Computing and Software for Big Science (2021) 5:9
https://doi.org/10.1007/s41781-020-00050-y

ORIGINAL ARTICLE

Operating an HPC/HTC Cluster with Fully Containerized Jobs Using HTCondor, Singularity, CephFS and CVMFS

Oliver Freyermuth[1] · Peter Wienemann[1] · Philip Bechtle[1] · Klaus Desch[1]

Received: 28 May 2020 / Accepted: 16 December 2020 / Published online: 29 March 2021

Abstract
High performance and high throughput computing (HPC/HTC) is challenged by ever increasing demands on the software stacks and more and more diverging requirements by different research communities. This led to a reassessment of the operational concept of HPC/HTC clusters at the Physikalisches Institut at the University of Bonn. As a result, the present HPC/HTC cluster (named BAF2) introduced various conceptual changes compared to conventional clusters. All jobs are now run in containers and a container-aware resource management system is used which allowed us to switch to a model without login/head nodes. Furthermore, a modern, feature-rich storage system with powerful interfaces has been deployed. We describe the design considerations, the implemented functionality and the operational experience gained with this new-generation setup which turned out to be very successful and well-accepted by its users.

Keywords Scientific computing · Containers · Distributed file systems · Batch processing · Host management

Introduction

High performance and high throughput computing (HPC/HTC) is an integral part of scientific progress in many areas of science. Researchers require more and more computing and storage resources allowing them to solve ever more complex problems. But just scaling up existing resources is not sufficient to handle the ever increasing amount of data. New tools and technologies keep showing up and users are asking for them. As a result of these continuously emerging new tools, software stacks on which jobs rely become increasingly complex, data management is done with more and more sophisticated tools and the demands of users on HPC/HTC clusters evolve with breathtaking speed [1]. Therefore, the diversity and complexity of services run by HPC/HTC cluster operators has increased significantly over time. To cope with the increased demands, an ongoing trend to consolidate different communities and fulfil their requirements with larger, commonly operated systems is observed [2].

This work describes the commissioning and first operational experience gained with an HPC/HTC cluster at the Physikalisches Institut at the University of Bonn. We call this cluster second generation Bonn Analysis Facility (BAF2) in the following. Occasionally we compare the setup of this cluster with the one of its predecessor (BAF1). Both BAF1 and BAF2 were purchased to perform fundamental research work in all kinds of physics fields ranging from high energy physics (HEP), hadron physics, theoretical particle physics, theoretical condensed matter physics to mathematical physics. BAF1 was a rather conventional cluster whose commissioning started in 2009. It used a TORQUE/Maui-based resource management system [3] whose jobs were run directly on its worker nodes, a Lustre distributed file system [4] without any redundancy (except for RAID 5 disk arrays) for data storage and an OpenAFS file system [5] to distribute software which was later supplemented by CVMFS [6] clients (see "CVMFS" for more information on CVMFS). The latter provides software maintained by CERN and HEP collaborations. BAF1 maintenance was characterized by a large collection of home-brewed shell scripts and other makeshift solutions.

✉ Peter Wienemann
peter.wienemann@uni-bonn.de

Oliver Freyermuth
freyermuth@physik.uni-bonn.de

Philip Bechtle
bechtle@physik.uni-bonn.de

Klaus Desch
desch@physik.uni-bonn.de

[1] Physikalisches Institut, Universität Bonn, Nußallee 12, 53115 Bonn, Germany

In contrast, BAF2 is adapted to the increasingly varying demands of the different communities. These requirements and which solutions we chose to tackle them will be discussed in "BAF2 Requirements" followed by a short overview of the new concepts of this cluster in "Cluster Concept". After this introduction, we will present the key components of the cluster in "CVMFS", "Containerization", "CephFS", "XRootD", "Cluster Management" and "HTCondor" in-depth. The paper will conclude with a presentation of experiences and observations collected in the first two years of operation in "Operational Experience". On purpose, benchmarks of the system are not presented. While we performed some benchmarking of the components before putting the system into operation, we believe that our results cannot be easily transferred one-to-one to setups using different hardware or which operate at different scales. Another aspect which comes into play is that even refined synthetic benchmarks are quite different from the constantly evolving load submitted by users. A realistic simulation of the load caused by a diverse mix of jobs is very difficult, and even after the two years of operations, new use cases and workloads appear on a regular basis.

For this reason, we consider the presentation of experiences with the operational system in its entirety under realistic production workloads of users to be more useful and will present the observed effects in "Operational Experience" in detail before concluding in "Conclusion".

BAF2 Requirements

The requirements on BAF2 are as broad as the range of research fields it serves. The most challenging constraints are imposed by running analysis jobs of the ATLAS high energy physics experiment [7]. This experiment uses a huge software stack which is provided and maintained centrally by a dedicated team on a specific platform (at the time of writing the migration from Scientific Linux 6 to CentOS 7 is still not fully completed). Given the rapid development of software of a collaboration of $\mathcal{O}(3000)$ members, it would be unfeasible for each collaborating institute to maintain and validate its own software installation. Therefore, the software is distributed to all participating institutes via the CVMFS file system [6, 8–10]. We will describe this file system, for which we now also operate our own server infrastructure, in more detail in "CVMFS". The collaboratively maintained software also puts constraints on the platform of computing resources used for ATLAS purposes. At present, the software framework only runs on x86_64 Scientific Linux 6 [11] and CentOS 7 [12] machines (with Scientific Linux 6 slowly dying out). Given the age of Scientific Linux 6 and CentOS 7, there is an increasing tension between ATLAS requirements and other applications which require a more up-to-date software stack. The same is true for support of modern hardware components. Luckily, modern virtualization technology provides an attractive solution to this constraint. We set up the cluster in such a way that each user can choose the operating system (OS) in which her/his jobs should run using modern container technology [13], and the actual bare metal operating system is never exposed to the user. The container setup is described in "Containerization". Thanks to the container awareness of the HTCondor resource management system [14–21] the implementation of such a setup is straightforward. More details on how we use HTCondor are given in "HTCondor".

Another particular requirement for ATLAS data analysis is providing interfaces for the distributed data management (DDM) tools deployed in ATLAS [22, 23]. Most importantly, we are running an XRootD service [24]. This allows automatic transfers and subscription of datasets with high throughput to the BAF2 cluster. More information on XRootD is given in "XRootD". The datasets shipped to the BAF2 cluster are finally analyzed by reading the corresponding data files from a POSIX file system. As POSIX file system, we have chosen CephFS [25] due to the reasons explained in "CephFS".

All mentioned resources are deployed and orchestrated using Foreman [26] and Puppet [27]. We describe this setup in "Cluster Management".

As a general objective, free and open source software (FOSS) tools are used for cluster management and operation wherever possible. Reasons are not only financial ones but also our appreciation of the possibility to easily debug, patch and extend available software. When patching software we always try to feed the modifications back into the respective developer community to avoid accumulating large patch sets over time which could diverge from the upstream project and make future upgrades more difficult.

Cluster Concept

While the previously operated BAF1 cluster was conventionally set up using login nodes and without redundancy of the cluster services, a different approach was chosen for BAF2 due to the increasing diversity of requirements.

The new workflow from the users' perspective is that the jobs are submitted directly from their desktop machines. On that very machines, not only access to kerberized home directories maintained and backed up by the central university computing centre and mounted via NFS v4.2 [28] is provided, but the cluster file system CephFS can also be accessed directly (via NFS v4.2). The users' jobs can then either access CephFS with high bandwidth or use HTCondor's file transfer mechanism to store smaller input and output files in the home directories.

For each job, users can choose the operating system which should be used, and the cluster workload manager HTCondor takes care to instantiate the corresponding container

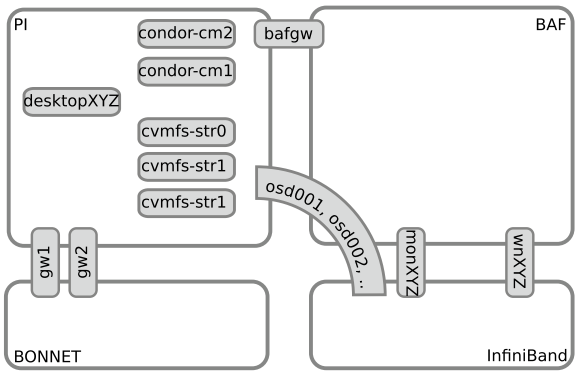


Fig. 1 Schematic overview of the BAF2 network setup. Storage nodes (`osdXYZ`, `monXYZ`) and worker nodes (`wnXYZ`) are connected to an InfiniBand and a cluster-specific Ethernet network (BAF network). To avoid the cluster gateway node (`bafgw`) with 1 Gbit/s network interfaces to become a throughput bottleneck for grid data transfers, the `osdXYZ` nodes are also connected to the public Physikalisches Institut network (PI). The PI network is also the place where additional infrastructure nodes like CVMFS servers and the HTCondor central manager nodes are located. This is also the network from which jobs are submitted (`desktopXYZ`). The PI network in turn is connected via a redundant (2×10 Gbit/s) gateway setup to the campus network (BONNET)

behind the scenes. This means that from the users' point of view, the `ssh` command to a login node is replaced by submission of an interactive job, and a manifold of required environments can be offered without maintaining separate, dedicated login nodes.

Additionally, almost all components of the new setup are designed to provide high availability to ensure continuous service availability, reduce pressure on operators in case of failures and to ease upgrade procedures. This is true for the cluster workload management system HTCondor, for which two separate central manager virtual machines are operated as discussed in "HTCondor", the CernVM file system as illustrated in "CVMFS", the cluster file system CephFS as explained in "CephFS" and also the connectivity to the distributed data management (DDM) system of the experiments via XRootD as detailed in "XRootD". The cluster gateway machine is currently not operated redundantly, but work is ongoing to merge the gateway functionality into the redundant main gateways, keeping the network separation via firewall rules, but gaining redundancy and bandwidth.

As visualized in Fig. 1, the network topology was designed to clearly separate the cluster network from the general purpose network in which the users' desktop machines are located. The ethernet network used within the cluster is set up to use private addresses which are then masqueraded by the cluster gateway while the InfiniBand network operates with private addresses only and has no separate outbound connectivity. This separation was part of the operational security considerations and is mostly invisible to the users, since an interactive cluster job emulates an `ssh` connection to a worker node which HTCondor realizes via the Connection Broker service as explained in "HTCondor".

Using this approach, full flexibility is exposed to the users. The scheduling is based on a fair-share algorithm, so any user can submit as many jobs as needed and may at times use the full cluster resources.

We monitor our complete setup (servers, desktops, printers, environmental parameters like temperatures and humidity, etc.) with Zabbix [29]. It collects values from monitored devices, plots them and notifies operators in case defined threshold values are reached. An intuitive web user interface allows one to easily configure all relevant monitoring settings and to define who is allowed to see which systems/services.

Compared to BAF1 which in its final state comprised of roughly 800 CPU cores and 370 TB of net storage space the hardware upgrade to BAF2 is not as large as the conceptual differences. At the end of 2019, the BAF2 resources comprise of 1120 CPU cores (Intel® Xeon® CPU E5-2680 v4) which add up to 2240 virtual CPUs due to enabled simultaneous multithreading (SMT). Those cores are distributed over 40 compute nodes which are equipped with between 128 GB and 1 TB of RAM. The CPUs are supplemented by one GPU server with four Nvidia® GeForce® GTX 1080 Ti GPUs with 11 GB of memory each. In total, 580 TB of net space is available for data storage. The corresponding storage system comprises in total eleven nodes.

CVMFS

The CernVM File System (CVMFS) has been developed by the European Organization for Nuclear Research (CERN) to distribute centrally maintained software to data centres spread around the globe. It is extensively used by high-energy physics experiments to make their centrally maintained software frameworks available on computing clusters at participating institutions which are run by local operators. Given the development speed and complexity of the software of large collaborations, decentralized maintenance of the experiment software would be too inefficient and error prone.

Since software frameworks are quasi-static contents, CVMFS has been designed as read-only file system. The read-only design makes aggressive multi-tiered caching—both on the provider and on the consumer side—relatively simple. Delivered content can be hosted on standard web servers and HTTP is chosen as transfer protocol. As a result, standard web tools can be used to deliver and cache content and problems with firewall rules are minimized. Even if the content is delivered via untrusted caches and network connections, the authenticity and integrity of the delivered

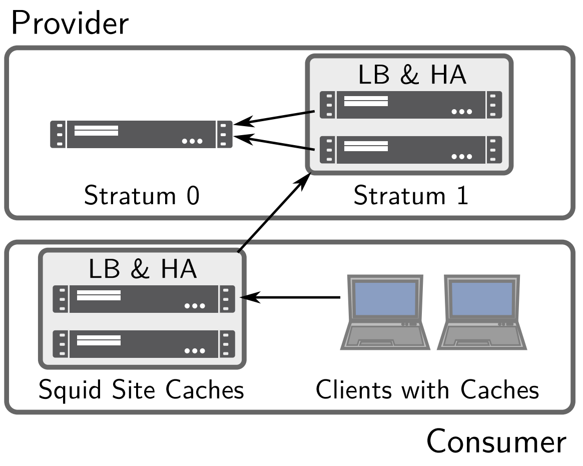


Fig. 2 Schematic overview of the CVMFS data flow. The data originates from a stratum 0 node. This is mirrored on stratum 1 servers for load balancing (LB) and high availability (HA) purposes on the provider side. On the consumer end, the data can be cached by local Squid servers to reduce latency in addition to client side disk-based caches. If desired the Squid servers can also be run in a highly available, load balanced setup

content is ensured using cryptographic hashes. CVMFS also offers transparent file (de-)compression and deduplication which is very valuable e.g. in cases where multiple versions of software packages are stored. Additionally, catalogues are created at directory or directory tree level and are put under version control to always serve a consistent state to the clients.

Technically, CVMFS is implemented as a POSIX compliant file system using the Linux kernel FUSE module. Unlike many distributed file systems, it is very efficient in reading many small files scattered over plenty of directories like it is typical for storage hosting software frameworks.

CVMFS is structured in "repositories". The typical mount point convention for a repository with the name `repository` by an organization with the DNS domain name `example.com` would be the FQRN (fully qualified repository name) `/cvmfs/repository.example.com`.

An example for the data flow in CVMFS is shown in Fig. 2. The provider runs a "Stratum 0" system which allows to temporarily access the read-only repositories in read-write mode by overlaying them with an overlay file system (AUFS [30] or OverlayFS [31]). All data written is chunked, compressed, checksummed and deduplicated when committed and served via a standard HTTP server software. The "Stratum 1" machines are optional mirrors of the full repository content and commonly used to reduce the load on the "Stratum 0" node and to provide a backup, additional load balancing and high availability (HA).

The client could directly access the "Stratum" servers (if public), but for larger sites, it is very common to put Squid caches [32, 33] in between to significantly reduce the latency and provide load balancing (LB) and high availability. Additionally, the clients keep local caches of the catalogues and the accessed data.

While the current design of CVMFS requires clients to poll for changes[1], future plans for CVMFS involve the addition of a message brokering system to inform clients or other subscribers about changes to the repository. On the other end, a repository gateway allows to scale out the commit operation to the "Stratum 0" machine to multiple responsible repository managers, which both distributes the load and will allow to delegate privileges for committing changes.

We employ CVMFS for two purposes at the University of Bonn:

- Software distribution,
- Distribution of extracted container images.

For these two purposes, we run two different repositories with the respective mount points:

- `/cvmfs/software.physik.uni-bonn.de`
- `/cvmfs/container.physik.uni-bonn.de`

In addition, we use nine CVMFS repositories provided by CERN both on end user desktop machines and cluster worker nodes.

The software provided via the local repository extends the CERN repositories and distribution packages. To prevent an unnecessary maintenance burden, we only provide software which matches at least one of the following conditions:

- Only for use within the institute (proprietary or licensed software that allows for storage on a shared file system),
- Software maintained by local developers for local working groups or teaching,
- Software not provided by Linux distributions, either not at all or with an inappropriate version.

Notably, we do not rebuild a full software stack with tools such as Spack [34] or EasyBuild [35] as is often done in HPC centres, since most of the software tools used in the community are provided by the CERN repositories and rebuilding and revalidating all of them with a different software stack is not feasible at the scale of a single data centre.

To ease the selection of software from our local repository for users, we employ Lmod [36], a Lua [37] based system for environment modules. Using a hierarchical structure, a

[1] This is by default performed every 5 min.

single Lua module is written setting up the software-specific environment for each version. With one command, users can list the available software for the container environment they have chosen and load a software version of their choice.

Lmod takes care of manipulating the environment variables accordingly and also allows users to “unload” modules and “reload” another version, cleaning up and adjusting the environment accordingly. This allows us to expose various software packages both on the computing cluster and on the desktop machines without interference with the packages provided by the Linux distribution.

This approach to distribute software has been well accepted by the users, so it now also serves as data source for laptops used within the institute or for teaching purposes. In this case, a fraction of the available repositories is automatically mirrored to the local disks of the devices to guarantee operation without network connectivity. Ongoing developments within CVMFS such as the “Shrinkwrap” tool [38] which allows for partial deduplication in a regular file system by the usage of hardlinks will improve the space efficiency of this approach.

Containerization

Virtualization has revolutionized IT operations during the last 15–20 years. The virtues are isolation of the host and the guest system, easy preservation and recovery of computing environments and to a certain extent, a decoupling of the operations of host and guest systems. With the advent of hypervisors like VMware [39], Xen [40], KVM [41], VirtualBox [42], etc. virtualization started to conquer computing centres. But the high degree of isolation which virtual machines (VMs) offer also comes with a price: A loss of performance due to additional abstraction layers (hardware emulation). The emulation overhead was reduced by the introduction of para-virtualization [43]. Thanks to this approach, it is not necessary anymore to emulate all hardware components. Still, virtualization played a subordinate role in high-performance and high-throughput computing until recently. However, in recent years also, virtualization options with a less strict isolation have become available. They virtualize on the operating system (OS) level making use of kernel namespaces [44] and partially also cgroups [45] to provide so called containers. Well-known representatives of container software are Jails [46], OpenVZ [47], Solaris Containers [48], LXC [49] and Docker [50]. While Jails and Solaris Containers are only available under FreeBSD and Solaris, respectively, OpenVZ was the first container implementation for Linux. But like LXC, it remained a niche product. Containers became really popular with the advent of Docker. These days they are widely used for quick service deployment and have become an integral part of the DevOps tool chain. Despite the fact that containers overcome the performance issues of VMs, they still lacked acceptance in the HPC/HTC community. One of the reasons is the need of running a daemon with root privileges to enable access to isolation and mounting functionalities not exposed to unprivileged userspace. This design entails the danger of privilege escalation vulnerabilities, such as those discovered both in the early days of Docker (see for example [51, 52] and also in recent years (e.g. [53]). As a result, administrators of clusters executing user code were cautious about exposing this functionality. Therefore, containers were mostly used for isolation of services and not for compute jobs. In recent years, kernel development has caught up on these requirements and with the advent of unprivileged user namespaces[2], most features are now accessible on recent operating systems.

Following these developments, recently, new container options have become available, for example Singularity [55], Charliecloud [56, 57], runC [58] and Podman [59]. Some of these are specifically designed for HPC/HTC applications, and they all support unprivileged user namespaces, even though this may lead to a reduction of the feature set. Namely, the main limitation is the mounting of arbitrary file system images, since the involved parts of the kernel have not been designed for exposure to unprivileged users [60]. On the other hand refraining from mounting images also reduces the attack surface as the kernel vulnerability CVE-2016-10208 [61] has shown.

A standardization effort in form of the Open Container Initiative (OCI) [62] focuses on streamlining the interface to configure these container runtime implementations and specifying the layout and format of container images which can be used with user namespaces. This initiative has been started by Docker and is a project of the Linux Foundation [63]. “runC” is the reference implementation of the container runtime for OCI.

At the time of writing, we are offering Debian 10, Scientific Linux 6, CentOS 7, CentOS 8 and Ubuntu 18.04 to our users. Our container images are built locally based on official DockerHub images maintained by the upstream distributions. We have chosen Singularity as container runtime engine, because HTCondor offers an off-the-shelf interface to Singularity. In the long run, we plan to switch to a fully unprivileged OCI-based solution. Singularity is used to pull the existing images from Docker Hub [64], install updates and additional packages and add adaptions for local usage. This includes the creation of directories for bind mounts, setting up the Lmod environment modules system (see “CVMFS”), the installation of base packages required for

[2] Most of the unprivileged user namespace functionality is available since kernel 3.8 [54].

HEP software, a CephFS quota checking tool, extensions to the shell profile for more comfortable interactive usage and GPU support and additional packages requested by local groups.

The recipes to build the containers are stored in a git repository and a rebuild is triggered either when the recipe is changed or a day has passed since the last rebuild. This ensures that the latest security updates are available. For building the "sandbox" mode of Singularity is used which means the "file system" is stored in a flat file structure. This build artefact is subsequently compressed to a tarball. The tarballs are then pulled by the CVMFS "Stratum 0" server (see "CVMFS") and deployed to CVMFS. The decoupling of the build service and the CVMFS servers ensures continued function if one of the services is out of order temporarily, and allows for easy extension to pull user images or other user-created data to CVMFS.

The images are kept for 30 days such that running jobs always keep the container they have started with (at present the maximal job runtime is set to one week—see "HTCondor"). Care has been taken to not make use of Singularity-exclusive functionality as far as possible such that the recipes can be carried over to a different container build software in the future.

CephFS

Many distributed file systems suffer from the fact that they need a central database to store information on the location of the data objects which might be distributed over many storage nodes. While such file systems scale quite well in terms of storage capacity, the metadata handling can become a problem for the systems. In addition, the database keeping the metadata can easily become a single point of failure.

Ceph [25] has been designed to overcome these problems. Instead of storing information on the location of data objects in a central database, Ceph keeps this information in a so-called CRUSH map (Controlled Replication Under Scalable Hashing) [65]. This approach allows clients to directly contact storage nodes by algorithmically determining the location of storage objects without consulting a metadata server. To ensure good scalability, the CRUSH map ensures a pseudo-random distribution of data objects. Another measure to ensure good scalability is the introduction of placement groups. Keeping track of object placement on a per-object basis becomes impractical with very large numbers of objects. Therefore, each object is assigned to a so called placement group (PG) and each PG is in turn assigned to one or more storage devices. The number of recommended PGs in a Ceph cluster depends on the number of storage devices [66], but a typical number is $\mathcal{O}(100)$ per storage device. This number is a compromise between resource usage (CPU, memory), data durability and data distribution; the latter should be as even as possible to maximize the usable space.

Ceph also allows to store multiple replicas of data objects to cope with data unavailability, e.g. due to hardware failure or maintenance work. To save storage space, Ceph can also apply erasure coding (EC) instead of replication. The simplest form of erasure coding divides data objects into $K = 2$ shards and adds $M = 1$ extra data chunk to ensure data reconstruction in case one storage device is lost/unavailable. This is the equivalent of RAID 5. The distribution of replicas/shards can be controlled by defining failure domains in the CRUSH map. Therefore, data redundancy is implemented in a host/rack/room/data centre/region-aware way. Thanks to Ceph's built-in data replication/erasure coding, a Ceph cluster can be run on commodity hardware. A key point of this is that there are commonly no RAID systems running below Ceph, but each single disk is handled by one Ceph object storage device (OSD) daemon and Ceph itself takes care of recovery in case of disk failures. For an EC setup, the reconstruction of the stored objects happens on the OSD servers themselves, which means the clients contact the primary OSD which in turn collects the K shards from the other OSDs to reconstruct the object, so downsides of the erasure coding technique are an amplification of overall network traffic and increased latency.

In addition to OSD servers, Ceph also requires so-called monitor (MON) daemons to watch the state of the cluster and distribute the OSD maps to the clients such that they are informed about the current CRUSH map to follow. They also take care of distributing this information and the history of these maps to the OSD daemons to coordinate potential recovery activities. These MON daemons are usually run on separate hardware from the OSD servers. To ensure a quorum and prevent split-brain scenarios in case the MON cluster is split into subsets which cannot communicate with each other, an uneven number of monitor daemons should be used. They can be combined with manager (MGR) daemons which monitor the Ceph cluster or execute administrative tasks at one central entry point.

All these components are designed with high availability and scalability in mind. For example, a cluster continues to operate as long as the majority of monitor nodes is online, and clients perform fail-over without intervention. Failing OSDs or complete hosts (depending on the CRUSH map) can be excluded and data can be rebalanced to the remaining hosts automatically. Increasing or decreasing the number of nodes does not enforce a downtime, hence the same is true for maintenance work on the actual hardware or software upgrades.

The Ceph object storage can be used as foundation for a multitude of different systems running on top. The most common ones are the Ceph Object Gateway, a RESTful [67] gateway to the Ceph object store, Ceph block devices commonly used as volumes in virtualization clusters, and the Ceph file system (short CephFS) which is a POSIX-compliant file system running on top of an existing Ceph cluster.

Our setup makes use of the described erasure coding with $K = 4$ and $M = 2$. This means 33% of the raw disk space is invested into redundancy. At least seven OSD servers are required for a robust operation with this configuration ($K + M + 1$). Otherwise the loss of a single node and *host* as failure domain does not allow to fulfil the desired storage constraints leading to an erroneous cluster state. While this is not a catastrophic event by itself, an additional disk failure can lead to a read-only state if only K data shards remain. In addition our tests revealed that the recovery from such a state likely triggers a chain reaction of problems (e.g. overloaded servers becoming effectively unavailable, additional disk failures). Furthermore, all data are compressed using the Snappy algorithm [68]. This is performed only if a heuristic check of the first blocks of the object suggests sufficient compressibility.

In terms of hardware configuration, we started off production with seven OSD servers offering 36 HDD slots each. While initially using two slots for small SSDs to keep the metadata information for Ceph's storage backend BlueStore [69] of 16 HDDs each per server, we have since migrated to the usage of larger PCIe-connected NVMe devices [70] for increased I/O operations per second. The HDDs are mostly of 4 TB size while we use 12 TB disks to replace broken disks for a fluent upgrading of the cluster. The NVMe devices which are keeping the metadata for 18 HDDs each in the changed configuration are of 1.6 TB size.

The memory usage by Ceph OSD processes can be quite substantial, especially during recovery or rebalancing situations when a failure of the OSD daemon would trigger additional recovery operations. For this reason, care must be taken that sufficient memory is available on all Ceph storage nodes. The oldest file servers are equipped with 192 GB of memory for 36 OSD processes. In current Ceph releases[3], the OSD processes make use of a memory target value to automatically trim their memory usage (for example, by reducing cache size). The default target value is 4 GB for an OSD process which works well in our configuration.

The hardware setup may not be considered as ideal, since a failure of a single NVMe means that the data of 18 HDDs is effectively lost. However, error reporting and life time estimates of NVMes for server usage are usually much more reliable than information on mechanical HDDs, and the used erasure coding allows for the temporary loss of two full servers without data loss. In terms of I/O throughput, the PCIe-NVMes are not heavily loaded even in large scale data rebalancing situations.

The metadata kept on those NVMe devices in the file servers are stored by Ceph in form of a RocksDB [71] database and used to save key/value data to find back the objects stored on the disk. These metadata are notably independent from the metadata information for the actual Ceph file system (which is also organized in a RocksDB), which we store on NVMe devices within three separate, dedicated nodes in a three-replica configuration. This separation was made to ensure the metadata server processes taking care of the file system metadata are as close as possible to the underlying storage for maximum performance. These dedicated nodes are also running the monitor and manager daemons.

Compared to other network file systems, locking in CephFS is a very granular process. Clients query the file system metadata server and are granted granular capabilities on files or directories, such as accessing file ownership information, extended attributes or access to the file contents in read, write, buffered writing and many more modes. This usually allows multiple clients to hold capabilities to the same file as long as the capabilities do not conflict with each other. In typical HPC / HTC workloads, parallel reading of files by many clients works without further metadata server interaction after fetching a capability to read the file once. Such a capability-based system necessarily relies on the ability of the metadata server to request capabilities back from the clients in case of memory pressure or urgent needs by other clients, and—in the worst case of a client not responding—evicting the client. This is implemented in CephFS and care must be taken that clients do not stop network communication for a prolonged time due to system overload to prevent unwanted eviction which is one of the reasons for the health checking mechanism described in "HTCondor".

Network connectivity is established both via classic 1 Gbit/s ethernet for administrative access and InfiniBand making use of the IP-over-InfiniBand stack. While Remote Direct Memory Access (RDMA) [72] could technically also be used with Ceph, it did not work in our setup. The nodes almost immediately lost communication over the RDMA channel. It is likely that this is both due to usage of the older Mellanox OpenFabrics Enterprise Distribution (OFED®) [73] stack shipped with CentOS 7 which we have been using and missing improvements within Ceph which are currently under development.

For this reason, we configured our machines to use the IP-over-InfiniBand stack for communication. Increasing the maximum transmission unit (MTU) to the maximum possible value and switching the cards to connected mode allows for sufficient throughput: using InfiniBand FDR (Fourteen Data Rate) hardware which allows for a maximum throughput of 56 Gbit/s, we achieve 20 Gbit/s to 25 Gbit/s in a direct, simplex (uni-directional) test between two nodes using four

[3] This is valid starting from version 12.2.9 of the Luminous release series, 13.2.3 of the Mimic release series, and all stable Nautilus or newer releases.

parallel streams over the IP-over-InfiniBand stack. This total throughput is maintained in duplex mode which indicates the limitation is not from the wire, and further improvements can be expected from an update of the software stack or future production-ready RDMA support within Ceph.

Our clients make use of the CephFS FUSE client to allow for timely and safe upgrades to upstream releases, and to make use of all functionality[4]. The increased latency of system calls slowing down operations such as `open`, `close` and `stat` are not harmful in a HPC/HTC environment with a focus on high energy physics data, since data are usually stored in big files which are only rarely rewritten and often re-read. As a welcome side effect, bursts of metadata queries are throttled on the client side which effectively means a single user can not overload the metadata server with queries anymore—a situation which we regularly faced with the BAF1 Lustre system.

Additionally, we employ quotas and access control lists (ACLs). In CephFS, quota support is implemented within the clients, and quotas are bound to directories. They recursively apply to all data stored within and can be set both for the consumed space and the number of inodes. The necessary recursive information is stored within the metadata of CephFS which allows a client to immediately return the recursive directory size and contained number of inodes. This is exposed both via extended attributes and the classic `stat` syscall, so usual Linux commandline utilities can leverage this information and recursive `stat` syscalls are not necessary anymore. The quotas themselves are also stored within extended attributes and a convenient script to check the available and used quota easily is provided for all users.

Since cluster file systems in general do not perform well with many small files, we limit the space for each user to 500 GB and 100,000 inodes by default. The latter is mostly meant as a "tripwire" to catch misuse of the cluster file system for storage of software which is better kept on CVMFS (see "CVMFS"). Instead, we encourage users to compile on scratch space and use the HTCondor file transfer mechanism or store the software either in a tarball on the cluster file system or on CVMFS (see "CVMFS"). ACLs in turn are used to offer group storage and allow sharing of data between users. In addition users of the ATLAS experiment have several hundreds of terabytes of group space which is reserved for automatic data transfers handled by the distributed data management (DDM) system (see "XRootD").

Finally, it must be noted that both Ceph and CephFS are evolving quickly. CephFS has recently added the ability to snapshot directory trees and even optionally expose this functionality to users[5], and an implementation of data deduplication on the RADOS layer is under investigation.

[4] ACL (access control list) and quota support are not yet available in longterm support distribution kernels.

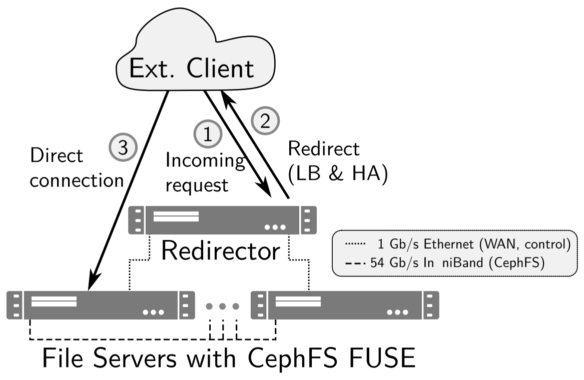

Ext. Client
Direct connection
3
1
Incoming request
2
Redirect (LB & HA)
Redirector
1 Gb/s Ethernet (WAN, control)
54 Gb/s In niBand (CephFS)
File Servers with CephFS FUSE

Fig. 3 Schematic overview of the data flow for the XRootD setup in Bonn. Incoming requests are initially received by the redirector which redirects requests to a group of file servers (in our case the Ceph OSD nodes which additionally run CephFS FUSE clients). This way the load between the file servers is shared and the setup is insensitive to outages of individual file servers

XRootD

XRootD [24] is a modular software framework to provide a well-scalable and fault-tolerant data access optimized for performance. It is commonly used to grant access to file-based data. It also incorporates authentication and authorization functionalities and can be easily extended with plugins.

The data transfer protocol is highly efficient also for small random reads, and multiple streams can be used to exhaust high bandwidth or high latency links. Native implementations of the XRootD protocol exist for example in the analysis software ROOT [74] which allows one to read files exported via XRootD directly.

XRootD can also handle large distributed data, building a file system with an hierarchical namespace from several distributed disks or even whole clusters. For that reason, it is often used at lightweight sites to replace a distributed file system. Recent developments like XCache [75] also offer transparent caching.

In our case, XRootD is used for connectivity to the Worldwide LHC Computing Grid (WLCG) [76] and to offer storage for usage by distributed data management (DDM).

There are in general two setups a site could implement for this use case. One possibility would be to use separate XRootD servers each exporting their own disk space, and cluster them together behind an XRootD redirector.

[5] It has been considered to make use of this functionality, but the CephFS developers are still fixing some caveats.

Since a highly performing distributed filesystem is already in use at Bonn (see "CephFS"), we have chosen another model for the implementation. One XRootD server process is running per OSD server, each exporting the full CephFS, and a central redirector running on a virtual machine handles all incoming connections (read and write requests), checks authentication and authorization and automatically load-balances these to all available data servers as shown in Fig. 3. The XRootD redirector automatically takes machine load[6] and availability into account such that no separate high availability setup is necessary. Since the protocol allows one to redirect the client to the data server directly, the redirector itself has very low load and does not present a bottleneck in this configuration. Furthermore, the redirector node does not need access to the underlying cluster file system, but is aware that all data servers serve the same file system and can cache file metadata (e.g. file existence).

In addition to the native XRootD protocol, basic WebDAV [77] commands can be handled by the very same infrastructure. HTTP redirects are used to redirect the client to the corresponding data server. This allows WebDAV compatible clients to store and retrieve data transparently.

For authentication and authorization, X.509 certificates are used throughout WLCG which are extended by attributes from the Virtual Organization Membership Service (VOMS) [78] granting special permissions, such as write access to ATLAS sites in Germany. XRootD verifies the X.509 certificates and handles these VOMS extensions via a dedicated plugin.

In the context of WLCG, it must be noted that the bulk of all data transfers happens via a Third Party Copy (TPC) mechanism [79], which requires extensions to the data transfer protocols to delegate authentication and authorization. This means that a central instance (Rucio) [23] is instructed to initiate a copy of a file from site A to site B. This can happen by policy (e.g. requirement of a minimum number of copies), automatically (by robots or automated file restoration) or by user request. The central instance, in turn, makes use of the File Transfer Service (FTS) [80] to actually perform the copy. The FTS server contacts source and destination servers and tries both a "push" and "pull" mechanism to establish a direct copy operation between the involved servers. A fallback to streaming the data is possible, but does not scale to the required bandwidths. For these "push" and "pull" techniques, the FTS server needs to delegate authentication and authorization to the servers.

[6] The load estimate is based on the most loaded network interface's load, CPU load, system load average, and the fraction of available memory.

This is achieved using various techniques. Currently X.509 proxy certificates are commonly used for XRootD while for WebDAV so-called macaroons[7] [81] are employed. Developments to move to a token-based authentication within WLCG are ongoing and some communities have already switched to a workflow based on SciTokens [82–86].

Cluster Management

We deploy and run all Linux machines—thus not only the BAF2 cluster—in our institute using Puppet [27] and Foreman [26]. In total, these are more than 350 nodes at the time of writing, some of which are bare metal installations and others are virtual machines.

Puppet is a configuration management tool that ensures that all nodes are in the desired state as specified by so-called manifests which in turn specify required system resources. Thanks to this abstraction layer it is rather easy to run different Linux flavours (e.g. representatives of the Debian and the Red Hat family) using the same language. Puppet takes care of a large part of the translation work to convert those resource statements into specific flavour-dependent commands to execute. Those manifests are written using a declarative language. Manifests can be bundled in modules which typically combine all the required functionality for running a certain service, e.g. installing, configuring, enabling and starting an NTP service.

The Puppet architecture follows a client–server approach where Puppet agents (clients) communicate with a Puppet master (server) or vice versa. A Puppet agent is a service required to run on all nodes under Puppet control. It is usually executed periodically to ensure the system state matches the desired state retrieved from the server which interprets the manifests and compiles a client-specific catalogue. Furthermore, the server collects machine-specific, extensible facts reported by the Puppet agent, which can in turn be used to define resources for the very same or other clients dynamically. One example for this would be automatic installation of RAID controller tools if the corresponding hardware is detected. The communication is secured using X.509 certificates which are typically issued by a Puppet certificate authority (CA)[8]. If communication to the server is interrupted, a cached copy of the catalogue can be used.

Due to the widespread use of Puppet a huge amount of Puppet modules has been developed by the Puppet user community. Checking Puppet Forge [87] is a good starting point if one wants to "puppetize" a service. From the rich repertoire of available modules, we have collected more than 100 Puppet modules used by us on GitHub (https://github.com/unibonn). Most of those repositories are forks.

[7] Maracoons are bearer tokens encapsulating limited capabilities and with expiration time.

[8] It is also possible to use an external CA.

Only in rare cases, we were forced to develop our own module because we did not find anything suitable on the market. If we found a well-designed module which lacked some functionality, we expanded it and tried to bring the patch into the upstream code. Although this typically means some small extra effort, we consider it worth the work in the long run since it simplifies synchronizing our fork with updates in the upstream code.

We keep track of all those modules using the Puppet environment and module deployment tool `r10k` [88]. It allows to specify which modules one wants to use for which Puppet environments. In addition the branch or the commit to be used can be specified.

Foreman is a host management tool. It allows automatic installations, configuration management, power state switching and similar actions required by host operators. Apart from bare metal installations, it can also deploy virtual machines. To accomplish this, it provides interfaces to libvirt [89], OpenStack [90], oVirt [91], VMware [39], Amazon Elastic Compute Cloud [92], Google Compute Engine [93] and more. If desired, Foreman can also handle DNS and DHCP entries, manage Puppet masters and the Puppet public key infrastructure. In addition to its base functionality, there are numerous plugins available [94]. They add support for more compute resource technologies (e.g. Docker), configuration management systems different from Puppet, provisioning extensions (e.g. Metal-as-a-Service), integration with monitoring or IP address management systems, and many more functionalities.

Authorization is another important feature of Foreman. It allows to define fine-grained permissions who can access, modify or delete which resources.

Foreman's web interface provides an overview of the Puppet health status of the Foreman managed hosts. It is easy to spot hosts where Puppet runs have failed. In case of failures one can nicely trace back the source of the problem and when it showed up for the first time. Foreman also offers auditing, allowing one to see who performed which change when. These features are very helpful for debugging purposes.

In addition to its web interface, Foreman also provides a commandline tool called *hammer* [95]. This is convenient for automized tasks or bulk changes.

Our Puppet module structure is based on the *roles and profiles* method [96, 97] although we do not follow it strictly due to the additional availability of Foreman. One difference is the absence of roles in our setup. They are basically substituted by Foreman host groups. Profiles use other modules to configure services in the desired way. A file server profile could e.g. set up the file service and in addition open the required firewall ports. If services have to be configured differently for different host types, we introduce Puppet class parameters for the corresponding profiles. Those parameters are set in Foreman for specific host groups, individual hosts or depending upon facts reported by the Puppet agent, such as specific hardware models. In order to avoid too crowded Foreman class listings, we have set up a filter which ensures that only profile classes are visible in Foreman.

HTCondor

HTCondor [14–21] is a workload management system optimized for high throughput computing. It has been formerly known as Condor from 1988 to 2012. Users describe their workload using a job description language and submit it to the workload management system on their submission node which are in our case centrally managed Linux desktop machines. These in turn announce the workload to the available cluster resources, and a matchmaking procedure takes place which fits the requested with the available resources.

The advantages of HTCondor as compared to other workload management systems such as TORQUE/Maui or Slurm [98, 99] are the very flexible and extensible job description language and the ClassAd [100] mechanism. All resources, requests or constraints are expressed as ClassAds, which are a set of uniquely named expressions that allow for evaluation, comparison and merging. They are the foundation for the matchmaking procedure and allow for a flexible introduction of custom resources and policies. Additionally, HTCondor offers support for containerized jobs, GPU resources, spilling over jobs to external clusters or even cloud resources and complex job dependencies which can be expressed as a directed acyclic graph, which is actively employed by our users to describe complex job pipelines and expose their dependencies to the scheduler.

Our choice of HTCondor over the other available solutions was motivated by the high flexibility and the native support for containerized jobs. One major requirement of existing users is to run their software in the very same environment used by their collaboration, which may be an outdated operating system. The decoupling of actual operating system and job runtime environment is described in more detail in "Containerization".

We have configured all worker nodes to be completely isolated in a private network hidden behind a network address translation (NAT) gateway, and only be accessible via HTCondor. This is facilitated by mediating connections via an HTCondor Connection Broker (CCB) running on the HTCondor Central Manager machines operating as redundant pair within the desktop network. Since HTCondor also allows for interactive jobs including forwarding of X11 by leveraging SSH even if a private cluster network is used, users can access the worker nodes in a controlled way with defined resource constraints.

Additionally, we have configured HTCondor to enforce a container environment to be used for each job, such that the user never accesses the bare metal machine. This increases the portability of jobs, and they could be executed on a different cluster if needed unless direct access to the cluster file system or other local resources is explicitly required. The user can choose between different, centrally maintained containers by adding a corresponding expression to the job ClassAd.

Due to the ClassAd functionality, a major difference between HTCondor and other workload management systems is that HTCondor does normally not have a concept of different job queues with varying resource constraints, but shares the resources fairly (or based on self-defined constraints) with dynamic partitioning.

This comes with several advantages over the approach employing fixed queues:

1. Resource allocation for jobs of different classes is only limited by the physical availability of resources, i.e. all resources fulfilling specified requirements are available to any job (unless administratively imposed constraints are applied),
2. Users are forced to specify their required resources in a more fine-grained way than by choosing a queue, which may be slightly mismatched to the actual requirements,
3. Run priorities can be handled more fairly, since they are not based on the queue, but on the actual resource reservation and usage.

All these advantages lead to a higher throughput and fairer scheduling. However, the flexible approach also leads to several operational problems as compared to queue based schedulers:

1. Jobs needing different resources than the bulk of jobs might have to wait for a longer time until matching resources are available,
2. Interactive resources may be used up completely.

HTCondor offers a `DEFRAG` daemon which can be configured to regularly drain random machines fully or partially to overcome parts of these issues. This naturally reduces the overall throughput of the system slightly.

In our case, we are not employing the `DEFRAG` daemon yet[9], but have overcome all of these problems in a satisfactory way by implementing a custom health check and machine reboot automation system. Additionally, the custom implementation also detects misbehaving or inefficient jobs and protects the compute nodes from entering an unhealthy system state by preventing that further jobs are sent to an unhealthy machine.

The custom health checking script which is designed to be extremely lightweight in terms of resource consumption is executed by the `STARTD` which is the HTCondor daemon handling the compute jobs on the compute node every minute. It produces expressions describing the current health state both in a simple logic information that can be consumed programmatically and in human and machine readable, detailed form. These are automatically merged into the configuration of the `STARTD` and become part of the machine's ClassAds, and are not only referenced in the expression which defines whether new jobs are accepted by the node, but are also visible to the HTCondor central manager nodes and hence can be monitored easily by both users and monitoring systems. Due to the low execution period, the script can actually throttle the job start rate, for example in case the I/O to the local hard drive of the worker node is overloaded by too many scratch intensive jobs starting in parallel.

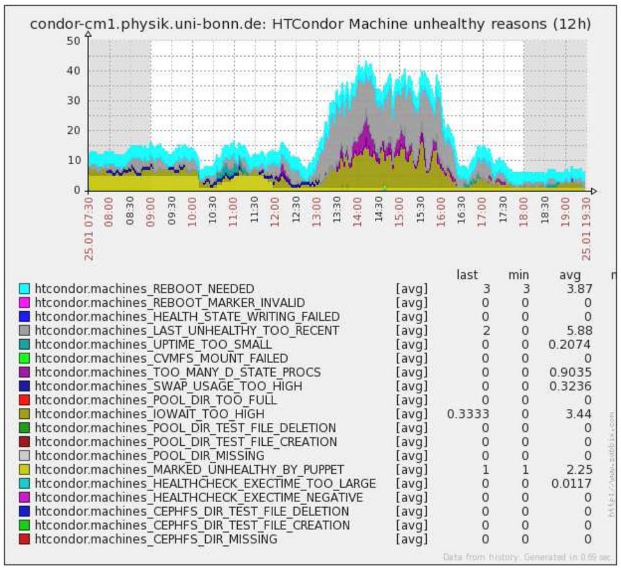
The list of checks performed by the script is easily extensible. As of now, a node is set unhealthy if at least one of the following conditions is met:

- The HTCondor pool directory (scratch space for jobs) either does not exist, is not writable or has insufficient free space;
- The cluster file system CephFS is either not accessible or not writable;
- Number of processes running in `D` state (as a fraction of the machine's CPU cores) exceeds a threshold;
- Swap space almost completely exhausted;
- High I/O wait CPU percentage;
- One or more CVMFS repositories are not accessible;
- System uptime is too small;
- Speed of network interface too low (has reduced to less than 1 Gbit/s due to a hardware issue);
- Kernel command line incorrect (allows administrative reconfiguration).

In addition to these checks, the overall execution time of the script is taken into account and if considered too high, the machine may be marked as unhealthy. Further external input for the script is provided by:

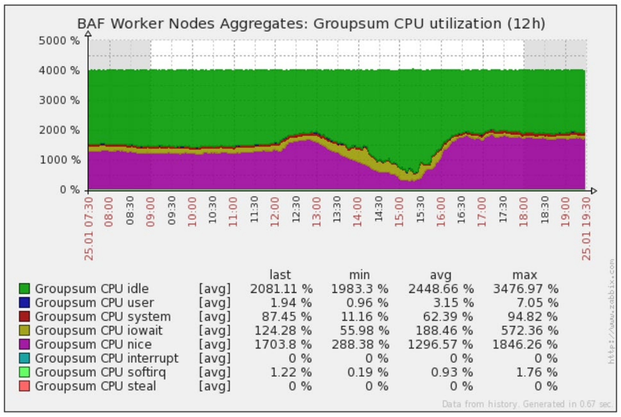
- Administratively configured health state deployed via Puppet;
- Reboot of node needed, deployed via automated reboot script.

[9] The `DEFRAG` daemon might still be useful for jobs requiring a significant fraction of a node's resources when all nodes are used by single-core jobs.



			last	min	avg
🟦	htcondor.machines_REBOOT_NEEDED	[avg]	3	3	3.87
🟪	htcondor.machines_REBOOT_MARKER_INVALID	[avg]	0	0	0
🟦	htcondor.machines_HEALTH_STATE_WRITING_FAILED	[avg]	0	0	0
⬜	htcondor.machines_LAST_UNHEALTHY_TOO_RECENT	[avg]	2	0	5.88
🟦	htcondor.machines_UPTIME_TOO_SMALL	[avg]	0	0	0.2074
🟩	htcondor.machines_CVMFS_MOUNT_FAILED	[avg]	0	0	0
🟪	htcondor.machines_TOO_MANY_D_STATE_PROCS	[avg]	0	0	0.9035
🟦	htcondor.machines_SWAP_USAGE_TOO_HIGH	[avg]	0	0	0.3236
🟥	htcondor.machines_POOL_DIR_TOO_FULL	[avg]	0	0	0
🟨	htcondor.machines_IOWAIT_TOO_HIGH	[avg]	0.3333	0	3.44
🟩	htcondor.machines_POOL_DIR_TEST_FILE_DELETION	[avg]	0	0	0
🟥	htcondor.machines_POOL_DIR_TEST_FILE_CREATION	[avg]	0	0	0
⬜	htcondor.machines_POOL_DIR_MISSING	[avg]	0	0	0
🟨	htcondor.machines_MARKED_UNHEALTHY_BY_PUPPET	[avg]	1	1	2.25
🟦	htcondor.machines_HEALTHCHECK_EXECTIME_TOO_LARGE	[avg]	0	0	0.0117
🟪	htcondor.machines_HEALTHCHECK_EXECTIME_NEGATIVE	[avg]	0	0	0
🟦	htcondor.machines_CEPHFS_DIR_TEST_FILE_DELETION	[avg]	0	0	0
🟩	htcondor.machines_CEPHFS_DIR_TEST_FILE_CREATION	[avg]	0	0	0
🟥	htcondor.machines_CEPHFS_DIR_MISSING	[avg]	0	0	0

Fig. 4 Health states of the various worker nodes. The states are cumulated and any node may enter multiple health states, so the total can exceed the number of worker nodes. Dominating colours are bright yellow (administratively marked out), dark yellow (too high I/O wait CPU), grey (the node was unhealthy recently), cyan (draining for reboot) and purple (many processes in D state)

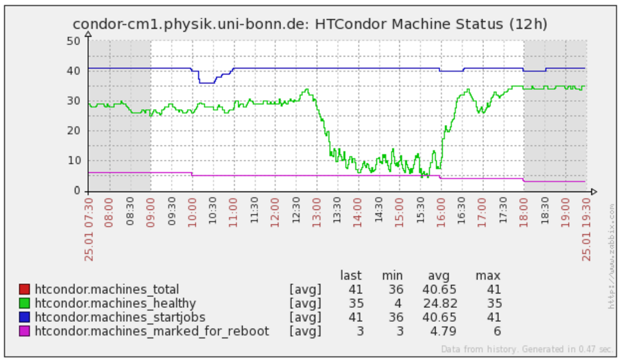


			last	min	avg	max
🟩	Groupsum CPU idle	[avg]	2081.11 %	1983.3 %	2448.66 %	3476.97 %
🟦	Groupsum CPU user	[avg]	1.94 %	0.96 %	3.15 %	7.05 %
🟥	Groupsum CPU system	[avg]	87.45 %	11.16 %	62.39 %	94.82 %
🟨	Groupsum CPU iowait	[avg]	124.28 %	55.98 %	188.46 %	572.36 %
🟪	Groupsum CPU nice	[avg]	1703.8 %	288.38 %	1296.57 %	1846.26 %
🟦	Groupsum CPU interrupt	[avg]	0 %	0 %	0 %	0 %
🟩	Groupsum CPU softirq	[avg]	1.22 %	0.19 %	0.93 %	1.76 %
🟥	Groupsum CPU steal	[avg]	0 %	0 %	0 %	0 %

Fig. 5 The CPU utilization across all worker nodes (percentages of logical core usage for each node are stacked, thus summing up to (number of nodes) x 100%). The most prominent colours are green for "idle" load, red for system time, yellow for I/O wait CPU and purple for "nice" CPU time. All user jobs are re-niced so their actual compute time is part of the "nice" region

Furthermore, to prevent flapping between healthy and unhealthy states, the script only marks a machine as healthy again if it has been consistently found healthy for a minimum time. This prevents issues for example in case of fluctuating I/O patterns.

If any of these checks fails for several minutes, the information is propagated to the central manager and it is



			last	min	avg	max
🟥	htcondor.machines_total	[avg]	41	36	40.65	41
🟩	htcondor.machines_healthy	[avg]	35	4	24.82	35
🟦	htcondor.machines_startjobs	[avg]	41	36	40.65	41
🟪	htcondor.machines_marked_for_reboot	[avg]	3	3	4.79	6

Fig. 6 Number of worker nodes and their status as function of time. The red line corresponds to the total number of worker nodes, the blue line is the number of nodes with `StartJobs = True`, the green line shows the number of healthy worker nodes and purple displays the number of worker nodes that are marked for a reboot (independent of actual draining for reboot)

recorded by our monitoring system. An example for the system reacting to an I/O overload triggered by jobs is shown in Fig. 4. Quickly, the cluster was filled by jobs heavily overloading the local scratch disks, leading to a state of high I/O wait CPU for all nodes, as visible in Fig. 5. The healthcheck marked the nodes as unhealthy as shown in Fig. 6 preventing more inefficient I/O overload which would also affect other running jobs (potentially including interactive jobs) on these worker nodes. Of course, the resources are marked as unavailable during that period, so user education is necessary to prevent the issues in the first place. Still, the health check prevents immediate damage which could be caused e.g. by nodes becoming completely unresponsive and losing network file system access in the process.

In addition to the health check system, a cron job executed once per hour checks whether the machine is in need of a reboot. If this is the case, a reboot marker file is created which contains the reboot reasons for consumption by the health check script. The eventual reboot reasons are also added to the machine ClassAd.

Actual indicators for a necessary reboot may not only be updated system components which require a reboot (such as the kernel), but also a machine is marked for reboot after a maximum uptime (30 days in our case) has been reached.

To prevent draining all nodes in parallel, a pseudo random offset is assigned to each node based on the FQDN of the machine and the health check script only marks the node as "unhealthy" once a minimum time and the offset has passed after the initial reboot marking. With this approach, the times when the machines start to be drained are spread out across a larger time scale (10 days in our case) than the maximum expected job runtime (seven days in our case) to maximize the throughput of the cluster.

Whenever a machine marked for reboot is empty from running jobs during cron job execution, it is immediately removed from the HTCondor cluster and marked for an automatic reboot, even if the random offset time has not yet passed. This allows the automated system to reboot all machines automatically when there are no jobs running instead of enforcing a draining of the nodes at a later time when this may lead to blocking of needed resources. This effect is also visible in Fig. 6 which shows that the number of total machines is temporarily reduced when a machine is removed from the cluster and rebooted. At any time, a reboot can be prevented administratively.

Using this approach, automatic updates are not an issue anymore for the compute nodes and it is feasible to comply to requirements by computer emergency response teams (CERTs). Furthermore, in case, hardware maintenance needs to be performed, a node can be drained administratively. Both administrators and users can always query the status of the machines and, if needed, planned reboots of all machines in a human-readable format.

Additionally, the regular reboots automatically lead to a defragmentation of a small fraction of the nodes, hence operating not unlike the HTCondor `DEFRAG` daemon, in a less deterministic, but less resource consuming way.

Furthermore, we have reserved some nodes (or parts of them) exclusively for interactive jobs, such that these can usually be started within a few seconds after a request by the user. This allows to use interactive jobs like “login nodes”, but with the full flexibility of choosing one of the centrally managed containers, and the possibility to request and use a defined set of resources, reducing interference with other interactive users.

In terms of actual machine configuration, all compute nodes have 128 GB of swap space on a local, spinning disk, where also the scratch space for user jobs resides. HTCondor is configured to employ cgroups v1 [45] to limit resource usage to what was requested by the user. This limits the CPU share and maximum resident memory. This leads to some special features:

1. In case a job spawns more threads and no other jobs are running on the node, it is allowed to access all cores on the node.
2. In case a job tries to allocate more memory than it has reserved, it can spill over to swap. It is only killed when also the swap space is filled up.

Especially the last of these points may come as a surprise. However, we actually observed users running jobs with short-term peaks in memory usage, or significant amounts of dead / rarely used allocated memory that can be easily spilt to swap and there is almost no gain from keeping it resident[10]. By offering the swap space, there is effectively

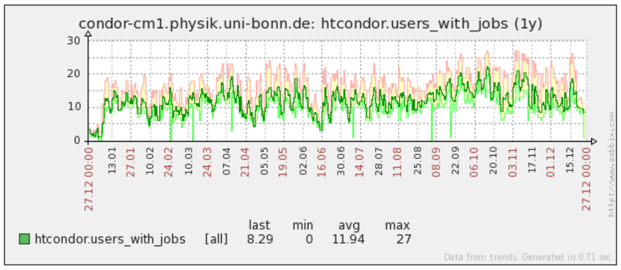


Fig. 7 Number of users with submitted jobs versus time. The dark green line displays the average while the light green and the light pink lines show the minimum and maximum values, respectively

more memory available for the system page cache which increases throughput. For jobs which allocate and never use their memory at all, we utilize zswap [101] to automatically compress pages and still keep them in memory before swapping them out to disk.

In case a misbehaving job exceeds the reservation drastically, the health check system will automatically block the node due to the high I/O wait CPU percentage. The same is true if swap is almost full. It also allows to offer job suspension functionality in the future, both in case jobs cause nodes to become unhealthy and if higher priority jobs arrive.

In addition to the cgroup limit on CPU and memory, disk space on the scratch disk is requested by users and accounted for by HTCondor, but we do not enforce this limit yet.

Another special part of our configuration is that we have enabled simultaneous multithreading (SMT) on all worker nodes and fully expose all logical cores as possible resources. This choice was made since the job workloads are very diverse. A significant fraction of jobs waits for data from storage or memory rather than for CPU cycles. The additional capacity to store processor states due to SMT allows better CPU usage efficiency. Cores assigned to a cgroup specify a share of the available cores and are not explicitly pinned. In addition, the usage of cgroups allows jobs to use more cores than requested if they would be idle otherwise. For this reason, we expect that thanks to the diversity jobs will on average profit both from the availability of SMT and also from exposing the logical cores as slots.

Finally, it must also be noted that there are still several areas where HTCondor is not yet perfect. One example for this is the handling of interactive jobs in containers. This was realized by starting an `SSH` daemon inside the container with user privileges, and then connecting to that from outside. This requires several tricks to work in an environment with enabled security mechanisms such as SELinux [102],

[10] This commonly happens when code with memory leaks or hoards is running, or programs written in languages employing garbage collection are executed.

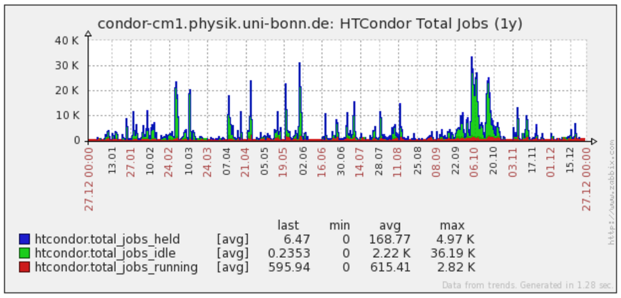


Fig. 8 Number of running, held and idle jobs versus time. The three categories are stacked

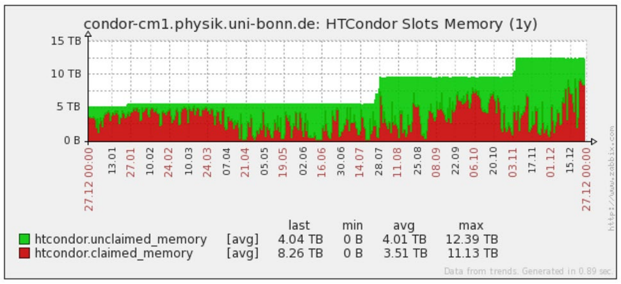


Fig. 10 Amount of claimed (red) and unclaimed (green) memory versus time. Both plots are stacked such that the sum always amounts to the total amount of memory in all worker nodes. The clearly visible steps correspond to performed RAM upgrades

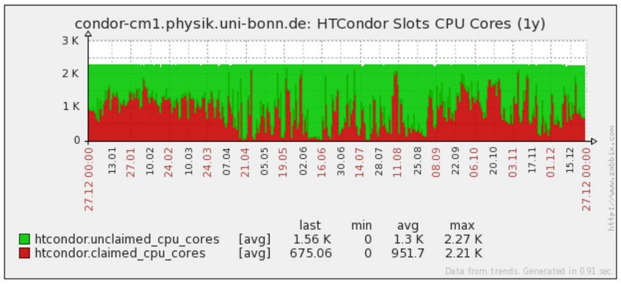


Fig. 9 Number of claimed (red) and unclaimed (green) CPU cores versus time. Both curves are stacked such that they always sum up to the same total number of logical CPU cores (2240), unless there are nodes taken offline

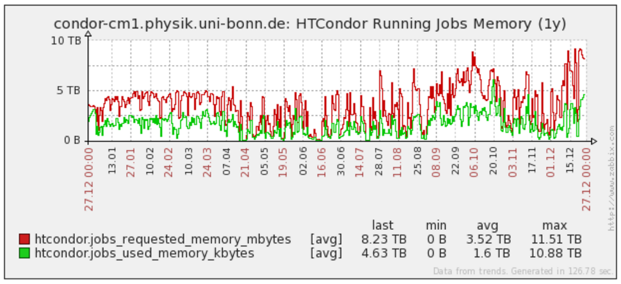
and means that all containers need to ship an `SSH` daemon. Recent versions of HTCondor (starting with release 8.8.0) have changed from this model to the usage of `nsenter` which is a tool used to attach a new process to running namespaces. This allows to run the `SSH` daemon outside of the container, but still enter the very same environment. At the time of writing, there are still open issues with the implementation of the new approach, but it is the most promising solution which will work with all container runtimes.

Operational Experience

We have run the setup described in the previous sections for almost two years now. Using this cluster meant a breaking change for the users. Compared to the BAF1 setup they not only had to learn how to use a new batch system but—due to the absence of login nodes—also had to adopt a new workflow. The changes were quickly and well accepted. Out of currently more than 130 registered users a fraction (with varying composition) of typically 10–15% have jobs in the queue (including interactive ones—see Fig. 7).

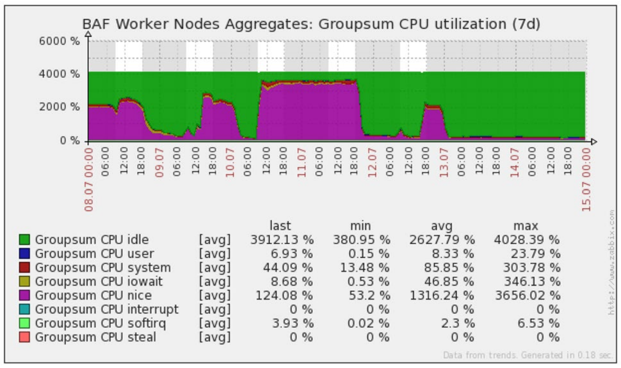
Figure 8 shows the number of submitted jobs versus time. The submitted jobs are split into three different categories depending on the job status: running (red), idle (green) and held (blue). In particular, the number of idle jobs is subject to large variations over time. Occasionally, there are more than 30,000 jobs waiting for free resources. This has to be related to the maximal number of available slots which is given by the number of virtual CPUs in the cluster worker nodes. Due to activated simultaneous multithreading (SMT), this number amounts to 2240 for the currently available cluster hardware. As a consequence the number of running jobs is only visible as a thin red area at the bottom of the plot. Held jobs usually represent a tiny fraction of all jobs. This state indicates a problem with the affected jobs. For our setup, most jobs in hold state have expired Kerberos tickets preventing access to the home directory. Unfortunately, the present version of HTCondor (8.8.7) does not offer convenient Kerberos ticket handling. It, therefore, falls on the user to take care that (s)he always keeps a valid Kerberos ticket granting ticket on the submit node of her/his jobs.

Figure 9 gives an idea of the average cluster utilization, The red entries show the number of claimed CPU cores and the green area denotes unclaimed compute resources. Both information is stacked such that they always sum up to the same number of CPU cores (2240) except for periods when nodes are taken out of the cluster for maintenance. It is apparent that even during times with tens of thousands of pending jobs, the compute resources are not fully used. This is partly since we reserved approximately 100 virtual CPUs for interactive jobs to ensure a quick availability of the requested container environment for interactive work. Second, there are often some “unhealthy” nodes for various reasons (see below). Such nodes are not accepting new jobs until they recover. Another reason is that limitations by other cluster resources like e.g. memory prevent full exploitation of all CPU resources. Figure 10 sheds some light on the memory utilization of the cluster. While in particular in the first third of the displayed interval the available memory has a very high average utilization, memory often cannot be fully utilized although it is limiting the usage of computing



		last	min	avg	max
🟥 htcondor.jobs_requested_memory_mbytes	[avg]	8.23 TB	0 B	3.52 TB	11.51 TB
🟩 htcondor.jobs_used_memory_kbytes	[avg]	4.63 TB	0 B	1.6 TB	10.88 TB

Fig. 11 Total requested (red) and used (green) memory of all jobs as function of time



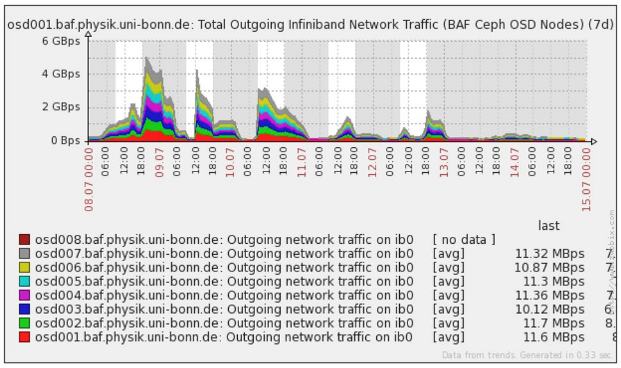
		last	min	avg	max
🟩 Groupsum CPU idle	[avg]	3912.13 %	380.95 %	2627.79 %	4028.39 %
🟦 Groupsum CPU user	[avg]	6.93 %	0.15 %	8.33 %	23.79 %
🟥 Groupsum CPU system	[avg]	44.09 %	13.48 %	85.85 %	303.78 %
🟨 Groupsum CPU iowait	[avg]	8.68 %	0.53 %	46.85 %	346.13 %
🟪 Groupsum CPU nice	[avg]	124.08 %	53.2 %	1316.24 %	3656.02 %
🟦 Groupsum CPU interrupt	[avg]	0 %	0 %	0 %	0 %
🟩 Groupsum CPU softirq	[avg]	3.93 %	0.02 %	2.3 %	6.53 %
🟥 Groupsum CPU steal	[avg]	0 %	0 %	0 %	0 %

Fig. 12 Varying CPU load of all worker nodes over a week of intensive usage. The most prominent colours are green for "idle" load, red for system time, yellow for I/O wait CPU and purple for "nice" CPU time. All user jobs are re-niced so their actual compute time is part of the "nice" region

resources. This happens when the way the memory is distributed over the worker nodes does not match sufficiently well the memory requirements of the respective jobs on those nodes. This effect is particularly pronounced if there are many multi-core jobs requesting a significant fraction of the total available RAM per worker node. The effect should be kept in mind when evaluating cluster utilization.

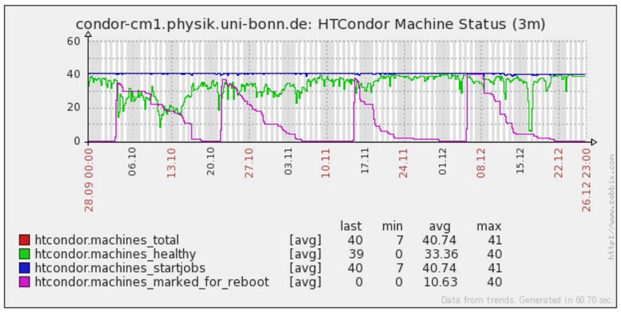
This discussion also underlines that it is important that users specify their needed resources reliably. Otherwise unnecessarily large amounts of cluster resources become "unusable". How well the users' estimated resources match their jobs' actual usage is displayed in Fig. 11 for memory as an example. Obviously the requested memory is on average roughly a factor of two larger than the actual usage. This is not an alarming disagreement but still there is room for improvement. Getting better estimates requires monitoring the situation, making users aware of the problem and teaching them how to determine reliable estimates. This is quite personnel-intensive work.

A possible way to reduce the amount of those "unusable" cluster resource remnants which do not fit the requirements of the available jobs at a given time is to backfill the cluster



		last	
🟥 osd008.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[no data]		
⬜ osd007.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	11.32 MBps	7.
🟨 osd006.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	10.87 MBps	7.
🟦 osd005.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	11.3 MBps	
🟪 osd004.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	11.36 MBps	7.
🟦 osd003.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	10.12 MBps	6.
🟩 osd002.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	11.7 MBps	8.
🟥 osd001.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	11.6 MBps	8

Fig. 13 Outgoing network traffic from the Ceph OSD nodes over a week of intensive usage. The traffic through the InfiniBand devices of the nodes is stacked in this visualization



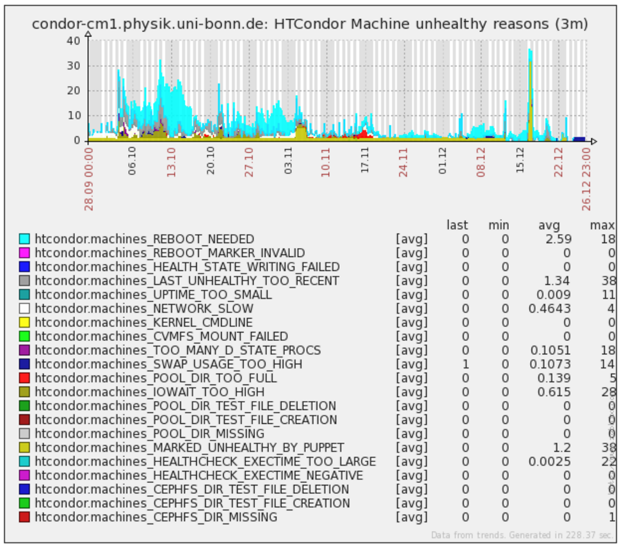
		last	min	avg	max
🟥 htcondor.machines_total	[avg]	40	7	40.74	41
🟩 htcondor.machines_healthy	[avg]	39	0	33.36	40
🟦 htcondor.machines_startjobs	[avg]	40	7	40.74	41
🟪 htcondor.machines_marked_for_reboot	[avg]	0	0	10.63	40

Fig. 14 Number of worker nodes and their status as function of time. The red line denotes the total number of worker nodes, blue is the number of nodes with `StartJobs = True`, green shows the number of healthy worker nodes and purple displays the number of worker nodes that are marked for a reboot (independent of actual draining for reboot)

with better suiting jobs. This is a topic which we are currently investigating.

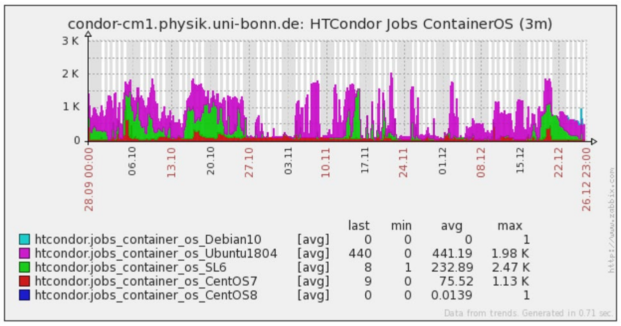
The same phenomenon is apparent when looking at the CPU usage and I/O to the storage system over time. Figure 12 shows the CPU load over a busy week. At times, the cluster was used up to almost 100%, with a small fraction being reserved for interactive usage. However, before this high CPU load period, the jobs were apparently bound by I/O from the cluster file system as becomes apparent when comparing the data with Fig. 13[11]. The outgoing traffic from the Ceph OSD servers usually saturates at about 5 GB/s. It must be taken into account that the traffic partially goes to other file servers for reconstruction of the

[11] CPU I/O wait from accessing CephFS is usually negligible in these cases both since CPU time is actively spent polling the network by the CephFS FUSE client and since other CPU tasks are often available to take over the waiting CPU cores. Actual I/O wait is mainly visible on the OSD nodes themselves.



			last	min	avg	max
🟦	htcondor.machines_REBOOT_NEEDED	[avg]	0	0	2.59	18
🟪	htcondor.machines_REBOOT_MARKER_INVALID	[avg]	0	0	0	0
🟦	htcondor.machines_HEALTH_STATE_WRITING_FAILED	[avg]	0	0	0	0
⬜	htcondor.machines_LAST_UNHEALTHY_TOO_RECENT	[avg]	0	0	1.34	38
🟦	htcondor.machines_UPTIME_TOO_SMALL	[avg]	0	0	0.009	11
⬜	htcondor.machines_NETWORK_SLOW	[avg]	0	0	0.4643	4
🟨	htcondor.machines_KERNEL_CMDLINE	[avg]	0	0	0	0
🟩	htcondor.machines_CVMFS_MOUNT_FAILED	[avg]	0	0	0	0
🟪	htcondor.machines_TOO_MANY_D_STATE_PROCS	[avg]	0	0	0.1051	18
🟦	htcondor.machines_SWAP_USAGE_TOO_HIGH	[avg]	1	0	0.1073	14
🟥	htcondor.machines_POOL_DIR_TOO_FULL	[avg]	0	0	0.139	5
🟨	htcondor.machines_IOWAIT_TOO_HIGH	[avg]	0	0	0.615	28
🟩	htcondor.machines_POOL_DIR_TEST_FILE_DELETION	[avg]	0	0	0	0
🟥	htcondor.machines_POOL_DIR_TEST_FILE_CREATION	[avg]	0	0	0	0
⬜	htcondor.machines_POOL_DIR_MISSING	[avg]	0	0	0	0
🟨	htcondor.machines_MARKED_UNHEALTHY_BY_PUPPET	[avg]	0	0	1.2	38
🟦	htcondor.machines_HEALTHCHECK_EXECTIME_TOO_LARGE	[avg]	0	0	0.0025	22
🟪	htcondor.machines_HEALTHCHECK_EXECTIME_NEGATIVE	[avg]	0	0	0	0
🟦	htcondor.machines_CEPHFS_DIR_TEST_FILE_DELETION	[avg]	0	0	0	0
🟩	htcondor.machines_CEPHFS_DIR_TEST_FILE_CREATION	[avg]	0	0	0	0
🟥	htcondor.machines_CEPHFS_DIR_MISSING	[avg]	0	0	0	1

Fig. 15 Number of unhealthy worker nodes as a function of time. The reason why they are unhealthy is colour-coded (see legend)

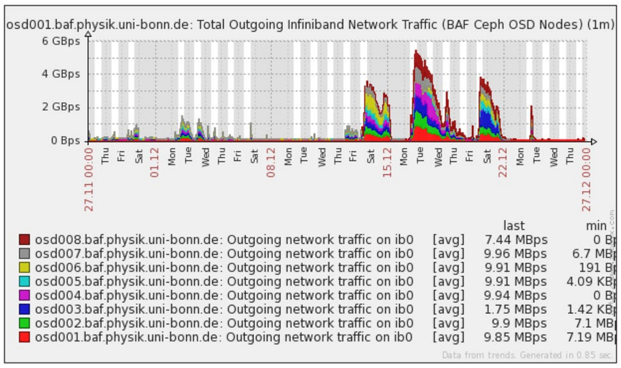


			last	min	avg	max
🟦	htcondor.jobs_container_os_Debian10	[avg]	0	0	0	1
🟪	htcondor.jobs_container_os_Ubuntu1804	[avg]	440	0	441.19	1.98 K
🟩	htcondor.jobs_container_os_SL6	[avg]	8	1	232.89	2.47 K
🟥	htcondor.jobs_container_os_CentOS7	[avg]	9	0	75.52	1.13 K
🟦	htcondor.jobs_container_os_CentOS8	[avg]	0	0	0.0139	1

Fig. 16 Number of chosen container OSes versus time. Debian 10 and CentOS 8 were only added at the end of the displayed interval and are thus hardly used yet

erasure coded objects from the different shards, and is not purely client traffic (see also “CephFS”). Furthermore, it might happen that the 5 GB/s can not be achieved due to the clients performing too many small random reads.

Figure 14 shows the status of worker nodes for a period of three months. There are two noticeable structures in this plot. First, it is apparent that typically every 20–30 days worker nodes are drained for reboots. This time pattern is mostly imposed by the release of security patches requiring reboots. The second interesting item is the health status evolution. Very often, there are at least a few nodes unhealthy. The underlying reasons why nodes are flagged unhealthy are broken down in Fig. 15. There is no prominent single cause but a plethora of possible events, like



			last	min
🟥	osd008.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	7.44 MBps	0 B
⬜	osd007.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	9.96 MBps	6.7 MB
🟨	osd006.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	9.91 MBps	191 B
🟦	osd005.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	9.91 MBps	4.09 KB
🟪	osd004.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	9.94 MBps	0 B
🟦	osd003.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	1.75 MBps	1.42 KB
🟩	osd002.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	9.9 MBps	7.1 MB
🟥	osd001.baf.physik.uni-bonn.de: Outgoing network traffic on ib0	[avg]	9.85 MBps	7.19 MB

Fig. 17 Outgoing traffic from Ceph OSD servers versus time

e.g. CPU I/O wait too high, swap usage too high, job working directories too full, too many D-state processes, etc. Those frequent issues clearly show the need for automatic handling of such events. Improving the situation, again, requires teaching users.

The new freedom to choose the job runtime environment is particularly appreciated since it reduces some of the continuous tension of BAF1. Figure 16 shows the number of chosen container OSes as a function of time. It is obvious that BAF2 users do exploit the variety of offered OSes. The chosen OS is highly correlated with the community the users are associated with. This is no surprise since most communities concentrate on a specific computing platform and provide all needed tools for this environment. Offering a wider range of OSes thus simplifies cross-community usage of computing resources. Other aspects which come into play when choosing the operation system are:

- Is software used whose vendor only supports specific platforms?
- Is an operating system with long-term support needed?
- Does the used software framework rely on new software tools which are not available on older, long-term support platforms?
- Which operating system is used on desktop computers?

The GPU node was added to BAF2 later on while the rest of the cluster was already in production. Unfortunately integrating the GPUs turned out to be cumbersome. The reason is on the one hand that we have to use an HTCondor version from the 8.8 release series to get up-to-date CUDA support. On the other hand, the upstream switch to `nsenter` caused issues with interactive containerized jobs in this release series which have only been partially resolved up to version 8.8.7 (see discussion in “HTCondor”). Fortunately, it was possible to bypass those issues for interactive jobs by using a series of workarounds, notably, a separate script submitted

Table 1 Usage statistics for the CVMFS repositories `software.physik.uni-bonn.de` and `container.physik.uni-bonn.de` on December 31, 2019. The given sizes are uncompressed file sizes

Fully qualified repository name	Size/GB	# files
`software.physik.uni-bonn.de`	248	3,238,748
`container.physik.uni-bonn.de`	950	20,014,623
Total	1198	23,253,371

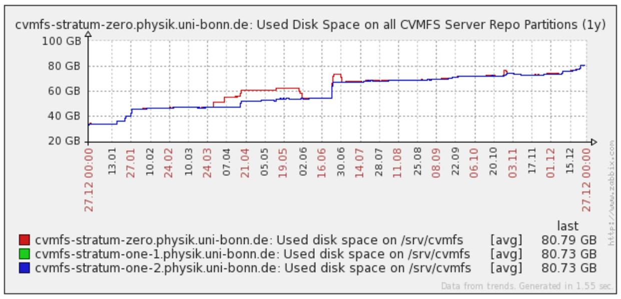


Fig. 18 Used disk space on the CVMFS stratum zero (red) and stratum one servers (green and blue). The temporary discrepancies in space usage are caused by slightly delayed object expiration on the stratum one servers, storage of software packages before deployment on the stratum zero scratch space, and orphaned objects caused by bug temporarily affecting our setup

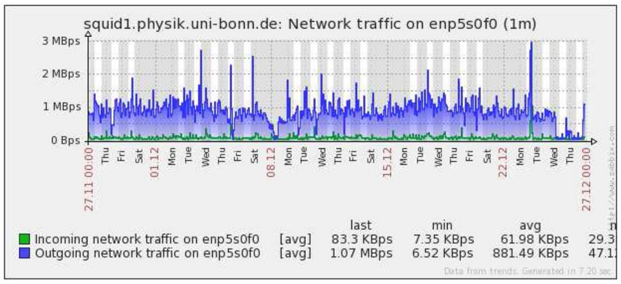


Fig. 19 Network throughput for the first Squid server as function of time. The blue region shows the outgoing traffic, the green region the incoming traffic

as a batch job to which the user connects to emulate an interactive job, a tool to create a pseudoterminal within the job environment and restoration of the shell environment within the job. Hopefully this will become obsolete with one of the next HTCondor releases.

Figure 17 shows the outgoing traffic of all Ceph OSD servers during heavy read I/O load. The network traffic amplification due to erasure coding (see "CephFS") has to be taken into account, but still, the achieved throughput is $\mathcal{O}(5\,\text{GB/s})$ when large files are processed sequentially. It is expected that this bandwidth can be scaled up by increasing the number of OSD servers and optimizing the bandwidth of

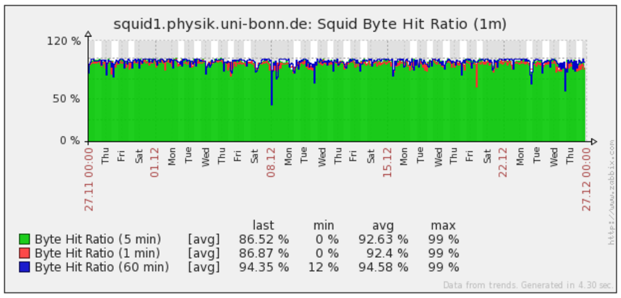


Fig. 20 Cache hit fraction (for cached data in bytes) for the first Squid server as function of time. The colours show different averaging intervals: 1 min (pink), 5 min (green), 60 min (blue)

the utilized IP-over-InfiniBand stack further, but already at this point the cluster file system is not the limiting resource for most of the BAF2 jobs.

The performance of metadata queries / system calls is not monitored yet, since an increased latency is taken for granted by employing the Ceph FUSE client and not limiting throughput for the compute jobs.

Table 1 in combination with Fig. 18 nicely illustrates the power of CVMFS for software and (unpackaged) container image distribution. At the end of the year 2019, we stored in total more than 23 million files summing up to 1198 GB of (uncompressed) software and container images. Thanks to the deduplication and compression features of CVMFS, this requires less than 100 GB of disk space on the CVMFS stratum zero and the stratum one servers. Larger steps in Fig. 18 usually happen if completely new software or container images are added whereas adding new releases or updated images of already available software causes hardly visible increases in disk requirements.

In addition to self-provided and self-maintained software in our local CVMFS repositories, we also provide 24 TB of software from nine CERN-hosted CVMFS repositories. Many of our users crucially rely on software from the latter repositories. Given the high frequency of changes in the used software stack, a conventional, self-maintained software distribution via a local storage system would be unfeasible. The importance of CVMFS for the cluster users is reflected by the traffic on our Squid servers which cache accesses to both local and remote stratum one servers as shown in Fig. 19.

Clearly, the incoming traffic is negligible as compared to the outgoing traffic. The transferred outgoing data rate amounts to $\mathcal{O}(1\,\text{MB/s})$ with occasional spikes being an order of magnitude larger. This can be explained by checking the average HTTP client request rate of $\mathcal{O}(10\,\text{Hz})$ with similar occasional spikes: the CVMFS client of each node checks the catalogues of all repositories every 5 min for updates, and the Squid servers reduce these many randomized requests to a single outgoing request each 5 min.

Furthermore, the Squid servers are also used to cache operating system package updates for all servers and desktop machines, so a fraction of the regular requests is caused by checking for system updates, and by the Frontier system [33] used as distributed database caching system for compute jobs in the HEP community. The latter is the main cause for occasional major spikes in HTTP request rates.

The value of the Squid servers can additionally be seen in Fig. 20 which illustrates that on average, 95% of the amount of data requested from the Squid servers can be served from the cache contents (cache hits).

Conclusion

The implementation and operation of the described BAF2 cluster has come with significant changes both to the choice of technologies and to user workflows as compared to its predecessor, breaking with the conventional concept of login nodes and turning each desktop machine into a submit node. Since it is difficult to simulate the interplay of the individual cluster components under realistic conditions in a test environment, we only present performance measurements from the production era here. Judging from approximately two years of operational experience and user feedback, the chosen approach turned out to be powerful and flexible. The described design allows us to fulfil both our present users' requirements and solve operational challenges in a much better way than conventional concepts could do. It may be considered as a successful model to satisfy the constantly evolving needs of increasingly heterogeneous user groups on a shared computing cluster, at the same time minimizing the operational effort to cover the existing demands and adapt to new requirements.

We hope to overcome some remaining minor issues (e.g. removal of the workarounds for interactive, containerized usage of GPUs, more user-friendly handling of Kerberos ticket renewal) in the near future. In addition, we will soon add the possibility for jobs using a message passing interface (MPI) [103] implementation to use the resources of multiple hosts for highly parallel computing tasks, and we have plans to allow users to run Jupyter [104] jobs on BAF2 resources via JupyterHub [105] and explore novel abstractions such as HTMap [106]. It would also be desirable to extend our monitoring to offer information about correlations.

Compared to its predecessor, BAF2 uses many more standardized, widespread tools integrated into a more modular operating concept. This allows us to perform frequent micro-updates of the deployed software rather than occasional huge changes. As a result, we are closer to the respective development communities and can provide code contributions or valuable feedback on new features with short feedback loops. Last but not least, this community involvement also means more joyful work.

The modular and standardized components enable us to react more quickly to new developments and demands. In addition, BAF2 runs more smoothly with less maintenance work than its predecessor. So in summary, BAF2 is a success story.

Acknowledgements We would like to thank the German Research Foundation (DFG) who provided funding under grant number INST 217/835-1 FUGG to build up the described infrastructure. We are indebted to Anjishnu Bandyopadhyay, Frank Frommberger, Michael Hübner, Katrin Kohl, Helmut Kortmann, Oliver Ricken, Christian Wessel and Andreas Wißkirchen for their help during the commissioning and operation of the described setup. In addition, we are grateful to Jan Stillings and the University Computing Centre (Hochschulrechenzentrum)—in particular the network department, Achim Elsner and Martin Ragg—for their support.

Funding Open Access funding enabled and organized by Projekt DEAL. The hardware used in this study was purchased using a grant by the German Research Foundation (DFG) (grant number INST 217/835-1 FUGG). The personnel was financed by the University of Bonn.

Data Availability Statement All retained data generated and analysed during this study are included in the figures of this published article. Data which go beyond the aggregate information shown in the figures of this article are not retained. Data which allow to infer information about individual users cannot be provided on legal grounds.

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

Code availability Most of the employed software to run the described setup is generally available as open source.

References

1. Albrecht J, Alves AA, Amadio G, Andronico G, Anh-Ky N, Aphecetche L, Apostolakis J, Asai M, Atzori L et al (2019)

A roadmap for HEP software and computing R&D for the 2020s. Comput Softw Big Sci 3:1. https://doi.org/10.1007/s41781-018-0018-8
2. Huerta EA, Haas R, Jha S, Neubauer M, Katz DS (2019) Supporting high-performance and high-throughput computing for experimental science. Comput Softw Big Sci 3:1. https://doi.org/10.1007/s41781-019-0022-7
3. TORQUE/Maui. http://adaptivecomputing.com/cherry-services/torque-resource-manager. Accessed 20 Jan 2020
4. Lustre. http://lustre.org. Accessed 20 Jan 2020
5. OpenAFS. https://www.openafs.org. Accessed 20 Jan 2020
6. CVMFS. https://cernvm.cern.ch/portal/filesystem. Accessed 20 Jan 2020
7. Bird I, Buncic P, Carminati F, Cattaneo M, Clarke P, Fisk I, Girone M, Harvey J, Kersevan B, Mato P, Mount R, Panzer-Steindel B (2014) Update of the computing models of the WLCG and the LHC experiments. Technical Report. CERN-LHCC-2014-014. LCG-TDR-002. https://cds.cern.ch/record/1695401. Accessed 20 Jan 2020
8. Buncic P, Sanchez C Aguado, Blomer J, Franco L, Harutyunian A, Mato P, Yao Y (2010) J Phys Conf Ser 219: 042003. https://doi.org/10.1088/1742-6596/219/4/042003. https://cds.cern.ch/record/1269671. Accessed 20 Jan 2020
9. Blomer J, Fuhrmann T (2010) In: 2010 Proceedings of the international conference on computer communications and networks (ICCCN) (IEEE, 2010). https://ieeexplore.ieee.org/document/5560054. Accessed 20 Jan 2020
10. Dykstra D, Bockelman B, Blomer J, Herner K, Levshina T, Slyz M (2015) Engineering the CernVM-filesystem as a high bandwidth distributed filesystem for auxiliary physics data. J Phys Conf Ser 664:7. https://doi.org/10.1088/1742-6596/664/4/042012
11. Scientific Linux. https://www.scientificlinux.org/. Accessed 20 Jan 2020
12. The CentOS Project. https://www.centos.org/. Accessed 20 Jan 2020
13. Priedhorsky R, Randles T (2017) Linux containers for fun and profit in HPC. https://www.usenix.org/system/files/login/articles/login_fall17_03_priedhorsky.pdf. Accessed 20 Jan 2020
14. HTCondor. https://research.cs.wisc.edu/htcondor. Accessed 20 Jan 2020
15. Litzkow M (1987) Remote Unix-turning idle workstations into cycle servers. In: Proceedings of usenix summer conference, pp 381–384. https://research.cs.wisc.edu/htcondor/doc/remoteunix.pdf
16. Litzkow M, Livny M, Mutka MW (1988) Condor — a hunter of idle workstations. In: Proceedings of the 8th international conference of distributed computing systems, pp 104–111. https://research.cs.wisc.edu/htcondor/doc/condor-hunter.pdf
17. Epema D, Livny M, van Dantzig R, Evers X, Pruyne J (1996) A worldwide flock of condors: Load sharing among workstation clusters. Future Gener Comput Syst 12:53
18. Livny M, Basney J, Raman R, Tannenbaum T (1997) Mechanisms for high throughput computing, SPEEDUP 11. https://research.cs.wisc.edu/htcondor/doc/htc_mech.pdf
19. Basney J, Livny M (1999) High performance cluster computing: architectures and systems.In: Buyya R (ed) Prentice Hall PTR, vol 1, ISBN-13: 978-0130137845. https://research.cs.wisc.edu/htcondor/doc/hpcc-chapter.pdf
20. Tannenbaum T, Wright D, Miller K, Livny M (2001) In: Sterling T (ed) Beowulf cluster computing with Linux, MIT Press, ISBN-13: 978-0262692748. https://research.cs.wisc.edu/htcondor/doc/beowulf-chapter-rev1.pdf
21. Thain D, Tannenbaum T, Livny M (2005) Distributed computing in practice: the condor experience. Concurr Pract Exp 17(2–4):323. https://doi.org/10.1002/cpe.938
22. Garonne V, Graeme A, Lassnig M, Molfetas A, Barisits M, Beermann T, Nairz A, Goossens L, Megino F Barreiro, Serfon C, Oleynik D, Petrosyan A (2012) The ATLAS distributed data management project: past and future. Technical Report. ATL-SOFT-PROC-2012-049, CERN, Geneva. https://cds.cern.ch/record/1455298. Accessed 20 Jan 2020
23. Rucio scientific data management. https://rucio.cern.ch. Accessed 20 Jan 2020
24. Furano F, Hanushevsky A (2009) Scalla/xrootd WAN globalization tools: where we are. Technical report. CERN-IT-Note-2009-003, CERN, Geneva. https://doi.org/10.1088/1742-6596/219/7/072005, https://cds.cern.ch/record/1177151. Accessed 20 Jan 2020
25. Ceph. https://ceph.io. Accessed 20 Jan 2020
26. Foreman. https://theforeman.org. Accessed 20 Jan 2020
27. Puppet. https://puppet.com. Accessed 20 Jan 2020
28. RFC 7862: Network file system (NFS) version 4 minor version 2 protocol. https://tools.ietf.org/html/rfc7862. Accessed 20 Jan 2020
29. Zabbix. https://www.zabbix.com. Accessed 20 Jan 2020
30. AUFS. http://aufs.sourceforge.net. Accessed 20 Jan 2020
31. OverlayFS. https://www.kernel.org/doc/html/latest/filesystems/overlayfs.html. Accessed 20 Jan 2020
32. Squid. http://www.squid-cache.org. Accessed 20 Jan 2020
33. Frontier distributed database caching system. http://frontier.cern.ch. Accessed 20 Jan 2020
34. Spack. https://spack.io. Accessed 20 Jan 2020
35. EasyBuild documentation. https://easybuild.readthedocs.io. Accessed 20 Jan 2020
36. Lmod: a new environment module system. https://lmod.readthedocs.io. Accessed 20 Jan 2020
37. Lua. https://www.lua.org. Accessed 20 Jan 2020
38. CernVM-FS Shrinkwrap utility. https://cvmfs.readthedocs.io/en/stable/cpt-shrinkwrap.html. Accessed 20 Jan 2020
39. VMware. https://www.vmware.com. Accessed 20 Jan 2020
40. Xen. https://xenproject.org. Accessed 20 Jan 2020
41. KVM. http://www.linux-kvm.org. Accessed 20 Jan 2020
42. VirtualBox. https://www.virtualbox.org. Accessed 20 Jan 2020
43. Paravirtualization. https://wiki.xen.org/wiki/Paravirtualization_(PV). Accessed 20 Jan 2020
44. Namespaces in operation. https://lwn.net/Articles/531114. Accessed 20 Jan 2020
45. cgroups. https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v1/cgroups.html. Accessed 20 Jan 2020
46. Jails. https://www.freebsd.org/doc/handbook/jails.html. Accessed 20 Jan 2020
47. OpenVZ. https://openvz.org. Accessed 20 Jan 2020
48. Solaris containers. https://oracle.com/solaris. Accessed 20 Jan 2020
49. LXC. http://linuxcontainers.org. Accessed 20 Jan 2020
50. Docker. https://www.docker.com. Accessed 20 Jan 2020
51. CVE-2014-3499. https://nvd.nist.gov/vuln/detail/CVE-2014-3499. Accessed 20 Jan 2020
52. CVE-2014-9357. https://nvd.nist.gov/vuln/detail/CVE-2014-9357. Accessed 20 Jan 2020
53. CVE-2019-5736. https://nvd.nist.gov/vuln/detail/CVE-2019-5736. Accessed 20 Jan 2020
54. M. Kerrisk. Namespaces in operation, part 5: user namespaces. https://lwn.net/Articles/532593. Accessed 20 Jan 2020
55. Singularity. https://sylabs.io. Accessed 20 Jan 2020
56. Charliecloud. https://hpc.github.io/charliecloud. Accessed 20 Jan 2020
57. Priedhorsky R, Randles T (2017) In: SC '17: Proceedings of the international conference for high performance computing, networking, storage and analysis, Association for computing machinery, New York. https://doi.org/10.1145/3126908.3126925

58. runC. https://www.docker.com/blog/runc. Accessed 20 Jan 2020
59. Podman. https://podman.io. Accessed 20 Jan 2020
60. Corbet J (2018) Unprivileged filesystem mounts, 2018 edition. https://lwn.net/Articles/755593. Accessed 20 Jan 2020
61. CVE-2016-10208. https://nvd.nist.gov/vuln/detail/CVE-2016-10208. Accessed 20 Jan 2020
62. Open container initiative. https://www.opencontainers.org. Accessed 20 Jan 2020
63. Linux Foundation. https://www.linuxfoundation.org. Accessed 20 Jan 2020
64. Docker Hub. https://hub.docker.com. Accessed 20 Jan 2020
65. Weil SA, Brandt SA, Miller EL (2006) Maltzahn C (2006) in SC '06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. Association for Computing Machinery, New York, NY, USA. http://www.ssrc.ucsc.edu/Papers/weil-sc06.pdf. Accessed 20 Jan 2020
66. Ceph Placement Groups Documentation. https://docs.ceph.com/docs/master/rados/operations/placement-groups/#choosing-the-number-of-placement-groups. Accessed 20 Jan 2020
67. Fielding RT (2000) Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine, USA. https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Accessed 20 Jan 2020
68. Snappy, a fast compressor/decompressor. https://github.com/google/snappy. Accessed 20 Jan 2020
69. New in Luminous: BlueStore. https://ceph.io/community/new-luminous-bluestore/. Accessed 20 Jan 2020
70. NVM Express. https://nvmexpress.org. Accessed 20 Jan 2020
71. RocksDB: A persistent key-value store for fast storage environments. https://rocksdb.org. Accessed 20 Jan 2020
72. RDMA Consortium. http://www.rdmaconsortium.org. Accessed 20 Jan 2020
73. Mellanox OpenFabrics Enterprise Distribution for Linux. https://www.mellanox.com/products/infiniband-drivers/linux/mlnx_ofed. Accessed 20 Jan 2020
74. ROOT Data Analysis Framework. https://root.cern.ch. Accessed 20 Jan 2020
75. Yang W, Hanushevsky A, Ito H, Lassnig M, Popescu R, De Silva A, Simon MK, Gardner R, Garonne V, Destefano J, Vukotic I (2018) Xcache in the ATLAS distributed computing environment. Technical Report. ATL-SOFT-PROC-2018-031, CERN, Geneva. https://doi.org/10.1051/epjconf/201921404008. https://cds.cern.ch/record/2648892. Accessed 20 Jan 2020
76. Bos K, Brook N, Duellmann D, Eck C, Fisk I, Foster D, Gibbard B, Grandi C, Grey F, Harvey J, Heiss A, Hemmer F, Jarp S, Jones R, Kelsey D, Knobloch J, Lamanna M, Marten H, Mato Vila P, Ould-Saada F, Panzer-Steindel B, Perini L, Robertson L, Schutz Y, Schwickerath U, Shiers J, Wenaus T (2005) LHC computing grid: technical design report. Version 1.06. Technical Design Report LCG (CERN, Geneva, 2005). https://cds.cern.ch/record/840543. Accessed 20 Jan 2020
77. WebDAV Resources. http://www.webdav.org. Accessed 20 Jan 2020
78. Virtual Organization Membership Service. https://italiangrid.github.io/voms. Accessed 20 Jan 2020
79. Third Party Copy. https://twiki.cern.ch/twiki/bin/view/LCG/ThirdPartyCopy. Accessed 20 Jan 2020
80. File transfer service. https://fts.web.cern.ch. Accessed 20 Jan 2020
81. Birgisson A, Politz J Gibbs, Erlingsson U, Taly A, Vrable M, Lentczner M (2014) In: NDSS '14: Proceedings of the 2014 network and distributed system security (NDSS) symposium (Internet Security, 2014). https://research.google.com/pubs/archive/41892.pdf. Accessed 20 Jan 2020
82. SciTokens: federated authorization for distributed scientific computing. https://scitokens.org. Accessed 20 Jan 2020
83. Withers A, Bockelman B, Weitzel D, Brown DA, Gaynor J, Basney J, Tannenbaum T, Miller Z (2018) CoRR. Accessed 20 Jan 2020
84. Derek W, Brian B, Basney J, Todd T, Zach M, Jeff G (2019) In: EPJ web conference 214:04014. https://doi.org/10.1051/epjconf/201921404014. Accessed 20 Jan 2020
85. Withers A, Bockelman B, Weitzel D, Brown DA, Patton J, Gaynor J, Basney J, Tannenbaum T, Gao YA, Miller Z (2019) CoRR
86. Altunay M, Bockelman B, Ceccanti A, Cornwall L, Crawford M, Crooks D, Dack T, Dykstra D, Groep D, Igoumenos I, Jouvin M, Keeble O, Kelsey D, Lassnig M, Liampotis N, Litmaath M, McNab A, Millar P, Sallé M, Short H, Teheran J, Wartel R (2019) WLCG Common JWT Profiles. https://doi.org/10.5281/zenodo.3460258
87. Puppet Forge. https://forge.puppet.com. Accessed 20 Jan 2020
88. r10k. https://github.com/puppetlabs/r10k. Accessed 20 Jan 2020
89. libvirt. https://libvirt.org. Accessed 20 Jan 2020
90. OpenStack. https://www.openstack.org. Accessed 20 Jan 2020
91. oVirt. https://www.ovirt.org. Accessed 20 Jan 2020
92. Amazon Elastic Compute Cloud. https://aws.amazon.com/ec2. Accessed 20 Jan 2020
93. Google Compute Engine. https://cloud.google.com/compute. Accessed 20 Jan 2020
94. Foreman plugins. https://projects.theforeman.org/projects/foreman/wiki/List_of_Plugins. Accessed 20 Jan 2020
95. Hammer—the CLI tool (not only) for Foreman. https://github.com/theforeman/hammer-cli. Accessed 20 Jan 2020
96. Dunn C. Designing Puppet — Roles and Profiles. https://www.craigdunn.org/2012/05/239. Accessed 20 Jan 2020
97. The roles and profiles method. https://puppet.com/docs/pe/2018.1/the_roles_and_profiles_method.html. Accessed 20 Jan 2020
98. Slurm Workload Manager. https://slurm.schedmd.com. Accessed 20 Jan 2020
99. Yoo AB, Jette MA, Grondona M (2003) SLURM: Simple linux utility for resource management. In: Feitelson D, Rudolph L, Schwiegelshohn U (eds) Job Scheduling Strategies for Parallel Processing. JSSPP 2003. Lecture Notes in Computer Science, vol 2862. Springer, Berlin, Heidelberg. https://doi.org/10.1007/10968987_3
100. Raman R, Livny M, Solomon M (1998) Matchmaking: distributed resource management for high throughput computing. In: Proceedings of the seventh IEEE international symposium on high performance distributed computing (HPDC7), 98, Chicago, Illinois,USA, pp 140–146, IEEE Computer Society. https://doi.org/10.1109/HPDC.1998.709966
101. zswap. https://www.kernel.org/doc/html/latest/vm/zswap.html. Accessed 20 Jan 2020
102. SELinux. https://github.com/SELinuxProject. Accessed 20 Jan 2020
103. MPI Forum. https://www.mpi-forum.org. Accessed 20 Jan 2020
104. Project Jupyter. https://jupyter.org. Accessed 20 Jan 2020
105. JupyterHub. https://jupyter.org/hub. Accessed 20 Jan 2020
106. HTMap. https://htmap.readthedocs.io. Accessed 20 Jan 2020