increase and become a nuisance.

3.6 Tiered Penalty Statuses

When Arbiter detects that a user has violated the established usage policies, it applies a penalty status. There are different levels of penalty statuses, but they all penalize the user for a set period of time with lower quotas on each resource. The penalties are applied in tiers. When a user is penalized repeatedly, the length of time he or she spends in a penalty status is increased and the resources available are further decreased. Ideally, this throttling with increasing penalties should discourage users from running computationally-intensive tasks for an extended time and is intended to be only a minor inconvenience for users with legitimate interactive work suited to login nodes. It should, however, lessen the impact of large tasks that are left running long enough to warrant increased throttling.

Tiered penalties are enforced by tracking the number of times a user has been penalized: each time a penalty status is applied, this count increases. It is decreased automatically with time. As a result, users who infrequently exceed the thresholds will not be throttled to the same extent as users who repeatedly do so in a short time.

3.7 System Requirements and Setup

Arbiter is written for CentOS 7 systems deployed at CHPC. It should be operable on Linux kernel versions beyond 3.10, provided cgroups v1 is used. More recent versions of systemd use a *hybrid mode* integrating cgroups v1 and v2 (though the mode will be superseded by the *unified mode* of cgroups v2) [**?**]. Arbiter can be deployed on systems with a hybrid hierarchy (by setting `memsw = false` in the configuration files), though some functionality may not be available. The Arbiter source and installation instructions are found in Appendix A.

Arbiter is designed to run as a systemd service. This makes it easy to restart and manage. It also means the service has its own cgroup, separate from a user slice (so it cannot limit itself). Output is logged to a configurable directory. The verbosity of the output when `--print` (`-p`) is used can be controlled with the `--verbose` (`-v`) and `--quiet` (`-q`) flags.

The Arbiter service has several prerequisites. It requires Python 3.6 or newer. In addition, Python packages like `matplotlib` and `toml` need to be installed. Next, a log directory must be created and the configuration files must be updated to reflect this location. Finally, modifications to the sudoers file are required if Arbiter is not run as root. Flags to modify the service's behavior, including the option to use `sudo`, are shown in Table 1.

Configuration files can be made independent and distinct for each node. Limits can be adjusted to accommodate different usage types on different resources. The configuration files contain options to set Arbiter's refresh and collection interval, the amount of information

Flag	Description
`--rhel7-compat`	Runs Arbiter with RHEL 7/CentOS 7 compatibility.
`--status-on-creation`	Applies the statuses when Arbiter first detects new users.
`--accounting`	Turns on accounting for all users by turning it on for a specific, persistent user.
`--etc`	Sets the directory of configurable modules.
`--config`	Sets the configuration files to use. Configurations cascade if multiple files are specified.
`--sudo`	Uses sudoers permissions to write out limits and to enable cgroup accounting.
`--print`	Prints the Arbiter service's logging to stdout.
`--verbose`	Turns on more verbose output.
`--quiet`	Outputs only critical information.
`--exit-file`	Tells Arbiter to stop itself (and restart, if enabled in the service file) if a file is updated.

Table 1: The flags that can be used to change the behavior of Arbiter.

kept in memory, the limits on specific users, and email and log settings. They are extensible and can be used to apply Arbiter at sites other than CHPC and on a variety of resources.

3.8 Permissions and Security

Arbiter requires access to files in the cgroup hierarchy. The number of files required, however, is minimal; permissions are granted only where necessary.

Arbiter writes limits to the files in the cgroup hierarchy; this is generally faster than running shell commands from within Python. It also requires, however, that the user running the service has permission to modify the files (which are otherwise owned by root). To accomplish this, the group associated with cgroup files is set to a group of which the user running Arbiter is a member. Commands to set the permissions on such files are defined in a custom *sudoers* file. This file can be generated using a script to minimize the burden on system administrators. When a new user is detected, the file permissions are updated from the command line. This must occur each time a new user connects to the node but is not necessary for new connections by users already on the node (who are present in the cgroup hierarchy).

4 NOTIFYING AND EDUCATING USERS

The information collected by Arbiter while evaluating usage on a node is included in reports sent to users and administrators when users' statuses are updated. The historical information provided by Arbiter is a representation of a user's impact on a machine. The resource consumption is shown in both a table and plots; in previous versions of the script, only instantaneous usage metrics were available. Arbiter now provides historical context for its throttling and other actions. An example of the plots generated by Arbiter is shown in Figure 3; it contains a recent history of processes and the threshold for processes to be considered impactful. Messages from Arbiter also include descriptions of policy and expectations on login nodes.

When throttling is removed (a user returns to a default status), a message is sent to notify the user and system administrators. This message includes a description of the expectations on login nodes and provides references to policy information.

In addition to evaluating individual users' impacts, Arbiter also periodically checks the overall utilization of the nodes on which it is deployed. If the fraction of total CPU or memory usage is above thresholds set by the administrators, the service notifies CHPC staff and lists connected users in order of resource usage. This system helps administrators take action to prevent the aggregated activity of users from becoming excessive and rendering the node unresponsive.

5 LIMITATIONS AND POTENTIAL IMPROVEMENTS

During the development and testing of Arbiter, some limitations relating to the operating system (CentOS 7 at CHPC) and associated versions of systemd became apparent. One such issue is a bug in systemd (found in v219 and fixed in v240) that requires accounting be enabled each time Arbiter starts because enabling cgroup accounting does not persist on reboot [4]. Another limitation is the limited support of kernel memory in the cgroups memory controller on CentOS 7. This prevents Arbiter from getting accurate data from memory.usage_in_bytes since that file accounts for kernel memory, but the value in the memory.kmem.usage_in_bytes file—normally used to subtract that kernel memory usage—is zero [1]. To resolve this issue, Arbiter replaces cgroup memory data with the sum of a user's collected PID memory data if the `--rhel7-compat` flag is used.

Since the Linux kernel 3.10 and systemd v219, many improvements have been made to systemd cgroups. Upgrading to cgroups v2 and a new version of systemd would not only resolve the issues above but would also allow for new features. For example, systemd v239 adds unit slice drop-ins, allowing default limits of user-$UID.slice to be applied automatically when a slice is created without a PAM integration [3].

Another limitation is that Arbiter evaluates users' impacts on individual nodes with no regard for behavior on other nodes. When a status is updated, the limitations imposed on the user apply only on the node on which the violation of policy was noted. Future

improvements may include adding a shared database that allows user tracking and limiting to apply across multiple nodes simultaneously. A centralized database server could then be used to effect penalties on all login nodes when excessive usage is discovered on one.

One of the more common usage patterns on HPC login nodes is data movement between clusters. This is a legitimate use of the login nodes often consumes considerable CPU, memory, and time writing to disks and network interfaces, which can affect responsiveness. One potential improvement of Arbiter could be to use the input and output or networking cgroup controllers to gain additional insight about usage and further limit individuals' impact on nodes.

6 CONCLUSION

Arbiter, a service that tracks usage on login nodes and automatically limits users' access to resources with cgroups, has recently been deployed on nodes at CHPC. The service collects usage metrics, applies limits automatically, and informs users of potential violations of usage policies.

Arbiter is a new technology and has been introduced in production environments at CHPC only recently; its impact cannot be quantified reliably. However, the service reduces the load each user can place on a given login node and is expected to improve reliability. It throttles users an average of 10.7 times each day across the login nodes at CHPC and prevents individual users from monopolizing resources. The number of overall high usage emails received by CHPC staff is plotted in Figure 4; usage on nodes has decreased since Arbiter started throttling and sending emails to users. Prior to this time, it was running but sending emails only to CHPC staff and not throttling any users. In the short time Arbiter has been deployed, it has identified several users who have been misusing login nodes repeatedly and has facilitated communication because of its graphical and verbose email updates.

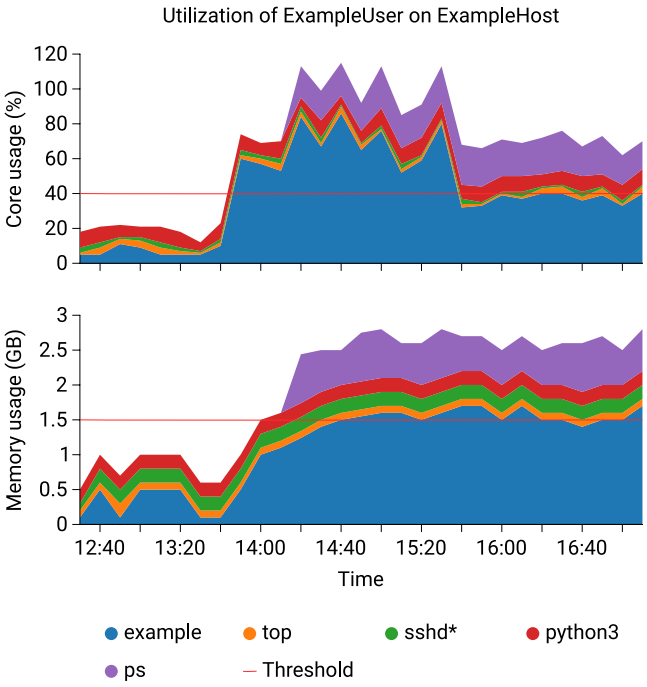

Figure 3: An example of the plots included in an email. The plots show both CPU and memory utilization over time to help users identify problematic behaviors and processes.

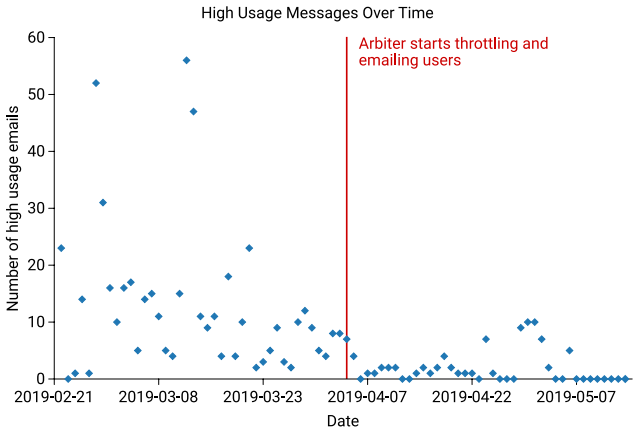

Figure 4: The number of overall high-usage emails received by CHPC staff. Arbiter was enabled and started enforcing limits in early April; at this time, the load on login nodes decreased substantially.

ACKNOWLEDGMENTS

We would like to thank Anita Orendt and the user services team at CHPC for their continued support. We would also like to thank Paul Fischer for his contributions and suggestions and Albert Lund for his work on the predecessor of the Arbiter service.

REFERENCES

[1] [n. d.]. Enabling kmem accounting can break applications on CentOS7. https://github.com/opencontainers/runc/issues/1725.
[2] [n. d.]. Memory Resource Controller. https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt.
[3] [n. d.]. systemd System and Service Manager. https://github.com/systemd/systemd/blob/master/NEWS.
[4] Ryutaroh Matsumoto. 2018. systemd-run --user -t -p MemoryHigh=1G stopped working. https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=903011.
[5] Curtis Maves and Jason St. John. 2018. cgroups_py: Using Linux Control Groups and Systemd to Manage CPU Time and Memory. (2018).
[6] Paul Menage. [n. d.]. cgroups. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.

A ACQUIRING AND INSTALLING

The Arbiter source is open to other sites interested in deploying or extending it. It is available through a Git repository (located at https://gitlab.chpc.utah.edu/arbiter2/arbiter2), which also contains guidelines for configuring and installing the service. Problems during the installation process can be addressed with the log files generated by Arbiter (with `-v` to enable verbose output) or the logs associated with the service (when configured).

A.1 Installation Process

The installation process is bundled with the source. The instructions may vary with different operating systems and environments. In general, the Arbiter service can be installed by

(1) installing a recent version of Python 3 (version 3.6 or higher for the current release of Arbiter);
(2) installing the required Python packages (such as `matplotlib`, `requests`, and `toml`);
(3) configuring a user to run the service, such as root or a user with limited superuser privileges;
(4) verifying the configuration file and any site-specific variables or functions for correctness;
(5) testing the configurations and installations by running the scripts directly; and
(6) setting up a service to run the scripts (this process is facilitated by a shell script).

The installation procedures are described in further detail in the installation guide included with the source. Because Arbiter is still being developed, system administrators should consult the updated installation guide (which may differ from the above instructions) when installing the service.