Consolidating the interactive analysis and Grid infrastructure at DESY.

Christoph Beyer[1], *Thomas* Finnern[1], *Martin* Flemming[1], *Andreas* Gellrich[1], *Thomas* Hartmann[1]*, *Yves* Kemp[1], , *Birgit* Lewendel[1], *Johannes* Reppin[1], *Krunoslav* Sever[1], , *Sven* Sternberger[1], and *Christian* Voss[1]

[1]DESY, Notkestraße 85, D-22607 Hamburg, Germany

Abstract. Within WLCG, the DESY site in Hamburg is one of the largest Tier-2 sites with about 18500 CPU cores for *Grid* workloads. Additionally, about 8000 CPU cores are available for interactive user analyses in the *National Analysis Factory* [NAF]. After migrating these two batch systems onto a common HT-Condor based set-up during the previous four years, we recapitulate the lessons learned during the transition especially since both use cases differ in their work-loads. For Grid jobs start-up latencies are negligible and the primary focus is on an optimal utilization of the resources. Complementary, users of the NAF expect a high responsiveness of the batch system as well as the storage for interactive analyses. In this document, we will also give an outlook to future developments and concepts for the DESY high-throughput computing. In the ongoing evolution of the HTC batch system, we are exploring how to integrate anonymous jobs with the batch system as back-end for Function-as-a-Service workflows as well as an option for dynamic expansions to remote computing resources.

1 Introduction

DESY's main compute resources at the Hamburg site consist of two clusters. One system is the high-performance "*Maxwell*" cluster for memory-limited and low latency tasks such as online reconstruction of data from the European XFEL or PETRA III beamlines as well as complex simulations. This cluster operates with the SLURM scheduler and has primarily a per node scheduling policy and a low-letency interconnect between nodes. For production tasks and user analyses the complementary high-throughput computing *HTC* cluster operates with a per job requirement policy and with ethernet as interconnect between nodes. Here the aim is on an optimal utilization of resources. The HTC batch cluster is based on HTCondor [2] and serves two main user groups.

One user group consists of high-energy physics communities like ATLAS, Belle II or CMS. Workloads from these communities are predominantly production-like tasks and are brokered through the Worldwide LHC Computing Grid. Such Grid jobs do not require short start latencies and are mostly of a few common job types of like CPU-intensive simulations or I/O bound reprocessing jobs.

*e-mail: thomas.hartmann@desy.de

The other user group is individual users from the same high energy physics communities. These users run their jobs individually and their job requirements on CPU, I/O, memory or latency tend to vary much more than Grid workloads. User expectations range from immediate starts of minimal jobs with run times of a few seconds to large tasks in the order of CPU months.

In this paper we describe how we consolidated over the last years the HTC batch cluster to cover both HTC user groups as well as the lessons learnt.

As the HTC clusters are suitable for a broad range of workloads, a number of services and policies have been added over the years. For example, we dynamically run user Jupyter notebooks [4] as HTC jobs and use the NAF cluster to offload the user notebooks. Also we investigate dynamic Apache Spark clusters [5] offloading the Spark workers to the HTC cluster as computing back-end. Due to the flexibility of the cluster, we investigate the HTC cluster as back-end for a light-weight Function-as-a-Service (FaaS) application. Since the FaaS engines are optimized for fast, low-latency responses, we aim to offload heavy computational tasks to the HTC batch system. As such we aim to make massive computing power available within the FaaS framework as well as extend the batch HTC cluster to function-like workflows.

2 HTC Batch Cluster Set-Up

The HTC batch cluster consists of computing nodes dedicated to the Grid production-like workloads and end user analyses within the NAF. We migrated the previous Grid cluster, based on MySched [6], to HTCondor in 2015 and migrated in 2017 the previous NAF cluster from PBS to HTCondor.

With both use cases on the same technical base, the original intention is to completely merge both use cases without differentiating between both job classes.

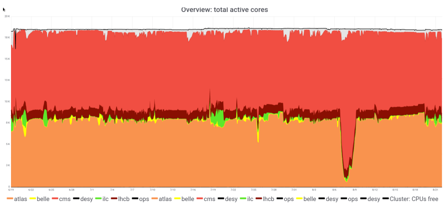
2.1 Workload Patterns

Figures 1a and 1b show the utilization of HTC cores dedicated to Grid and end user analyses. The differences in their utilizations is obvious. The Grid resources are in use for >95% of the time and the resources distribution to the user groups is quite constant. In contrast, the user jobs within the NAF context are much more variable and individual users' and groups' can change their computing demand significantly over short time periods. Some groups/users show a more production-like usage, while other groups/users show clear spiking behaviours in their need for computing resources.

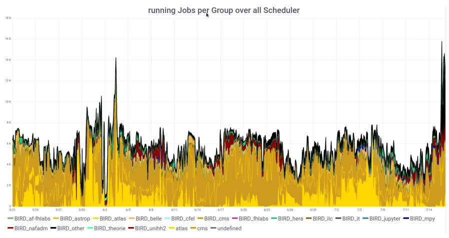
As example for resource utilizations of individual jobs, Figure 2 shows the CPU usage of Grid jobs on a batch node over one day. In general, such Grid jobs tend to have run times with at least one hour and recurring usage patterns, such as CPU or I/O intensive tasks.

In contrast, user jobs within the NAF can range from arrays of jobs with run-times of a few seconds to jobs, that run multiple days. To avoid blocking resources for too long, we enforce a maximum job runtime of one week as the aim of the NAF is to allow interactive workflows as well. Also myriads of short jobs pose a challenge. Millions of jobs submitted by a user in one round can for one overload the batch system's negotiation capacity. And short jobs can be in general a significant source for inefficiencies if their run times are shorter than a matchmaking cycle. To intercept such adventurous user behaviour, we configured the HTCondor start daemons on the NAF batch nodes to keep computing slots open for a user for a few minutes. As thus a batch node's daemon will attract and run subsequent short jobs by the same user reducing the brokering overhead for jobs with runtimes less than the negotiation cycle.

Grid workflows rely on WAN protocols like http, xrootd or gridFTP for reading or writing files. As such, staging files within the job is common for processing. In contrast, individual users assume a global namespace with all storages being available on all nodes. We provide different types of storage types on the NAF ranging from home-directories over AFS, a fast scratch space based on IBM Spectrum Scale [13] and for long-term storage dCache instances [14] per major user group. The scratch space and long-term storage are mounted via NFSv4 and NFSv4.1, respectively, on all user facing nodes via TCP/IP over ethernet. Like for job brokerage, user access patterns can be straining to the storage systems. For example, users could happen to assume, that massive parallel r/w accesses to single files is possible. To react to such patterns, we need to careful hardening of the systems. On the client side, also the NFS4 driver implementations in older Linux kernels <3.10 showed to be not as stable as more modern Linux releases. Since some users still depend on Scientific Linux 6, we are encouraging users to move to containers to ease the necessary transition to more current Linux distributions.



(a) Grid workloads.



(b) User analyses.

Figure 1: Utilization of the HTCondor batch clusters over two months.

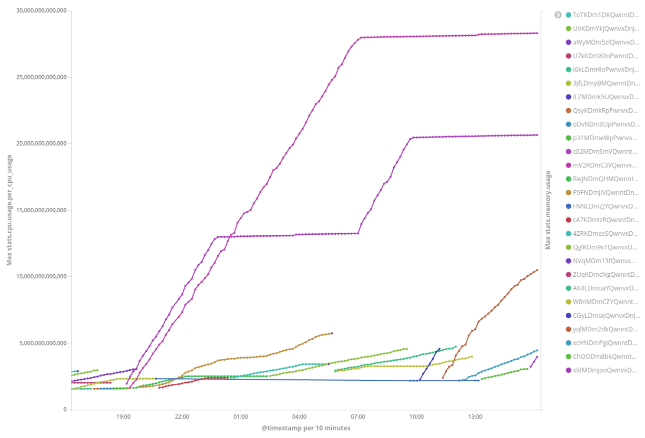


Figure 2: Aggregated CPU utilization of Grid jobs on a batch node over 24 hours.

2.2 Cluster Consolidation

We initially envisaged a complete consolidation of Grid and NAF into a single HTCondor instance. We assumed that we can run Grid as well as user workflows on the same nodes and profit from the combined entropy of different jobs, i.e., filling between longer running Grid jobs free resource with shorter user jobs.

However, due to the various challenging user workflows, we decided against running Grid and NAF user jobs in a fully merged cluster. Thus, we configured the batch nodes to attract only either Grid jobs or only NAF user jobs but not both.

On the administrative level however, we benefit from HTCondor as the common technical base. We have optimized the node maintenance and set up & configure the HTCondor cluster members from the same Puppet [15] definitions. Batch nodes running Grid workloads or NAF user jobs are based on the same Puppet manifests and we differentiate them only in their detail configuration managed via Hiera [16] variables. While in principle a mixed workload cluster would be possible, where batch nodes are grouped by node specific tags into NAF or Grid workloads. However, due to the fluctuating NAF workload patterns, we keep currently NAF and Grid workloads explicitly separated.

We had considered flocking clusters to allow jobs from one sub-cluster to overflow into the other to optimize utilization of resources. But as the Grid resources are mostly fully utilized, NAF jobs would only very rarely overflow in that direction. Allowing Grid jobs to overflow to NAF resources would negatively impact the responsiveness. As the NAF resources are already utilized beyond 80% on average, utilizing the remaining free resources would negatively impact the fast job turn-around that we require for interactive user analyses. Backfilling of NAF resources with preemptable jobs might be an option. However, preemptable jobs still require more than 30s as grace period to shut down properly as experiences from backfilling our Maxwell HPC cluster has shown.

Thus, we manage the cluster resources through the same infrastructure but keeping the workloads separate.

3 Access & Authentication

The interface for Grid jobs to the batch system is based on ARC CE 5 [7]. On the NAF side, users can submit jobs through one or more workgroup servers per experiment group. These workgroup servers do not submit directly to HTCondor but act as remote submitters to the actual schedulers. This allows us to be more flexible during maintenances.

While Grid workloads stage their input and output data primarily via network protocols as http, xrootd or dcap, NAF users expect remote storage resources to be available in the local namespace. Thus, we mount via NFSv4 the dCache long-term storage instances, a fast scratch space based on GPFS/IBM Spectrum Scale and the AFS home directories into the batch nodes' local namespaces. Access control to these remote file systems is managed through UIDs/GIDs and AFS tokens, which are derived from Kerberos tickets.

Since Kerberos tickets have limited lifetimes, jobs would be limited to these lifetimes for remote file I/O operations. To allow users longer queued and running jobs, we added a token renewal service. Here, a renewal shepherd updates user Kerberos tickets on a regularly basis on the batch nodes with corresponding user jobs.

4 Jupyter Notebooks

Over the last years, Jupyter notebooks have become a major alternative for accessing computing resources compared to classic approaches like `ssh`. For to access NAF resources we have set up a Jupyter hub. Users can initiate on it interactive Jupyter notebooks where the actual notebook instance is run within a HTCondor job. As our HTCondor nodes are already intended for NAF user jobs, this allows for a dynamic scaling of notebook resources as the existing batch infrastructure is already available including storages expected by users.

An user can spawn a new notebook from the Jupyter hub web user interface. In the background, a HTCondor job is submitted to the cluster. After start-up of the notebook job, the URL and token for accessing the notebook instance is handed back to the user. The user than can then transparently interact with her or his notebook instance through a http browser.

A major challenge was to optimize the responsiveness of the job start. While users can accept that ordinary batch job can take up to a few minutes, for notebooks users expect a more *immediate* start as the offloading to the batch system is intentionally hidden. For this we have defined a specific jobs class for Jupyter notebooks in HTCondor and on each batch node we reserve a dedicated slot for this job class. As Jupyter notebooks can be idle for most of the time, such a slot can in principle be overbooked with respect to the CPU core count per node. However, the memory usage per job puts a natural limit to overbooking and we rely on the overall job entropy per node to not require all the requested memory. Thus, we can keep dedicated slots available for user notebooks and reduce the latency for starting notebook jobs to be in the order of the negotiation cycle of the HTCondor resource manager.

5 Containerization

Since containerized applications have become the industry solution to bundle a program's dependencies, we are integrating container support into the HTCondor cluster. Originating from the High-Performance Computing community, Singularity [18] turned out to be the preferred container engine for high-throughput Grid workloads within the HEP community. Singularity's main advantages are its lightweight design as no long-running daemon processes are necessary and the possibility to run transportable container images in user space.

5.1 Singularity Integration

When we migrated the batch resources for Grid workloads from the ageing *Scientific Linux 6* to *CentOS 7* in 2018, we set up a dedicated compute element for legacy applications. Jobs submitted to the this CE are transparently transformed and started in a *Scientific Linux 6* Singularity container on a *CentOS 7* batch node. We host the *Scientific Linux 6* Singularity image as an expanded directory tree in a CVMFS [19] repository for dynamic distribution. This allows legacy users to make use of our *CentOS 7* resources while extending their adaptation time for a full migration to *CentOS 7*.

For end users on NAF resources, we have added support for Singularity containers as well. Users can request in their job submission scripts that their job payloads are started in a Singularity container of their choice. As the end of life of *Scientific Linux 6* is nearing at the end of 2020 and as the popular Python version 2 has already left support, we actively support our users to migrate their legacy workloads to such container solutions, since we have to migrate all native *Scientific Linux 6* installations to newer releases due to good practise and compliance rules.

6 Opportunistic Resources

Complementary to the High-Throughput Computing cluster, DESY operates a high-throughput computing cluster "*Maxwell*", that operates on a full node scheduling policy. Main users are on-site and related photon resources like *PETRA-III* and the *European XFEL* as well as theoretical physics and detector development communities. While the "*Maxwell*" cluster is well utilized during data taking periods, in intermediate times nodes can sometimes be idle for soem time. To opportunistically harvest such free computing resources, we set up a dedicated Compute Element to submit HEP workloads. If compute resources are requested by high priority users, these opportunistic jobs have to be evicted within 30s between a warning `SIGTERM` and final `SIGKILL` signal.

Similarly we are investigating on how to best utilize idle resources in our OpenStack virtualization instance without affecting running virtual machines in their CPU, memory or disk/network I/O.

7 Outlook

Both, the "*Maxwell*" HPC cluster and the HTCondor infrastructure, constitute the most significant of the computing power at the DESY Hamburg site. We plan to harness these resources as back-end for other internal services as well as to dynamically scale-out to external resources.

In a scale-out scenario, the simplest approach could be to start HTCondor batch nodes daemons in dynamically created resources. Such a dynamic daemon would register its available computing resources to the central HTCondor cluster manager and accept jobs in the following. Since such resources are not necessarily under complete control of DESY, either one would need to ensure a trusted authentication and authorisation scheme or find ways to mitigate potential trust issues.

For on-site resources, one can assume that a common authentication canopy would be possible. But remote resoruces managed by other entities pose a significant challenge. To fully trust such remote resources, we would need to manage a complex identity namespace distributed over all interconnected services, that can range from storage instances with file ownerships and permissions to identities under which applications are executed. Additional compexity originates from many external scientists visiting DESY only temporarily.

Currently visiting scientists need to be granted local accounts to access the local compute and storage resources. In many cases the scientists need such accounts only for a limited time until the results are evaluated and published. If one could move from dedicated user accounts to an authorization a token for data taking episode, the risk of an user identity loss could be mitigated.

Thus, a common approach to handle authentication over distributed local services as well as as remote resources would be desirable.

7.1 Event-Driven Workflow Integration

In HTCondor complex workflows can be realized with DAGman as meta-scheduler [3]. However, DAGman allows only to realize workflows of inter-dependent jobs within HTCondor. To integrate HTCondor with external events and workflows, a broader workflow engine is needed.

Classic workflow engines poll states from the connected systems and initiate new actions based on rules. In contrast, in an event-based scheme the workflow moves from *poll* to a *push* model, where the status change itself initiates a new action. With an event-type message as intermediary, a status change in one system can drive complex workflows through triggers and rule-sets without atomic steps of polling and compiling states. To integrate various systems into such an event-based workflow set-up, a common message bus is necessary for exchanging events. Complementary, conditions, triggers and the actual actions in form of functions require a form a registry.

To realize such a side-wide workflow set-up, we use Apache Kafka [8] as message bus and Apache OpenWhisk [9] as FaaS platform to connect events and actions.

7.1.1 dCache Storage Event

Storage events are a prominent example for event-driven workflows. Since version 4.2 *dCache* allows to announce changes in its namespace and propagated these changes as individual events via Kafka [11]. Integrating the dCache Kafka event streams with a FaaS platform would allows then for automatized workflows. For example, in an event-driven workflow a dCache instance generates itself a message of a file transfer to have finished, which then initiates automatically the processing of this file with a pre-registered function. Receiving such an event, a rule engine evaluates the event with pre-registered rules and initiates the processing job from a set of registered functions. Thus, recurrent tasks could be automatized and further chained to complex workflows spanning different systems, which could be on-side as well as potentially off-side.

In contrast, a classic workflow system would be waiting for a file transfer to finish checking regularly for the transfer status, before submitting a processing job on the file after numerous polls.

7.1.2 FaaS-Offloading

As functions are intended to be light-weight and short running applications, a FaaS platform like OpenWhisk is not ideal for processing of large amounts of data. Thus, it is advisable to off-load for any significant data processing onto a better suited batch system. For this, we integrate a HTCondor cluster into the FaaS platform, where a storage event initiates a compute jobs submission to the HTCondor cluster. Such a workflow is sketched in Figure 3, where a function initiates the heavy processing job, which is addressed merely indirectly and does not need to be called directly. As such it acts like a template generating a compute job from the pre-registered definition of the function [10].

7.2 Anonymous Jobs

In all common operating systems, processes are bound to local users and groups. Similar, files belong to a user and group and possibly more fine grained access control rules. To connect different storage and compute elements, one would need a common identity namespace or mapping from one ID namespace to another, where operation authorization is derived from authentication. Naturally, managing and securing such a global identity and authorization namespace can be arbitrary complicated as it requires the same and consistent authentication in all involved elements or a complex mapping.

Moving from authentication to an authorisation based resource control can allow a more secure set-up and especially more flexible workflows. Especially, anonymous jobs, that are not tied to a specific identity, would allow to scale-out to external, less trusted resources, e.g., dynamically spawned VMs in external clouds or user desktops on the campus.

7.2.1 Authorization Tokens

In a token-based access scheme, usage of compute resources or file access is based only on the ownership of a certain token. On presenting a token with a specific set of capabilities, a compute or storage element can grants to the anonymous owner of the token access to its resources. Since for most tokens schemes capabilities can be fine-grained and restricted, the actual capabilities can be limited to only the very necessary.

In a FaaS set-up combined with a token access control, only a central token generator and authentication instance are needed. Here, a template function is set up in the FaaS framework

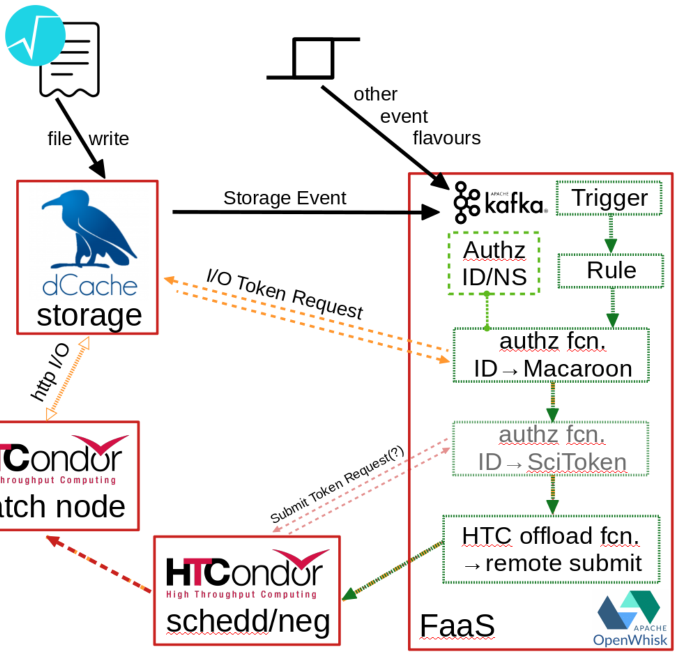


Figure 3: Storage event based workflow for automatic processing of files through functions and off-loading to the HTCondor batch system

to generate access tokens on request. It is the only instance, that needs knowledge about identities and their capabilities regarding resources. On an incoming event, it request on behalf of the registered identity by its set of rules an authorisation tokens from the corresponding storage or compute element. Such a tokens can be strictly limited to the needs of the actual process, e.g., a file token that allows to read just one specific file.

7.2.2 dCache Storage Macaroons

For example, the dCache team [12] has implemented macaroons [17] as authorization token scheme since version 4.2. Such a macaroon token can be restricted to various limitations like specific paths, IP ranges or time frames. The bearer of such a macaroon token then can only read or write files within the limitations. Since only ownership of a macaroon is relevant, the identity of the bearer can be irrelevant.

7.2.3 Token-based Jobs

As sketched in Figure 3, we envisage compute jobs that are by their nature *anonymous*. If the file I/O is authorized by macaroons and if the submission to a batch system is also managed through a token, such a job would have no need to run under a dedicated user/group ID. An anonymous job could be executed on a given, somewhat arbitrary host under a generic user account with reduced capabilities, which would limit the abuse potential.

Obviously, stripping any authentication requirements from an actual job implies that another stage in the whole workflow has to handle the authentication. Such an authentication stage has to ensure the validity of the actual user, who defines and initiates the workflow.

Also the authentication stage needs to use the users credentials to request authorization tokens from any remote resources. Thus, a central authentication-to-authorization stage allows to concentrate sensible data handling to a well defined and limited system - naturally, due to its sensitivity, it needs to be well guarded and managed with care.

In our current approach, we work on implementing such an authentication-to-authorisation stage as a set of function in our OpenWhisk framework. Sensible information like credentials in form of X509 certificates etc. are kept in a Gitlab CI/CD secrets store[20] to which only dedicated functions have access.

7.2.4 Example Workflow

For example, a user is regularly reprocessing her or his data with a static, rarely changing program. On a storage event, an authz function requests on behalf of the user from the dCache storage instance one macaroon token for reading the input file and a second token to write any output data to a separate, dedicated path.

Followingly, a processing job would be generated receiving the just generated macaroons to read and writes its in- and output. Since no authentication is required at the processing stage, the job process itself can run under an *anonymous* user account, that exists only locally on final batch node.

By decoupling the input/output from the identity namespace, this would further allow to more easily scale out to external, less trusted resources. If a malicious actor would intercept such a token, the harm would be limited to an unintended read of only the input file as well as writing unwanted output onto the output path, where the potential harm is wasted space without any option corrupt any other data.

8 Conclusion

With the goal to consolidate high-throughput clusters for Grid and NAF user workloads, we merged our management infrastructure and reduced significantly the administrative workload. Production and user jobs can show quite different usage patterns that each require specific optimizations to best utilize and protect the batch resources. With the gained experiences, we work on further integrating all computing resources at DESY. As such, workloads can range from classic high-throughput jobs over user jobs, with a wide field of job patterns, to high-performance workloads. While these requirements can probably not be solved by one technical solution, we envisage to ease for users the selection and use of the computing resources without putting the burden on users to become experts in one or the other technical product.

Furthermore we are working on integrating a wide range of computing resources into the DESY compute pool for opportunistic usage.

To make the our computing resources more pervasively available, we actively investigate novel workflows, where the actual computing process is not actively initialized by a user but just an inevitable component of an event triggered process chain.

References

[1] For all URLs memento snapshots have been preserved on `archive.org`. To get each URL's document version as seen during writing, call the original URL under the `202001` timestamp, e.g., `https://web.archive.org/web/202002/`*`original-url`*

[2] D.Thain, T. Tannenbaum and M. Livny, Concurrency - Practice and Experience **17**, 323-356 (2015)

[3] Peter Couvares, Tevik Kosar, Alain Roy, Jeff Weber and Kent Wenger, "Workflow in Condor", in In Workflows for e-Science, Springer Press, January 2007 https://research.cs.wisc.edu/htcondor/dagman/dagman.html

[4] Jupyter "*The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.* ", `https://jupyter.org/`

[5] Apache Spark "*Apache Spark^{TM} is a unified analytics engine for large-scale data processing.*", `https://spark.apache.org/`

[6] A. Gellrich "*Job schedul in Grid batch farms*", 2014, Journal of Physics: Conference Series, volume 503, numer 3, page 032038, `doi:10.1088/1742-6596/513/3/032038`

[7] Nordugrid *ARC CE 5* http://www.nordugrid.org/arc/releases/15.03u20/

[8] Apache Kafka "A distributed streaming platform" https://kafka.apache.org/

[9] Apache OpenWhisk "Open Source Serverless Cloud Platform -Executes functions in response to events at any scale" https://openwhisk.apache.org/

[10]OpenWhisk "OpenWhisk Actions" https://github.com/apache/openwhisk/blob/master/docs/actions.md

[11] Paul Millar for the dCache Team "Storage Events: distributed users, federation and beyond" https://www.dcache.org/manuals/workshop-2018-05-28-DESY/storage-events.pdf

[12] Paul Millar for the dCache Team "Macaroons and SciToken" https://www.dcache.org/manuals/workshop-2018-05-28-DESY/paul4.pdf

[13] IBM Spectrum Scale "Advanced storage management of unstructured data for cloud, big data, analytics, objects and more" https://www.ibm.com/us-en/marketplace/scale-out-file-and-object-storage

[14] P. Fuhrmann et al. "dCache, agile adoption of storage technology", 2012, Journal of Physics: Conference Series, volume 396, number 3, pages 032077, https://iopscience.iop.org/article/10.1088/1742-6596/396/3/032077

[15] Puppet "Open source Puppet provides tools to automate managing your infrastructure" https://puppet.com/docs/open-source-puppet

[16] Puppet: Hiera https://puppet.com/docs/puppet/latest/hiera_intro.html

[17] Arnar Birgisson and Joe Gibbs Politz and Ulfar Erlingsson and Ankur Taly and Michael Vrable and Mark Lentczner "Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud",2014, Network and Distributed System Security Symposium https://research.google/pubs/pub41892/

[18] Singularity "Enabling and securing your performance critical applications from the core, through the cloud, and out to the edge." https://sylabs.io/docs/

[19] CVMFS "The CernVM File System provides a scalable, reliable and low-maintenance software distribution service." https://cernvm.cern.ch/portal/filesystem

[20] Gitlab Secrets https://docs.gitlab.com/charts/installation/secrets.html