

Dynamic Resource Extension for Data Intensive Computing with Specialized Software Environments on HPC Systems

Christoph Heidecker*  Matthias J. Schnepf*  R. Florian von Cube†
Manuel Giffels*  Martin Sauter* Günter Quast*

*Institute for Experimental Particle Physics, Karlsruhe Institute of Technology, Germany

†Physics Institute 3A, RWTH Aachen University, Germany

Modern High Energy Physics (HEP) requires large-scale processing of extensive amounts of scientific data. The needed computing resources are currently provided statically by HEP specific computing centers. To increase the number of available resources, for example to cover peak loads, the HEP computing development team at KIT concentrates on the dynamic integration of additional computing resources into the HEP infrastructure. Therefore, we developed ROCED, a tool to dynamically request and integrate computing resources including resources at HPC centers and commercial cloud providers. Since these resources usually do not support HEP software natively, we rely on virtualization and container technologies, which allows us to run HEP workflows on these so called opportunistic resources. Additionally, we study the efficient processing of huge amounts of data on a distributed infrastructure, where the data is usually stored at HEP specific data centers and is accessed remotely over WAN. To optimize the overall data throughput and to increase the CPU efficiency, we are currently developing an automated caching system for frequently used data that is transparently integrated into the distributed HEP computing infrastructure.

1 Introduction

Modern High Energy Physics (HEP) searches for the fundamental building blocks of matter by looking for new particles and rare processes at energies mankind has never reached before. This research requires both giant experimental setups and large scaled processing of huge amounts of data. While simulations are required to predict the behavior of characteristic quantities, data analyses scan for noticeable differences between measured data and theory prediction. Both require a huge amount of computing resources, which are currently deployed at HEP specific computing centers.

Since the demand for computing resources is heavily increasing in HEP, improvements of workflows and new computing concepts and resources are required. Whereas static computing resources that are dedicated to HEP serve the base load, the HEP computing development team at KIT concentrates on the dynamic integration of additional computing resources. This allows for example the covering peak loads caused by journal or conference deadlines. Furthermore, this allows the sharing of computing resources with other scientific communities to be guaranteed.

This makes the bwForCluster NEMO (Wiebelt et al., 2017), whose resources are shared among different scientific communities, ideally suited for our developments. At the bwForCluster NEMO, requested resources are allocated following a fair share policy and released again once there is no demand for additional HEP resources, so that they can be used by other communities. In order to provide HEP users easy access to these additional computing resources, they are integrated into one common batch system, offering one single point of entry to all available resources for HEP. For this, the Institute of Experimental Particle Physics (ETP) at the Karlsruhe Institute of Technology (KIT) developed a tool taking care of the automatic and dynamic integration of these so-called opportunistic computing resources.

In contrast to dedicated HEP computing resources, opportunistic resources such as the NEMO HPC cluster neither provide the HEP specific software environment nor the HEP infrastructure that is specialized for data-intensive processing. Hence, we need to adapt these resources to fully support HEP workflows. The deployment of a specialized software environment is enabled by the possibility to virtualize or containerize workflows at the NEMO HPC center. In contrast to traditional virtualization, the container technology as a more lightweight approach encapsulates processes within an own environment. Deploying a fully configured software envi-

onment using those technologies enables us to fulfill the HEP requirements. The KIT group is currently exploring both technologies in order to increase the amount of available non-HEP dedicated computing resources that can be utilized for HEP workflows.

HPC clusters are designed for CPU-intensive workflows with multiprocess interaction, data transfers, as well as CPU utilization suffer, when thousands of independent processes which run in parallel access data stored at remote HEP Grid storages. Therefore, we investigate how data-intensive processing can be enabled on opportunistic resources. In the caching concept developed at ETP, an intermediate cache layer between the worker nodes and the remote storage systems deploys fast access to repeatedly accessed data. Since most HEP workflows repeatedly process the same data, this concept reduces the pressure on the network and increases the data throughput and thus the CPU efficiency.

In the following, we present the concepts of dynamic integration of HPC computing resources, provisioning of software environments and caching infrastructure developed at KIT.

2 Resource Allocation and Integration

There are different kinds of providers of computing resources, such as HPC centers and commercial cloud providers. Depending on the provider, the computing resources have to be requested via common batch systems or via specialized APIs. At the bwForCluster NEMO, we request resources in the form of virtual machines via their MOAB¹ batch system in combination with an OpenStack installation (Meier et al., 2016).

After resource allocation, those resources are integrated into our overlay batch system, in our case HTCONDOR², which manages all HEP user jobs. The usage of an overlay batch system provides a single point of entry for the users and thus simplifies the usage of multiple resources. Due to the pull mechanism of HTCONDOR, the integration into the common HEP resource pool is done by simply starting an HTCONDOR client process on the virtualized worker nodes.

¹Moab Cluster Suite, Adaptive Computing, Inc. URL: <https://www.adaptivecomputing.com/>

²HTCondor, University of Wisconsin-Madison, <https://research.cs.wisc.edu/htcondor/>

For the dynamic allocation and integration of resources, the HEP community at KIT developed the resource manager RESPONSIVE ON-DEMAND CLOUD ENABLED DEPLOYMENT (ROCED)³. ROCED periodically monitors the job queue and the available resources in our overlay batch system. ROCED recognizes a demand for additional resources and automatically requests those at a resource provider. The design of ROCED using adapters allows for the utilization of a wide range of resources provided by different resource providers. When the allocated resources are not used anymore, the overlay batch system client stops after a certain period and releases those resources. This allows for dynamically adjusting the amount of allocated resources to the current demand for resources.

At the time of submission of this paper, ROCED has been used in production for integrating resources like the bwForCluster NEMO into our HTCONDOR batch system for over two years already. Figure 1 displays the automatic allocation of resources adjusted to the current demand for a period of six days. The number of dynamically allocated CPUs is compared to the number of jobs that are queued or running within the overlay batch system. It gives an example of the stable behavior of ROCED and the high utilization of NEMO resources by the KIT HEP community.

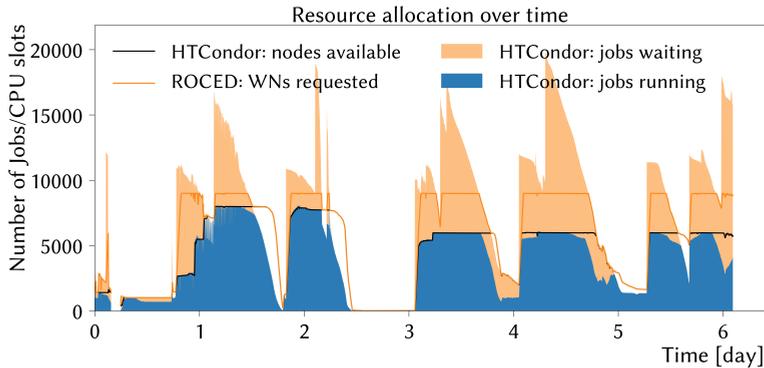


Figure 1: The demand for additional computing resources and their utilization at the NEMO HPC center is shown in units of CPU cores for an exemplary period of 6 days. The orange line shows the number of requested CPU cores, the black line shows the number of cores integrated into the overlay batch system. The blue area shows the number of the used CPU cores, whereas the stacked orange area shows the demand for more cores based on the overlay batch system's queue.

³ROCED, ROCED project, URL: <https://github.com/roced-scheduler/ROCED>

Currently, we are able to allocate up to 6000 CPU cores limited by fair share, which can be used for processing of HEP analyses. During testing phase, we were able to allocate up to 8000 CPU cores as shown in Figure 1. One of the analyses that profits heavily from the integration of NEMO resources is the Higgs analysis relying on the production of $\mu \rightarrow \tau$ embedded events (Bechtel et al., 2019), presented within these proceedings.

A more detailed view of the dynamic integration and provisioning of software environments is given in (Heidecker et al., 2018).

3 Provisioning of Software Environments

Scientific communities such as HEP have specific requirements regarding the environment to provide a verified software stack. This verified software environment ensures that the scientific results of the software can be relied on. Most analyses in the HEP community currently require Scientific Linux 6. However, over time, more and more analyses in HEP are switching to CERN CentOS 7. Both Scientific Linux 6 and CERN CentOS 7 are derivatives of RedHat Enterprise Linux 6/7 with a specific set of installed tools. However, those restrictions to specific software environments usually limit the amount of resources that can be utilized. Since most opportunistic resources do not provide the required software environment natively, virtualization and container technologies can be utilized for the provisioning of the HEP software environment. In this chapter, we give an overview of the utilized technologies as well as the advantages and disadvantages of some implementations we tested.

Virtualization is probably the most common technology of provisioning dedicated operating systems and environments. A complete operating system runs on a virtualized hardware, which is called a virtual machine. The host system itself decides what resources are shared with the virtual machine. Since the virtual machine is completely isolated from the host system, the user inside the virtual machine can be given all permissions. This enables the provision of a fully configured software environment inside the virtual machine, isolated from the host system and other users. In this case the HTCONDOR client process runs inside the virtual machine, which provides the HEP software environment as well. However, the virtualization of resources leads to administrative overhead and a few percent performance degradation.

A more lightweight solution to providing a software environment is to use container technologies. In contrast to virtualization, a container is a process that runs on the host system in an isolated namespace. This isolates the processes inside the container from other process on the host. Also in this case, the host controls the resources the container can access. Additionally to a certain memory and the number of CPU cores, this can also include direct access to a file system of the host.

Container technologies lead to less overhead than virtual machines, resulting in a smaller performance degradation. One implementation of the container technology is DOCKER⁴. DOCKER provides many additional features such as network monitoring of the isolated processes. Additionally, HTCONDOR offers built-in support for DOCKER, so that the batch system client is able to directly start jobs inside of DOCKER containers. On most of the HEP specific computing resources at KIT, we use this method to provide the required software environment. This allows us to use a modern operating system on bare metal and at the same time provide the HEP software environment for batch jobs. Furthermore, it is possible to provide different software environments for different jobs, which makes updates or changes more flexible.

Due to security concerns, DOCKER is usually not installed on HPC clusters. However, as the processes inside the virtual machine are completely isolated from the host system, it is possible to use DOCKER inside a virtual machine. On the bwForCluster NEMO, we use this setup in production, which allows us to use all additional features of DOCKER. This enables for example the monitoring of network usage as shown in Figure 2 for a typical data-intensive HEP job.

In the future, we would like to use this additional information for advanced job scheduling, taking into account available and utilized network bandwidth.

An alternative container technology is SINGULARITY (Kurtzer et al., 2017), which is explicitly developed for the usage on shared computing resources like HPC centers. It is designed to run completely in user space, which resolves security concerns, but results in a limited feature set compared to DOCKER. For example, the above-mentioned network monitoring is not possible using SINGULARITY containers. However, it enables direct access to the hosts file system like the common parallel filesystem provided at HPC centers, where jobs can profit from high bandwidth to

⁴Docker, Docker, Inc. version 17.05.0-ce. URL: <https://www.docker.com>

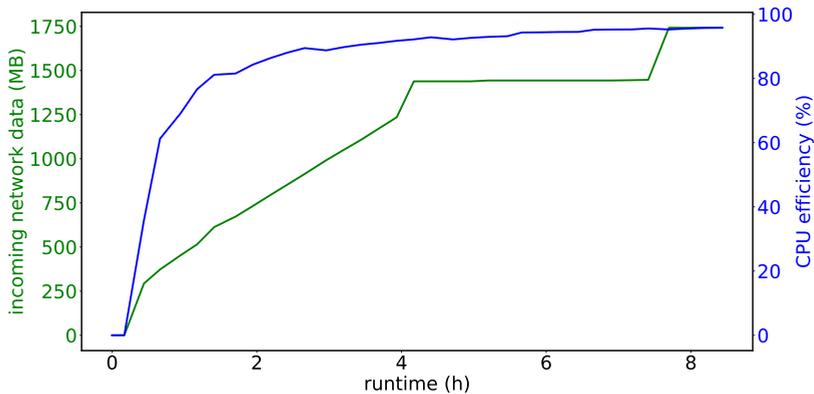


Figure 2: CPU efficiency (blue) and accumulated incoming network traffic (green) for a job of a typical HEP analysis workflow. The values were updated every 15 min. After the job’s initialization, the job reads data from remote storage and starts processing. This specific job read 1742.2 MB, took 8.5 h and had an 95.7% CPU efficiency on average.

data that is locally stored at the HPC center. This enables future optimizations of the data throughput for data-intensive HEP workflows.

Since DOCKER provides both the full feature set and the lowest overhead, we prefer setups using a pure DOCKER installation. Shared resources usually do not allow this, so we can either use virtual machines with integrated DOCKER support or SINGULARITY containers, depending on the resource provider. Here, one has to weigh the higher overhead caused by virtualization compared to containerization against the additional features provided by the combined setup. We prefer the combined setup using DOCKER containers within virtual machines, since we want to use the additional monitoring information provided by DOCKER to enhance our job to resource scheduling.

4 Optimization of data throughput

Workflows of the HEP community usually can be divided into two different types. Whereas simulations mainly require a lot of CPU hours, data analysis workflows require large datasets to be transferred to the worker nodes. The former benefit directly from the additional resources, but the latter require optimized infrastructures and adjusted hardware configurations.

Since opportunistic resources and especially HPC centers are not explicitly designed to support such data-intensive workflows, the HEP community at KIT is studying how to enable efficient processing of data. HPC centers as well as other opportunistic resources, usually do not provide sufficient long-term storage capacity to hold the huge amount of data produced by current HEP experiments. Hence, data analysis workflows transfer data from dedicated Grid storage systems to the HPC worker nodes for processing. The shared and limited network infrastructure between Grid storage and an HPC center limits the data transfers and can cause inefficient CPU utilization.

Here, we concentrate on a caching concept that stores parts of the accessed data on HPC storage volumes. This directly speeds up HEP analyses that repeatedly access the same data by exploiting the available high-performance parallel files systems available at the HPC center. Furthermore, it reduces the load on the shared external network connection and thus improves data throughput.

For the transparent integration of the caching approach into current HEP infrastructure, we concentrate on HEP specific data transfer protocols and the workflow management infrastructure HTCONDOR. We decided to concentrate on the XROOTD protocol (Dorigo et al., 2005), which was developed by the HEP community for world-wide access to a huge amount of data. It supports a hierarchical infrastructure to manage distributed storage systems and already provides basic caching functionalities. For the intermediate caching infrastructure, caching proxies are installed, which allow transparent access to remote storage capacities while caching frequently used data on storage volumes at the data center. Repeated access to already cached data is redirected to the copy at the data center instead of accessing the Grid storage element. Therefore, we deployed XROOTD cache proxies at the NEMO HPC center that utilize the distributed high-performance storage capacities to optimize data throughput for the processing of HEP workflows. The maximum bandwidth that the proxy can provide with this setup depends on both the performance of the storage system and the available bandwidth of the host running XROOTD services. We can avoid this limitation, since the jobs can also directly read cached files from the parallel file system. Unfortunately, it is not allowed to mount the storage volumes inside a virtual machine, which we currently use to provide our required software environment. However, using the container technology SINGULARITY allows for the use of the storage volumes without any security concerns.

In this case, the access request still goes through the XROOTD hierarchy to allow access to remote data as well as on-the-fly caching of data while streaming it to the worker. However, access to already cached files is redirected to the local mount of the parallel file system profiting from the fast Omni-Path interconnects. When using SINGULARITY containers, it is no longer as easy to determine the network traffic of the individual jobs as in the current setup. Hence, we currently prefer the first setup, which allows us to use Docker container within virtual machines providing monitoring data of the network utilization of jobs. However, on other HPC centers such as the ForHLR2 (Barthel et al., 2017) at Karlsruhe, which has no virtualization, the usage of SINGULARITY as well as the local redirection for cached data is a useful alternative to the NEMO setup.

As part of a large-scale computing infrastructure for future HEP experiments, we want to install caches at each computing resource connected to the HEP infrastructure. In this case, the challenge will be improving the data throughput of the processed workflows and efficiently utilize the limited cache volumes. Therefore, the HEP community at the KIT studies the advantages of coordinated caching. Here, multiple caches are coordinated to reach efficient utilization of limited cache volumes. Furthermore, data-intensive jobs are matched to the most suitable computing resource that already has cached requested files. This enables an optimized utilization of the computing resources and thus shorter processing times of the processed workflows.

In order to introduce the concept of data locality into the HTCONDOR batch system, an extension is currently being developed. As shown in Figure 3, it fetches location information from the XRootD service and injects it into the job-to-resource matching. In addition, this new HTCondor extension also coordinates the cache location of newly arrived data for further processing.

First studies on a prototype infrastructure showed a big improvement of the data throughput and thus a better utilized CPU. The highest CPU utilization was achieved by a combined utilization of cache access and network transfers. The advantages of such an infrastructure optimization for opportunistic resources will be studied in detail at NEMO. A successful utilization of the caching concept at the NEMO HPC center will enable future HEP experiments to efficiently utilize different kinds of opportunistic resources for data processing.

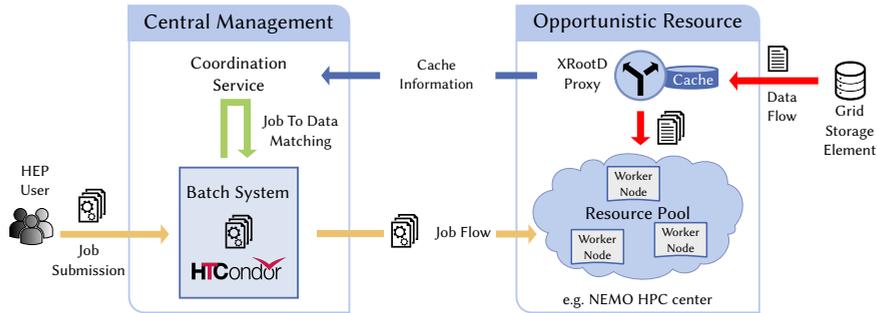


Figure 3: An XRootD proxy caches data transferred from Grid storage element on the fly. An extension of the HTCONDOR batch system injects the location of cached data into the job matching. Hence, jobs are matched to the most suitable computing resource that provides the best cache hit rate.

5 Summary

Opportunistic resources such as the shared HPC center NEMO provide a huge amount of computing resources.

Using virtualization and container technology, we are able to provide our HEP software environment on different resources such as HPC centers or commercial cloud providers. The preferred setup is to start containers inside virtual machines, which enables network monitoring for each of our batch jobs.

The HEP community at KIT developed ROCED, a tool to dynamically request resources and integrate them into common HEP computing infrastructure. At the bwForCluster NEMO, this setup has been in production for over two years, which allows us to efficiently utilize shared resources and thus cover peak loads.

Over time, we recognized CPU inefficiencies of data intensive jobs due to shared and limited network capacities. In order to avoid these inefficiencies, a concept of coordinated data caches utilizing the locally available parallel file system has been developed, which allows us to place frequently used data temporarily local to the workers. The concept developed in this manner is also applicable to other opportunistic resources as well as computing clusters with locally available storage attached.

With the presented techniques we are able to process all kind of batch jobs of HEP workflows on many different opportunistic resources in a dynamic and efficient way.

Acknowledgment

The authors acknowledge the support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG.

Corresponding Author

Christoph Heidecker: christoph.heidecker@kit.edu

Matthias Schnepf: matthias.schnepf@kit.edu

Institute of Experimental Particle Physics (ETP),
Karlsruhe Institute of Technology, Germany

ORCID

Christoph Heidecker  <https://orcid.org/0000-0002-8361-4520>

Matthias Schnepf  <https://orcid.org/0000-0003-0623-0184>

Manuel Giffels  <https://orcid.org/0000-0003-0193-3032>

License    4.0 <https://creativecommons.org/licenses/by-sa/4.0>

References

- Barthel, R. and S. Raffeiner (2017). »ForHLR: a New Tier-2 High-Performance Computing System for Research. October 12th 2016, Heidelberg«. eng. In: *Proceedings of the 3rd bwHPC-Symposium*. Heidelberg: Universitätsbibliothek Heidelberg, pp. 73–75. ISBN: 978-3-946531-70-8. DOI: [10.11588/heibooks.308.418](https://doi.org/10.11588/heibooks.308.418).
- Bechtel, J., S. Brommer, A. Gottmann, G. Quast and R. Wolf (2019). »Performance of the bwHPC cluster in the production of $\mu \rightarrow \tau$ embedded events used for the prediction of background for $H \rightarrow \tau\tau$ analyses«. In: *Proceedings of the 5th bwHPC Symposium. HPC Activities in Baden-Württemberg*. Freiburg, September 2018. 5th bwHPC Symposium. TLP, Tübingen, pp. 61–73. DOI: [10.15496/publikation-29043](https://doi.org/10.15496/publikation-29043).
- Dorigo, A., P. Elmer, F. Furano and A. Hanushevsky (2005). »XROOTD/TXNetFile: A Highly Scalable Architecture for Data Access in the ROOT Environment«. In: *Proceedings of the 4th WSEAS International Conference on Telecommunications and Informatics*. TELE-INFO'05. Prague, Czech Republic: World Scientific, Engineering Academy and Society (WSEAS), 46:1–46:6. ISBN: 960-8457-11-4. URL: <http://dl.acm.org/citation.cfm?id=1391157.1391203>.

- Heidecker, C., M. Giffels, G. Quast, K. Rabbertz and M. Schnepf (2018). »High precision predictions for particle collisions at the Large Hadron Collider«. eng. In: *Proceedings of the 4th bwHPC Symposium*. Tübingen: Universität Tübingen, pp. 28–31. DOI: 10.15496/publikation-25195.
- Kurtzer, G. M., V. Sochat and M. W. Bauer (2017). »Singularity: Scientific containers for mobility of compute«. In: *PLOS ONE* 12.5, pp. 1–20. DOI: 10.1371/journal.pone.0177459.
- Meier, K. et al. (2016). »Dynamic provisioning of a HEP computing infrastructure on a shared hybrid HPC system«. In: *Journal of Physics: Conference Series* 762.1, p. 012012. DOI: 10.1088/1742-6596/762/1/012012.
- Wiebelt, B., K. Meier, M. Janczyk and D. von Suchodoletz (2017). »Flexible HPC: bw-ForCluster NEMO. October 12th 2016, Heidelberg«. eng. In: *Proceedings of the 3rd bwHPC-Symposium*. Heidelberg: Universitätsbibliothek Heidelberg, pp. 128–130. ISBN: 978-3-946531-70-8. DOI: 10.11588/heibooks.308.418.