Computing and Software for Big Science (2023) 7:7
https://doi.org/10.1007/s41781-023-00100-1

RESEARCH

Lightweight Integration of a Data Cache for Opportunistic Usage of HPC Resources in HEP Workflows

Dirk Sammel[1] · Michael Boehler[1] · Anton J. Gamel[1,2] · Markus Schumacher[1]

Received: 15 February 2023 / Accepted: 30 June 2023

Abstract
A data caching setup has been implemented for the High Energy Physics (HEP) computing infrastructure in Freiburg, Germany, as a possible alternative to local long-term storage. Files are automatically cached on disk upon first request by a client, can be accessed from cache for subsequent requests, and are deleted after predefined conditions are met. The required components are provided to a dedicated HEP cluster, and, via virtual research environments, to the opportunistically used High-Performance Computing (HPC) Cluster NEMO (Neuroscience, Elementary Particle Physics, Microsystems Engineering and Materials Science). A typical HEP workflow has been implemented as benchmark test to identify any overhead introduced by the caching setup with respect to direct, non-cached data access, and to compare the performance of cached and non-cached access to several external files sources. The results indicate no significant overhead in the workflow and faster file access with the caching setup, especially for geographically distant file sources. Additionally, the hardware requirements for various numbers of parallel file requests were measured for estimating future requirements.

Keywords Caching · Particle physics · Benchmarks · Data analysis · HPC

Introduction

With the start of the High-Luminosity Large Hadron Collider (HL-LHC) [7] in the near future, the amount of data collected by the LHC experiments will increase significantly and reach, e.g., for the ATLAS experiment, at least 1 Exabyte in 2030 [3]. Distribution, storage, and processing of the data are important tasks of the Worldwide LHC Computing Grid (WLCG) [5]. The sites that are part of the WLCG are structured in a tiered hierarchy. The single Tier-0 site is located at CERN and is used for the prompt reconstruction of data. Fifteen Tier-1 sites provide long-term storage on tape drives and serve as local hubs for the distribution of the data to the ∼ 150 Tier-2 sites, which are used for storage, analysis, and simulation. Finally, Tier-3 sites are connected to the WLCG, but provide their resources only to local users.

In the current organization of the WLCG, the Tier-2 sites provide both storage- and computing resources. An alternative approach taken into consideration, the "data lake model" [14], would consist of data centers on the one hand and pure computing sites on the other hand.

As part of the WLCG, the University of Freiburg currently manages storage- and computing resources at Tier-2 level, providing 84 nodes with 3360 CPU cores, and 3.5 Petabyte of dCache [9] storage. The users of four local High Energy Physics (HEP) groups can use part of this storage to store data needed for their analyses. In addition, institutional storage- and computing resources at Tier-3 level are provided by the local High-Performance Computing (HPC) Cluster NEMO (Neuroscience, Elementary Particle Physics, Microsystems Engineering and Materials Science)[1] [19]. NEMO provides 768 Terabyte of storage and 900 nodes with 18000 CPU cores. Without the WLCG related storage and with the increase in data storage requirements after the

✉ Dirk Sammel
dirk.sammel@physik.uni-freiburg.de

Michael Boehler
michael.boehler@physik.uni-freiburg.de

Anton J. Gamel
anton.gamel@physik.uni-freiburg.de

Markus Schumacher
markus.schumacher@physik.uni-freiburg.de

[1] Physikalisches Institut, Albert-Ludwigs-Universität Freiburg, Hermann-Herder-Str. 3, 79104 Freiburg, Germany

[2] Rechenzentrum, Albert-Ludwigs-Universität Freiburg, Hermann-Herder-Str. 10, 79104 Freiburg, Germany

[1] https://www.nemo.uni-freiburg.de/.

Published online: 05 July 2023

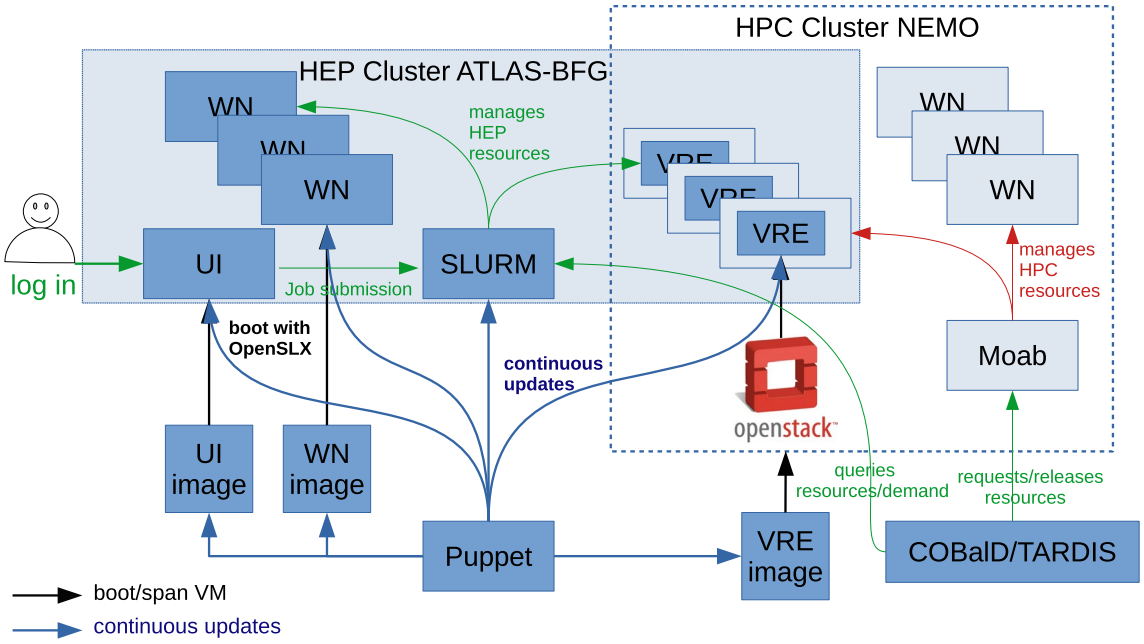

Fig. 1 Visualization of the computing environment in Freiburg

start of the HL-LHC, the institutional storage will likely be not sufficient. In the data lake model, users would need to perform their analyses using the data centers as file source, which might result in longer completion times depending on the geographical distance and/or network connection.

An alternative to this scenario is the implementation of a local disk caching setup. When a user requests data from a data center, it is downloaded in parallel to a local cache space. From there, it can be accessed for subsequent requests, reducing the network load and the latency. Due to the limited storage capacities, the data can be automatically deleted after predefined conditions are met.

There are already existing solutions and studies regarding data caching in the HEP community. For Tier-2 sites, A-REX Data Cache[2] and XCache [12] are available. A disk caching solution for local users at Tier-3 level was investigated under the name Disk-Caching-on-the-Fly.[3] Here we present a setup which can be deployed with minimal additional infrastructure and therefore very cost-efficiently.

This paper first gives a general overview of the computing infrastructure in Freiburg and the implementation of virtual research environments (VREs). After that, the implementation of an XRootD [8] disk caching setup in this environment is described. The performance was tested with benchmarks that use typical features of a HEP analysis.

Virtual Research Environments in Freiburg

A sketch of the components of the compute clusters in Freiburg is shown in Fig. 1. The relevant components of both clusters, the HEP cluster and the HPC cluster, are indicated. The HEP cluster (ATLAS-BFG) is integrated into the WLCG as Tier-2 computing facility. Both ATLAS production- and analysis jobs, as well as the jobs of local HEP users, are executed on the worker nodes (WN). The HPC cluster (NEMO) is explicitly dedicated to users in the state of Baden-Württemberg, including the local users in Freiburg.

The SLURM scheduler [20] is used to send jobs to the WNs of the HEP cluster, while the Moab scheduler[4] [1] is used to send jobs to the bare metal WNs of the HPC cluster. To submit jobs via SLURM, the users log in to a machine of the HEP cluster that serves as User Interface (UI). Although the HEP users can submit their jobs via Moab to the bare metal WNs, an advantageous option is to use SLURM to send their jobs to a VRE running on the WNs. These VREs offer, in contrast to the bare metal machines, a tailor-made setup for analyses. This setup is identical to the setup of the HEP cluster WNs.

COBalD/TARDIS (C/T) [10, 11] manages the integration of HPC resources into the HEP cluster as VREs by communicating with the schedulers: the demand is obtained from SLURM and new resources are requested and monitored via

[2] https://www.nordugrid.org/arc/arc6/tech/data/arex_cache.html.

[3] https://git.gsi.de/atay/xrootd-disk-caching-on-the-fly/.

[4] https://adaptivecomputing.com/moab-hpc-suite/

Moab, which allocates WNs from the HPC cluster. VREs are started on these allocated WNs as virtual machines via an OpenStack[5] instance. Once a VRE is operational, the resource appears in SLURM and can be utilized. C/T also instructs Moab to release resources when they are no longer required. Integration and release of resources are performed based on demand on the HEP cluster and availability on the HPC cluster.

The images for the VREs, UIs, and WNs are all built with Packer,[6] and have a large overlap in their configuration. Therefore, they are pre-configured by Puppet.[7] OpenSLX[8] is used to boot the UI- and WN-images, while the VRE images are booted with OpenStack. All images, the UIs, the WNs, the SLURM server, and the VREs are kept up-to-date by Puppet.

An easy integration of the caching setup into the existing computing environment was required. The setup described in the following section fulfills this requirement.

Caching Setup

The caching setup consists of three components: the client, the proxy server, and the cache space. If the client requests a file from an external site with the XRootD protocol, the request is forwarded to the proxy server. Such a file request can be, e.g., the request to copy a file with the `xrdcp` shell command or the request to open a file in ROOT[9] [6] using the `TFile()` class. The proxy server first checks if the requested file is already completely available in the cache space. If the file is found, the proxy server points the request to the location of the file in the cache space. If the file is not found, the request of the client is forwarded to the external site. While the client is accessing the file from the external site, the proxy server starts to download the file in parallel to the cache space. The download is only active while the file is accessed by a client: if the access terminates before the file is fully downloaded, the download is aborted, but the partial file remains in the cache space. The download is continued if the file is requested again. The caching setup is suitable for a multi-user environment: every client that requests the same file from the same external site is pointed to the file in the cache space by the proxy server. This workflow is shown in Fig. 2.

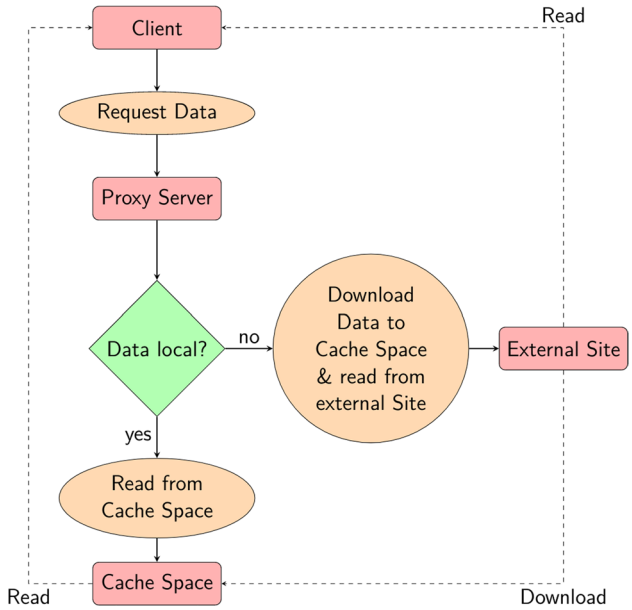

Fig. 2 Sketch of the caching workflow

Details about the setups of the different components and the used machines are described in the following subsections. The configurations of the machines where the client and the proxy server are running were deployed with Puppet.

This setup is not meant to be an optimized caching solution. The aim was to demonstrate that decent results can already be achieved by setting up a caching instance on an existing system landscape. The setup most likely would benefit from dedicated hardware, e.g., SSD storage for the cache space (due to the better latency, throughput, and input/output operations per second with respect to HDD storage [16]) or more RAM for the proxy server (especially for high numbers of parallel file requests). However, the use and optimization of additional hardware was out of scope for this study since we wanted to concentrate on commodity hardware which can be found in any data center.

Client

A virtual machine managed with OpenStack and running on a node of the NEMO cluster is used for the client. The machine has one virtual core of an Intel Xeon CPU E5-2630 v4 @ 2.20 GHz[10] and 2 GB of RAM. The network connection of the virtual machine is 1 Gb/s. CentOS Linux 7.8.2003[11] is used as operating system. Due to the nature

[5] https://www.openstack.org/

[6] https://www.packer.io/.

[7] https://puppet.com/.

[8] https://openslx.com/.

[9] ROOT is a data analysis framework which is commonly used in HEP.

[10] https://www.intel.com/content/www/us/en/products/sku/92981/intel-xeon-processor-e52630-v4-25m-cache-2-20-ghz/specifications.html

[11] https://wiki.centos.org/Manuals/ReleaseNotes/CentOS7.2003..

of the OpenStack environment, the resources are not exclusively reserved for the client, but are shared with other virtual machines.

The client package of XRootD 4.12.3[12] was used on the client. The configuration of the caching setup is done by environment variables. To enable the forwarding of the file request to the proxy server, the environment variable XRD_PLUGIN is set to the path of the XrdClProxyPlugin. With this plugin, the target URL of any file request is replaced by the URL stored in the environment variable XROOT_PROXY, which is set to the IP address of the proxy server. A X.509 certificate is required to authenticate against the proxy server. The VOMS[13] [2] package is used to create the X.509 certificate from a valid WLCG user certificate. The proxy server can restrict access to certain virtual organizations (VO). In that case, the user certificate and the X.509 certificate have to be associated with the respective VO. This guarantees that only users with appropriate permissions can trigger the download of files.

For the measurements of parallel file requests, a bare metal node of the NEMO cluster is used. This machine is equipped with 2 x AMD EPYC 7742 @ 2.25 GHz[14] with a total of 128 cores, and 512 GB of RAM. The resources of the bare metal node are exclusively used by the client.

Proxy Server

For the proxy server, VMware ESXi[15] [18] is used to configure a virtual machine with four virtual cores of an Intel Xeon CPU E5-2640 v3 @ 2.60 GHz[16] and 8 GB of RAM. The operating system is CentOS Linux 7.9.2009.[17] The proxy server has a network connection of 1 Gb/s. As in the case of OpenStack, the resources of the proxy server are shared with other virtual machines in the VMware ESXi environment.

The server package of XRootD 5.2.0[18] was used on the proxy server.[19] The XRootD server daemon is running and configured to act as a forwarding proxy and to use the disk caching features of XRootD. The default values of the caching parameters have been used, e.g., a *blocksize* of 1 MB and a *prefetch* of up to 10 blocks. Automatic deletion of the data in the cache space was disabled since this feature was not relevant for the benchmarks.

As a forwarding proxy, the server forwards the file request by the client to the external site or, if the file already exists in the cache space, points the client to the respective path. If the requested file is not yet completely present in the cache space, the proxy server downloads the file. To authenticate against the external site, the proxy server needs a valid X.509 certificate which is associated to the respective VO of the external site. The VOMS package is used to create a X.509 certificate from a WLCG host certificate. The X.509 certificate is created for the service account `xrootd` that is running the XRootD server daemon. To acquire a proper host certificate, the proxy server needs a public IP address. Since this was not possible with the OpenStack setup in Freiburg, the virtual machine was deployed with VMware ESXi.

Cache Space

The cache space is the location, e.g., a certain disk or directory, where the cached files are stored. Both the client and the proxy server need access to the cache space. The client needs read permissions, whereas the proxy server needs read- and write permissions. Any type of distributed file system can be used.

For the setup used in this study, a workspace[20] on the NEMO storage, which uses the file system BeeGFS[21] [15], was created and the necessary permissions were set. The network connection between the cache space and sites outside of the university network of Freiburg is 20 Gb/s. Each NEMO user has a storage quota of 10 Terabyte, but no such quota exists for individual workspaces. Therefore, the cache space could in principle use the total storage of 768 Terabyte provided by NEMO, if available.

Benchmark Setup

Benchmarks were performed to measure the performance of the caching setup and the resource requirements of the proxy server.

For the *workflow* benchmarks, the required time to complete the benchmarks is measured. This is done for different scenarios, including the default setup without caching.

For the benchmarks of the *proxy server*, the resource consumption of the proxy server is measured when multiple client requests have to be processed in parallel. Several

[12] https://xrootd.slac.stanford.edu/2020/06/11/announcement_4_12_3.html.

[13] https://italiangrid.github.io/voms/..

[14] https://www.amd.com/en/products/cpu/amd-epyc-7742.

[15] https://www.vmware.com/products/esxi-and-esx.html.

[16] https://www.intel.com/content/www/us/en/products/sku/83359/intel-xeon-processor-e52640-v3-20m-cache-2-60-ghz/specifications.html.

[17] https://wiki.centos.org/Manuals/ReleaseNotes/CentOS7.2009.

[18] https://xrootd.slac.stanford.edu/2021/05/20/announcement_5_2_0.html.

[19] The HEP framework on the client machine did not provide version 5.2.0 of XRootD, so an older version had to be used there.

[20] https://github.com/holgerBerger/hpc-workspace.

[21] https://www.beegfs.io/c/.

thousand client requests are not uncommon in a production environment, and hence it is important to know the expected resource consumption under such circumstances. Therefore, the results of the measurements of the proxy server were used to extrapolate the required resources for larger numbers of parallel requests.

Python 3.8.6[22] and the ROOT module for Python (PyROOT[23]) with ROOT version 6.22.06[24] are used for these benchmarks.

Input Files

The input files for the benchmarks were created in the ROOT format with the event generator Pythia 8.303[25] [4]. The files contain information about simulated $t\bar{t}$ events[26] from proton–proton collisions at a center-of-mass energy of 13 TeV. For each event, the number of particles in the event and several properties of the particles are stored in the files: the particle type, the particle status,[27] the energy, the mass, the transverse momentum, the azimuthal angle, and the pseudorapidity. The mean size of this information is about 26 kB per event, and the mean number of particles per event is about 1600.

Three files with different numbers of events, and therefore different file sizes, were generated: a small file with 50k events (1.3 GB), a medium file with 200k events (4.9 GB), and a large file with 500k events (13 GB).

Benchmark Description

Typical operations of a HEP analysis were performed in the benchmark: the ROOT Python module was used to open the requested file with the `TFile()` class. After that, the data in the file were loaded and a loop over the events was executed. For each event, a second loop over the respective number of particles of the event was executed. The pseudorapidity of each particle was filled into an histogram. In addition, the transverse momentum, the pseudorapidity, the azimuth angle, and the mass of each particle originating from the decay of the $t\bar{t}$ system was used to build a LorentzVector,[28] from which the mass of the $t\bar{t}$ system was reconstructed and filled into an histogram.

The input files, the code to create them, and the benchmark code are publicly available.[29]

Table 1 Measurements of the round-trip time (RTT) and their standard deviations for the external sites. The results are rounded to significant digits

Site	RTT [ms]
dCache FR	0.38 ± 0.10
KIT	2.80 ± 0.13
LRZ	10.3 ± 0.6
DESY	20.8 ± 3.4
BNL	89.5 ± 0.4
TRIUMF	157.15 ± 0.30

Workflow Benchmarks

Several input parameters were varied to test the respective dependency of the performance of the caching setup. All three file sizes were used as input, and the number of processed events was varied: 1, 100, 1000, and 50k events for all three files, 200k events for the medium and the large file, and 500k events for the large file. The files were distributed to several WLCG sites that provide resources to the ATLAS collaboration to determine the influence of the geographical distance to the external site from which the files were requested. The sites used for the benchmarks were KIT (Karlsruhe, Germany), LRZ (Munich, Germany), DESY (Hamburg, Germany), TRIUMF (Vancouver, Canada), BNL (Brookhaven, USA), and the WLCG infrastructure in Freiburg, denoted as "dCache Freiburg". Measurements of the round-trip time (RTT) using the `ping` command have been performed in order to estimate latencies due to the geographical distance and the individual network connections, and are shown in Table 1. As expected, more distant sites have a higher RTT. In addition, direct access to the files stored on the BeeGFS was tested.

The benchmark was performed for all combinations of file size, number of events, and external site. To reduce statistical uncertainties and the impact of external factors like network- or I/O load, every benchmark was repeated several times.

Proxy Server Benchmarks

For the measurements of the resource requirements of the proxy server, the above-mentioned parameters were fixed to the large file, 10k events (to ensure an overlap in the processing of the parallel requests), and KIT (to ensure a fast connection). Instead, the number of parallel requests that had

[22] https://www.python.org/downloads/release/python-386/.

[23] https://root.cern/manual/python/.

[24] https://root.cern/releases/release-62206/.

[25] https://pythia.org/manuals/pythia8303/Welcome.html.

[26] The collision of particles and the induced production of new particles is called an event.

[27] Pythia sets the status of a particle according to how it was produced and if it is still present at the end of the collision.

[28] https://root.cern.ch/doc/master/classROOT_1_1Math_1_1LorentzVector.html.

[29] https://github.com/ALU-Schumacher/caching_benchmarks.

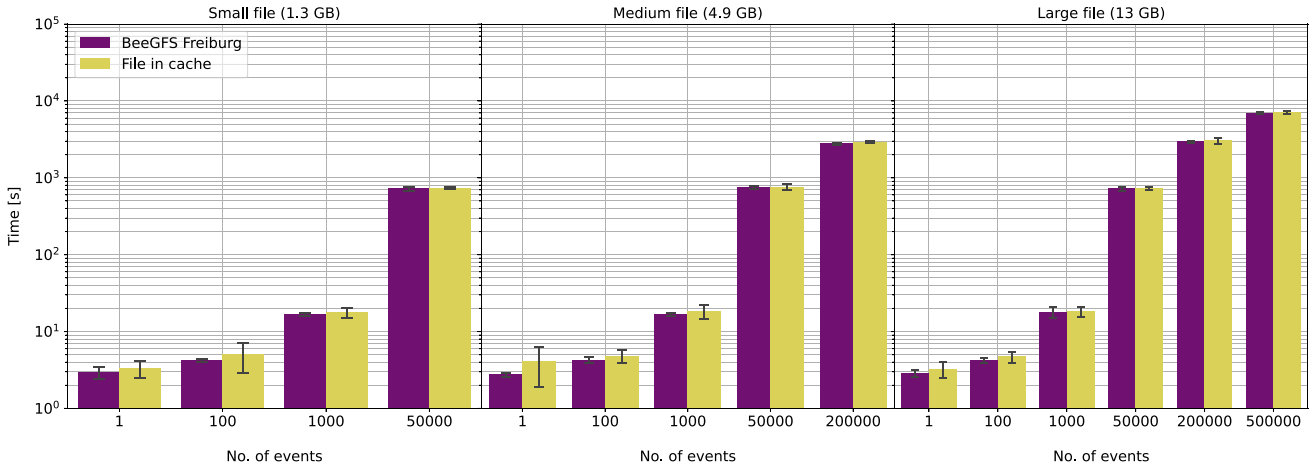

Fig. 3 Workflow benchmark results for BeeGFS Freiburg with (yellow) and without (purple) caching setup for different numbers of events, for the small, medium, and large files (from left to right)

to be processed by the proxy server was varied. The tested numbers were 25, 50, 75, 100, and 128. To avoid an impact on the results by parallel reading of a single file, 128 copies of the large file were placed at the KIT storage.

The memory consumption and the CPU load of the `xrootd` service were measured on the proxy server during the complete run of the benchmark.[30]

The measurements for each number of parallel requests were repeated several times to reduce the statistical uncertainties and the impact of external factors like network- or I/O load.

Results and Discussion

For the workflow benchmarks, the mean value and standard deviation (SD) of the measurements of the elapsed time were calculated, respectively, for all combinations of the input parameters. For the proxy server benchmarks, the mean value and standard deviation of the measurements of the CPU load and the memory consumption were calculated, respectively, for all numbers of parallel requests.

Workflow Results

The results of the workflow benchmarks for all sites without and with caching setup are shown in Table 2 and Table 3, respectively. Preliminary results of this study have been previously published [17].

Fig. 3 shows the workflow benchmark results for the caching setup with a hot cache, i.e., when the files are already available in the cache space on the BeeGFS, and for accessing the files directly on the BeeGFS in Freiburg without using the caching setup. This comparison is a test for potential overhead introduced by the caching setup, since the files are stored on the BeeGFS in Freiburg in both cases. For each file size and each number of events, the results are comparable. Therefore, no significant overhead by the caching setup is observed. While the completion time of the benchmark is larger for increasing numbers of events, it is independent of the file size. This allowed to merge the measurements for the three file sizes in the subsequent figures and tables.

The comparison between direct file access on the different sites and accessing the file in the cache space when using

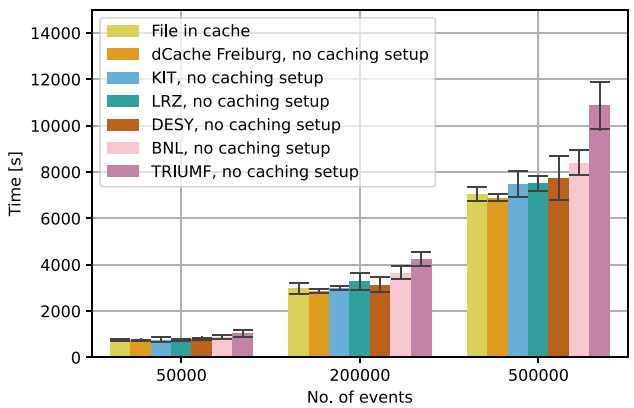

Fig. 4 Workflow benchmark results for the caching setup, from left to right: when the file is already available in the cache space (yellow), for access without caching setup for dCache Freiburg (dark yellow), KIT (blue), LRZ (green), DESY (brown), BNL (pink), and TRIUMF (lavender), for different numbers of events. The results for the three file sizes have been merged

[30] The commands to retrieve the respective information were `pmap -d pid tail -n 1 cut -d' ' -f7` for memory and `top -b -n 2 -d 0.2 -p pid tail -1 awk 'print $9'` for CPU, where `pid` was the process ID of the `xrootd` service.

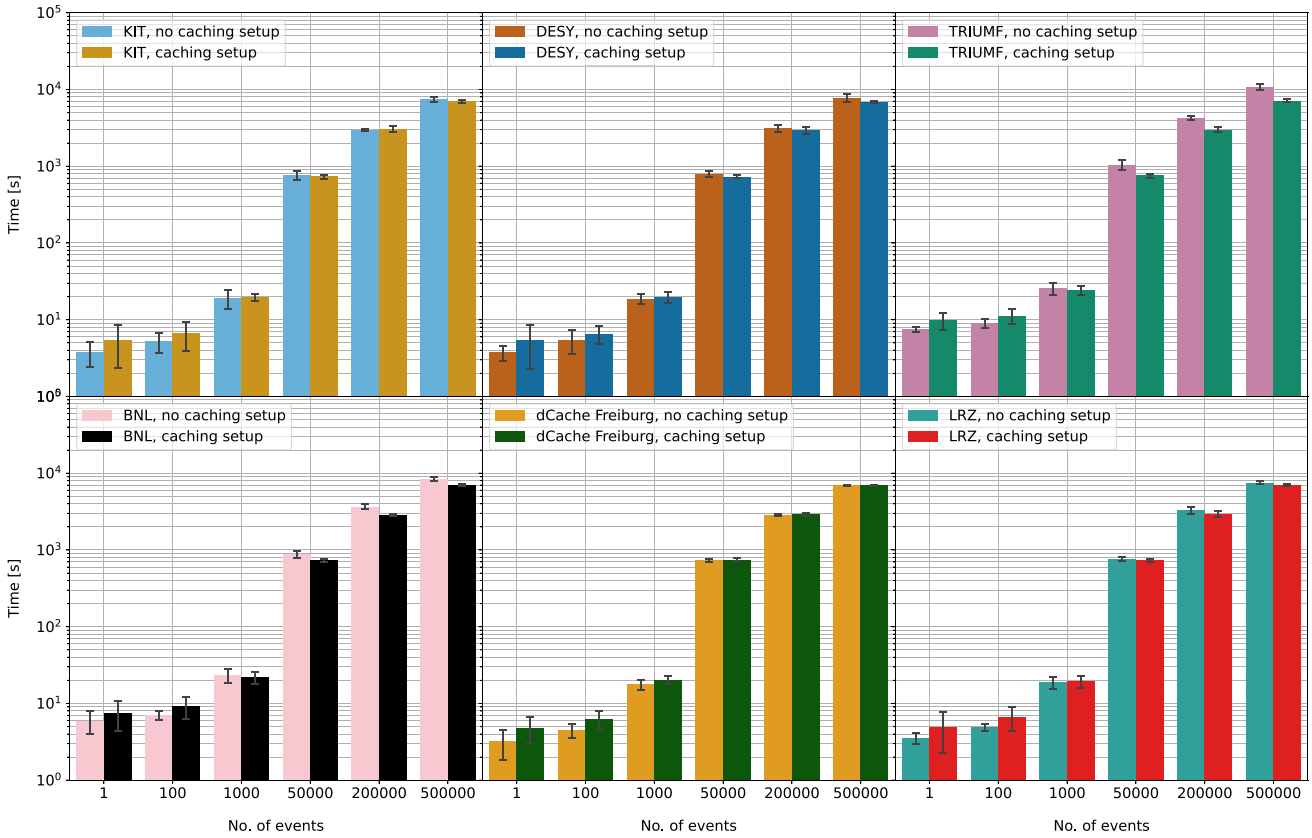

Fig. 5 Workflow benchmark results for KIT without caching setup (blue) and with caching setup (orange), DESY without caching setup (brown) and with caching setup (dark blue), TRIUMF without caching setup (lavender) and with caching setup (green), BNL without caching setup (pink) and with caching setup (black), dCache Freiburg without caching setup (dark yellow) and with caching setup (dark green), and LRZ without caching setup (turquoise) and with caching setup (red) for different numbers of events. The results for the three file sizes have been merged

Table 2 Workflow benchmark results without caching setup. Shown are the completion times of the benchmark for the different sites and numbers of events. The results for the three file sizes have been merged and are rounded to significant digits

No. of events	1	100	1k	50k	200k	500k
Site	Time [s]					
BeeGFS FR	2.9	4.24	17.1	730	2860	6960
dCache FR	3.2	4.5	17.6	730	2860	6890
LRZ	3.5	4.9	18.8	770	3300	7500
KIT	3.7	5.2	19	760	3000	7500
DESY	3.8	5.5	18.8	800	3140	7700
BNL	5.9	7.0	23	880	3660	8400
TRIUMF	7.5	9.1	26	1050	4260	10900

the caching setup is shown in Fig. 4. Reading files in the cache space is comparable to reading the files on sites that are geographically close to Freiburg, e.g., KIT. For sites at a larger distance, like BNL and TRIUMF, reading files in the cache space is faster up to a factor of ~ 1.5, and the client profits from the caching setup.

Figure 5 shows the workflow benchmark results for the external sites with and without the caching setup for a cold cache, i.e., when the file is not available in the local cache space. For 50k and more events, the completion time of the benchmark is lower for geographically far sites if the caching setup is used. This effect is largest for BNL and TRIUMF, but is also observed for DESY and LRZ.

The explanation for this effect is the simultaneous execution of the event loop by the client and the download of the file by the proxy server. For large numbers of events, the event loop, which reads the file from the external site, takes longer to complete than the time it takes to download

Table 3 Workflow benchmark results with caching setup. Shown are the completion times of the benchmark for the different sites and numbers of events. The results for the three file sizes have been merged and are rounded to significant digits

No. of events	1	100	1k	50k	200k	500k
Site	Time [s]					
Hot cache						
File in cache	3.6	4.8	18.1	740	2980	7060
Cold cache						
dCache FR	4.8	6.3	20.4	742	2880	6980
LRZ	5.0	6.7	19.5	730	2940	6960
DESY	5.4	6.6	19.6	730	2940	6860
KIT	5.4	6.7	19.5	730	3080	7010
BNL	7.5	9.3	22	739	2840	7050
TRIUMF	9.8	11.3	24.1	760	3000	7120

Table 4 Proxy server benchmark results. Thresholds, mean values, and standard deviations of the memory consumption for the different numbers of parallel requests

No. of requests	Threshold [GB]	Mean [GB]	SD [GB]
25	0.81	0.93	0.06
50	0.99	1.21	0.14
75	1.15	1.43	0.19
100	1.3	1.65	0.20
128	1.7	1.99	0.15

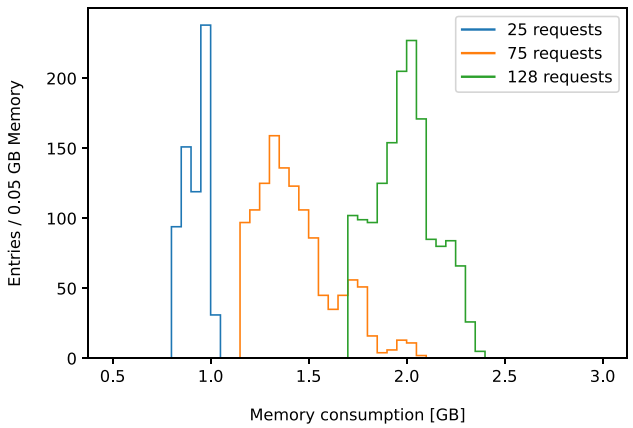

Fig. 6 Memory consumption of the `xrootd` service on the proxy server for 25, 75, and 128 parallel requests. Measurements below the respective thresholds are excluded

the file to the cache space. As soon as the file is completely available in the cache space, the event loop starts to read the local version of the file. Since reading the local version of the file is faster than reading from the external site, the completion time of the benchmark is reduced with respect to the benchmark without caching setup, where the file is read completely from the external site.

Because of this, using the caching setup results in faster file access for the client in case of large numbers of events

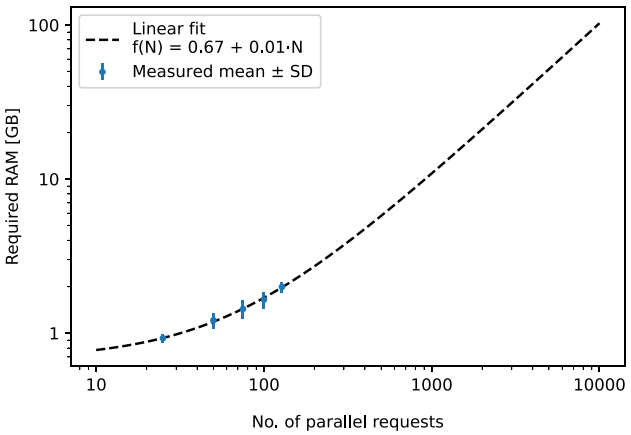

Fig. 7 Measured mean values and standard deviations of the memory consumption (blue dots), and the result of the linear fit (black dashed line)

and geographically far sites, even if the file is not already available in the cache space.

Proxy Server Results

In the analysis of the memory consumption of the `xrootd` service, values below a certain threshold were discarded to exclude idle time. With the remaining measurements, the mean value and the SD of the memory consumption were calculated for each number of parallel requests. The respective values for the different number of parallel requests are listed in Table 4.

Fig. 6 shows the distribution of memory consumption by the `xrootd` service for exemplary values of 25, 75, and 128 parallel requests.

The dependence of the memory consumption on the number of requests can be described by a linear function as shown in Fig. 7. The fit was executed with Numpy[31]

[31] https://numpy.org/.

Table 5 Proxy server benchmark results. Thresholds, mean values, and standard deviations of the CPU load for the different numbers of parallel requests

No. of requests	Threshold [%]	Mean [%]	SD [%]
25	5	53.28	29.76
50	5	64.68	40.37
75	5	86.84	52.59
100	5	96.94	58.01
128	5	101.59	61.28

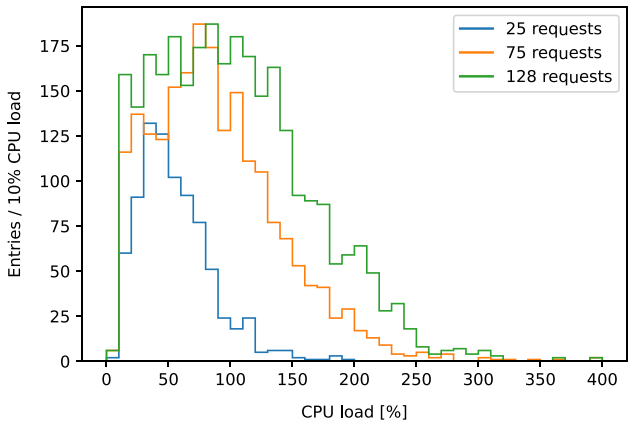

Fig. 8 CPU load of the `xrootd` service on the proxy server for 25, 75, and 128 parallel requests. Measurements below the threshold are excluded

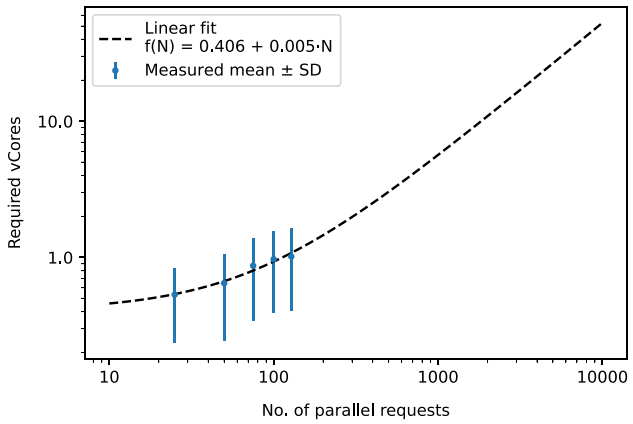

Fig. 9 Measured mean values and standard deviations of the CPU load (blue dots), and the result of the linear fit (black dashed line)

[13]. The function to get the required memory in GB for the number of parallel requests N is $f(N) = 0.67 + 0.01 \cdot N$. With this, the minimum number of required memory can be extrapolated to larger numbers of parallel requests, assuming that the observed linearity still holds for larger N. In a production environment, $1000 - 3000$ parallel requests are not unrealistic, and would require at least $\sim 11\,\text{GB} - 31\,\text{GB}$ of memory. The deployment of a cluster of proxy servers, instead of a single proxy server, would therefore be advisable. The implementation of such a setup is possible with XRootD.

The other measured metric was the CPU load of the `xrootd` service. As in the case of the analysis of the memory consumption, very low values of CPU load were observed during idle times. Therefore, a threshold of 5 % was introduced. The mean value and the SD of the CPU load were calculated from the remaining measurements for each number of parallel requests. The respective values for the different number of parallel requests are listed in Table 5.

Fig. 8 shows the distribution of CPU load by the `xrootd` service for exemplary values of 25, 75, and 128 parallel requests. Since the proxy server was equipped with 4 virtual cores, the maximum is at 400 %.

As in the case of the memory consumption, the dependence of the CPU load on the number of requests can be described by a linear function as shown in Fig. 9. All values were divided by 100 to derive a function for the number of required virtual cores (vCores). The function to get the required number of virtual cores for a given number of requests N is $f(N) = 0.406 + 0.005 \cdot N$. $1000 - 3000$ parallel requests would require at least $\sim 6 - 16$ virtual cores. This could be realized with a cluster of proxy servers consisting of 4 virtual machines.

Conclusion

A lightweight caching setup was successfully implemented for the HEP computing infrastructure in Freiburg, which includes VREs running on the opportunistically used HPC cluster NEMO. This setup consists of a client, a proxy server, and a cache space. No additional hardware was necessary for the cache space, since the file system provided by the HEP cluster was used. The only additional infrastructure component was a VM (4 cores and 8 GB RAM) serving as proxy server, which was running in the local VMware ESXi environment. Benchmarks that simulate a typical HEP workflow were devised to test the performance of the disk caching setup, and to measure the resource consumption on the proxy server during parallel file requests.

The disk caching setup outperforms non-cached access if the requested file is already available in the local cache space and if the external site is geographically far from Freiburg.

For large numbers of events in the benchmark, even initial requests, for which the file is not available in the cache space, are completed faster with the disk caching setup. The cause for this is the parallel download of the file, which makes it eventually available in the cache space.

These results have been achieved by using the default values of the caching parameters, e.g., *blocksize* and *prefetch*. Other values might result in a better performance, but might also depend on the specific environment where the caching setup is deployed. It is expected that the usage of SSD storage as cache space will increase the performance, but this has not been investigated since this study focuses on improvements with commodity hardware.

The resource consumption of the proxy server was measured by sending several requests in parallel, for different numbers of parallel requests. A linear dependence of both the memory consumption and the CPU load on the number of parallel requests was observed. This was used to extrapolate the measured resource consumption to larger, more realistic numbers of parallel requests. To handle $1000 - 3000$ parallel requests, at least $\sim 11\,\text{GB} - 31\,\text{GB}$ of memory and at least $\sim 6 - 16$ virtual cores would be necessary. This can be realized with a cluster of multiple, virtualized proxy servers, without the need for additional hardware.

Acknowledgements This research was performed on the bwForCluster NEMO supported by the Ministry of Science, Research and the Arts Baden-Württemberg through the bwHPC grant and by the German Research Foundation (DFG) through grant no. INST 39/963-1 FUGG (bwForCluster NEMO). The work was supported by the Federal Ministry of Education and Research (BMBF) within the projects 05H18VFRC1 "Entwicklung und Optimierung der Nutzung heterogener Rechenressourcen (Verbund: Innovative Digitale Technologien für die Erforschung von Universum und Materie (IDT-UM))" and 05H21VFRC2 "Weiterentwicklung, Integration und Optimierung von föderierten digitalen Infrastrukturen für ErUM (Verbund: Föderierte Digitale Infrastrukturen für die Erforschung von Universum und Materie (FIDIUM))".

Funding Open Access funding enabled and organized by Projekt DEAL.

Data availability The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

Declarations

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

References

1. Adaptive Computing Enterprises, Inc (2011) Introduction to Cloud for HPC. Technical report.
2. Alfieri R, Cecchini R, Ciaschini V, et al (2004) VOMS, an authorization system for virtual organizations. In: Fernández Rivera F, Bubak M, Gómez Tato A, et al (eds) AxGrids 2003: Grid Computing, Lecture Notes in Computer Science, vol 2970. Springer, Berlin, Heidelberg, Germany, pp 33–40, https://doi.org/10.1007/978-3-540-24689-3_5
3. ATLAS Collaboration (2022) ATLAS Software and Computing HL-LHC Roadmap. Technical report, CERN, Geneva, https://cds.cern.ch/record/2802918
4. Bierlich C, Chakraborty S, Desai N, et al (2022) A comprehensive guide to the physics and usage of PYTHIA 83. SciPost Phys Codebases. https://doi.org/10.21468/SciPostPhysCodeb.8
5. Bos K, Brook N, Duellmann D, et al (eds) (2005) LHC computing grid: technical design report. CERN, Geneva, https://cds.cern.ch/record/840543
6. Brun R, Rademakers F (1997) ROOT–an object oriented data analysis framework. Nucl Inst Meth in Phys Res A 389:81–86. https://doi.org/10.1016/S0168-9002(97)00048-X
7. Béjar Alonso I, Brüning O, Fessia P, et al (eds) (2020) High-Luminosity Large Hadron Collider (HL-LHC): technical design report. CERN Yellow Reports: Monographs, CERN, Geneva, https://doi.org/10.23731/CYRM-2020-0010
8. Dorigo A, Elmer P, Furano F et al (2005) XROOTD/TXNetFile: a highly scalable architecture for data access in the ROOT environment. WSEAS Trans Comput 4:348–354
9. Ernst M, Fuhrmann P, Gasthuber M, et al (2001) dCache, a distributed data storage caching system. In: Chen HS (ed) Proceedings of CHEP 2001. Beijing: Science Press
10. Fischer M, Kuehn E, Giffels M, et al (2022) Matterminers/cobald: v0.13.0. https://doi.org/10.5281/zenodo.1887872
11. Giffels M, Kroboth S, Schnepf M, et al (2021) Matterminers/tardis: The survivors (0.6.0). https://doi.org/10.5281/zenodo.2240605
12. Hanushevsky A, Ito H, Lassnig M et al (2019) Xcache in the atlas distributed computing environment. EPJ Web Conf. https://doi.org/10.1051/epjconf/201921404008
13. Harris CR, Millman KJ, van der Walt SJ et al (2020) Array programming with NumPy. Nature 585:357–362. https://doi.org/10.1038/s41586-020-2649-2
14. HEP Software Foundation (2019) A roadmap for HEP software and computing R &D for the 2020s. Comput Softw Big Sci. https://doi.org/10.1007/s41781-018-0018-8
15. Herold F, Breuner S (2018) An introduction to BeeGFS. Technical report.
16. Lüttgau J, Kuhn M, Duwe K (2018) Survey of storage systems for high-performance computing. Supercomput Front Innov 5(1):31–58. https://doi.org/10.14529/jsfi180103

17. Sammel D, Böhler M, Gamel AJ, et al (2022) How to bring HTC data to HPC resources: a caching solution for the ATLAS computing environment in Freiburg. In: Mosch C, Salk J, Wagner FW (eds) Proceedings of the 7th bwHPC Symposium. Open Access Repositorium der Universität Ulm und Technischen Hochschule Ulm, Ulm, Germany, pp 63–68. https://doi.org/10.18725/OPARU-46068
18. VMware, Inc. (2007) The Architecture of VMware ESXi. Technical report.
19. Wiebelt B, Meier K, Jancyk M, et al (2017) Flexible HPC: bwForCluster NEMO. In: Richling S, Baumann M, Heuveline V (eds) Proceedings of the 3rd bwHPC-Symposium. heiBOOKS, Heidelberg, Germany, https://doi.org/10.11588/heibooks.308.c3729
20. Yoo AB, Jette MA, Grondona M (2003) SLURM: simple linux utility for resource management. In: Feitelson D, Rudolph L, Schwiegelshohn U (eds) JSSPP 2003: Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, vol 2862. Springer, Berlin, Heidelberg, Germany, pp 44–60, https://doi.org/10.1007/10968987_3