ARC Control Tower
A flexible generic distributed job management framework

J.K. Nilsen[1], D. Cameron[2] and A. Filipčič[3]

[1] University Center for Information Technology, University of Oslo
[2] Department of Physics, University of Oslo
[3] Department of Experimental High Energy Physics, Jozef Stefan Institute

E-mail: `j.k.nilsen@usit.uio.no`, `david.cameron@cern.ch`, `Andrej.Filipcic@ijs.si`

Abstract. While current grid middleware implementations are quite advanced in terms of connecting jobs to resources, their client tools are generally quite minimal and features for managing large sets of jobs are left to the user to implement. The ARC Control Tower (aCT) is a very flexible job management framework that can be run on anything from a single users laptop to a multi-server distributed setup. aCT was originally designed to enable ATLAS jobs to be submitted to the ARC CE. However, with the recent redesign of aCT where the ATLAS specific elements are clearly separated from the ARC job management parts, the control tower can now easily be reused as a flexible generic distributed job manager for other communities. This paper will give a detailed explanation how aCT works as a job management framework and go through the steps needed to create a simple job manager using aCT and show that it can easily manage thousands of jobs.

1. Introduction
Grid middleware implementations such as Unicore[1], gLite[2], Condor[3] and NorduGrid's Advanced Resource Connector[4] (ARC) are quite advanced in connecting jobs to various resources. However, their corresponding client tools are quite minimal with respect to job management. Jobs can be submitted, monitored and fetched either manually through simple Command Line Interfaces (CLI's) or using more powerful and, hence, more complicated Application Program Interfaces (API's). The actual implementation of job management systems have been left to the users and at the Large Hadron Collider (LHC) this has led to implementations such as DIRAC[5] for LHCb[6] and PanDA[7] for ATLAS[8]. While these represent very capable workflow managers they have to be generic enough to handle a large variety of grid middleware and batch systems and are not optimally tuned to any specific middleware. Additionally they are both tuned towards pilot-based workload management (see, e.g., [9]), something which is not suitable for resources such as supercomputers and volunteer computing[10, 11] where Wide Area Network access from worker nodes is restricted or prohibited.

While ARC is very well tuned towards these kinds of resources it is, by design, not well suited for pilot jobs. One of the main principles is that no ARC software should be installed on the worker node and input and output data should always go through the Compute Element. The ARC Control Tower (aCT) was at first implemented due to the need for ARC to handle pilot jobs from ATLAS, by picking up pilot jobs from PanDA and submit their workloads to ARC Compute Elements (CE's) while making sure PanDA got the required life signs from the pilot jobs. By now the design of aCT has evolved and the dependencies on ATLAS and ARC are clearly separated. As will be explained in more details in Section 2 and exemplified in Section 3, aCT is a full-blown job management framework with proven capabilities to handle $\mathcal{O}(100k)$ ATLAS jobs per day and with clear use-cases for being reused as a generic distributed job manager for communities even outside the world of High Energy Physics (HEP).

2. Design of the ARC Control Tower

A high-level overview of the ARC Control Tower (aCT) is shown in Figure 1. aCT, represented by the turquoise box in the middle, is connected to an external job provider to the left and resources to the right. The workflow is as follows: aCT pulls job descriptions from the external job provider and translates the descriptions to XRSL[12], a job description language understood by ARC CE's. Next, aCT pushes the job descriptions to ARC CE's and monitors the states of the jobs until finished, fetches the jobs and push the required information back to the external job provider. During the runtime of a job, aCT communicates with the job provider to update job heart beats and check if the job should be cancelled. If a jobs fails, aCT checks if the failed state is due to a known error on the resource side, meaning that the job can be automatically resubmitted, or if the failure is likely to be caused by the job itself, in which case the failed state will be propagated back to the external job provider.

At the basis of aCT is the database with the ARC table. This is where all data concerning jobs and job states are stored. This table is used by the application engine to feed the ARC engine with new jobs, by the ARC engine to keep track of all job states from submission to finished jobs and finally by the application engine to know when to inform the external job provider that the job is finished. Additionally, the application engine may require a separate table within aCT to keep track of the job states as seen by the external job provider.

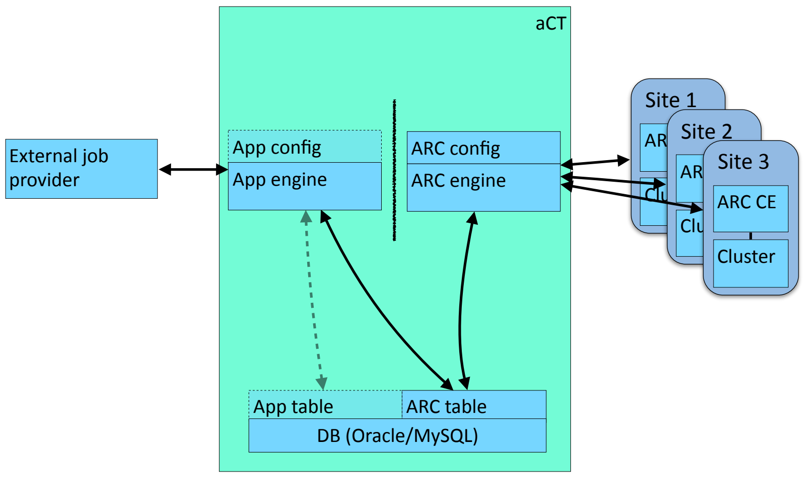


Figure 1. Overview of a general aCT setup. An application engine pulls job description from external job provider, converts the description to a job description format supported by ARC and stores it in a database. The ARC engine takes the converted job description and submitts it to ARC enabled sites and takes care of managing the job until finished. When jobs are finished the ARC engine fetch the jobs, and after the application engine has verified the job results the ARC engine cleans the jobs from the system.

The major part of the job management takes place in the ARC engine. The ARC engine is responsible for the communication with ARC CE enabled sites and the management of the jobs from the point of submission to the final state. For all the communication with ARC sites it uses the ARC Software Development Kit (SDK)[13]. The SDK consists of credential, compute, data, data staging and common libraries, whose API's are mostly high-level interfaces. In aCT the SDK is used for brokering, submission, cancelling, restarting, retrieving and cleaning of jobs and to get updates on job states.

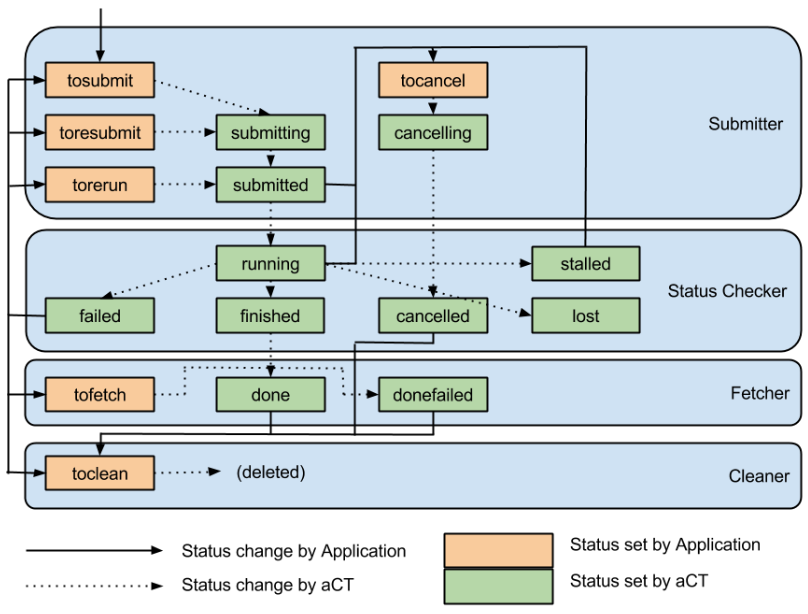


Figure 2. Overview of the ARC engine states and the actors governing them.

A job is a stateful entity. To manage a job, one needs to keep track of the state of the job and make sure it moves from one state to another. For example, a job with status *submitted* should either move to *running* or, if it does not start (or was mistakenly submitted) to *cancelling*. As shown in Figure 2, the ARC engine consists of a set of actors, the **Submitter**, the **Status Checker**, the **Fetcher** and the **Cleaner**. Each of the actors has responsibility for jobs in given states and moves them to the next state as follows:

- The **Submitter** picks up jobs that are to be submitted, resubmitted or rerun and moves them to the state of *submitting* and/or *submitted* by submitting them to ARC CE's. Additionally the submitter takes care of jobs to be cancelled.
- The **Status Checker** looks at jobs that are *submitting*, *running* and *cancelling* and, depending on information from the ARC CE's, move them to *running*, *finnished*, *stalled*, *lost* or *cancelled*.
- The **Fetcher** checks jobs that are *finished* and moves them to *done*. All successful jobs (i.e., jobs in *finished* state) are fetched per default while *failed* jobs are only fetched if the application engine sets their state to *tofetch*.
- Finally, the **Cleaner** picks up jobs that the application engine has decided can be deleted and removes them from the system.

One important consideration when it comes to the actors is that they need to communicate with many sites, an unpredictable operation on a Wide Area Network. In the event of a network issue somewhere between the aCT server and the site, an actor can get stuck waiting for a considerable time. For this reason the ARC engine uses one set of actors for each site so that the actors for the other sites can continue undisturbed.

3. Application Use-cases

The modularity and generic design of aCT makes it straight forward to adapt it to numerous use-cases. In this section three use-cases will be described to give an idea of the possibilities of aCT; a simple example to get the basic idea of aCT, a more complex HEP example and a web-portal based example from Life Sciences.

3.1. Simple Job Manager

The first use-case can be related to simple statistics gathering or Monte Carlo type simulations, mutually independent jobs with limited input and output data that should be run in large numbers. This kind of jobs may sound trivial to run on the grid. One scientist can easily implement a script to submit millions of jobs to the grid using the CLI tools available through any of the available grid middleware implementations. However, when running millions of jobs on the grid, some jobs will fail, some jobs will be stuck in long queues, some sites will go down for maintenance and some jobs might fail due to hardware issues and need to be resubmitted. To handle all these job states can easily be a very complex task when using CLI tools.

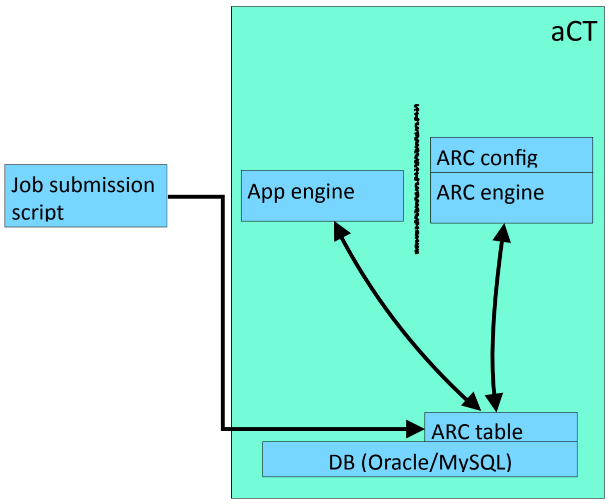


Figure 3. Minimal setup of aCT. A job script inserts job descriptions directly to the ARC table, no application table is needed.

However, as was shown in Section 2, this is exactly what aCT takes care of. Figure 3 shows a minimal setup of aCT corresponding to this use-case. Here, a submission script inserts XRSL job descriptions directly into the ARC table and the ARC engine takes care of most of the job states automatically. However, recalling the job state overview of Figure 2, some job states need to be set by the application. In particular, it is up to the application/user to decide if a job should be cancelled, if a failed job should be fetched and if a job should be cleaned from the system. In our simple example, a failed job only causes slightly decreased quality of statistics, so it is sufficient to check for *failed* states and move them to *tofetch* for later investigation and, when they are fetched, move them to *toclean* to have the ARC engine remove them from the system. Additionally, the application engine should have an actor to validate successful jobs and decide if they can be cleaned from the system. In our simple case these are the only features that need to be implemented to have a functional job manager.

3.2. ATLAS Application
At the other end of the complexity scale is the use-case of ATLAS. This use-case is currently in production to serve ATLAS with a wide variety of resources, from high-performance computers to volunteer computing [14, 15, 16, 11] and is described in more details in [10].

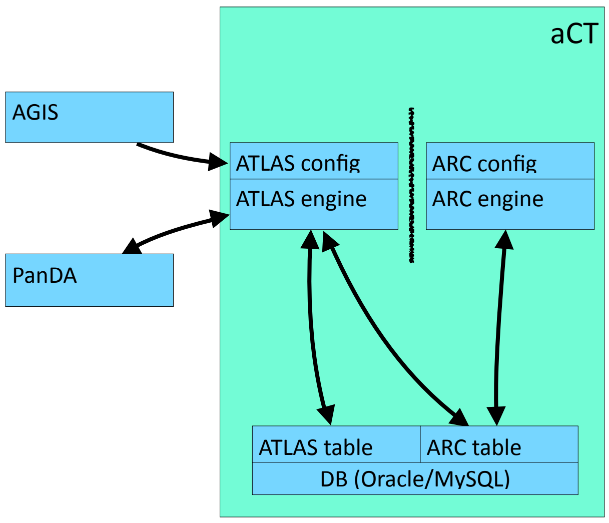


Figure 4. ATLAS aCT setup. The ATLAS engine pulls job descriptions from PanDA together with site information from AGIS. A separate application table is used to store PanDA job states, internal application specific jobstates and PanDA heartbeats.

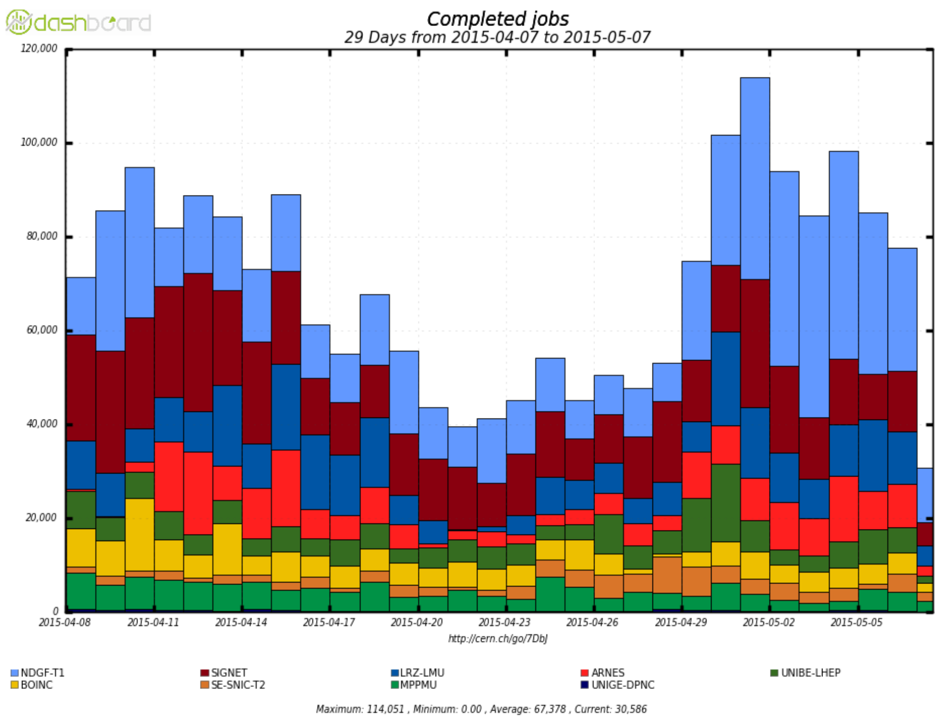
Figure 4 shows the current setup of aCT for ATLAS. Here PanDA is used as the external job provider while AGIS [17] is used to gather information about the sites to be used. In addition to the ARC engine there is an ATLAS engine that takes care of the communication with PanDA, update heartbeats for the jobs, and converts ARC job states to corresponding PanDA job states. Since the ATLAS engine needs to store a separate set of job states it has its own table in the aCT database and uses the ARC table to check on jobs submitted to ARC CEs. Note that there is no connection between the ARC engine and the ATLAS table; the ARC engine need not know anything about PanDA job states.

To give an idea of the load on aCT when used for ATLAS, Figure 5 shows two plots; on the left hand a snapshot of completed jobs per day and on the right a snapshot of the slots of running jobs, both sorted by ARC site during the period 7 April to 7 May 2015. As shown in the plot to the left, aCT completed up to 115,000 jobs in a single day. However, while the number of jobs per day gives some idea of the load on the aCT database, it tells little about the number and type of resources used. The figure to the right shows that aCT serves around 20,000 CPUs or cores on a wide variety of resources, such as volunteer computing (BOINC), pledged WLCG resources (e.g., NDGF-T1) and HPC resources (e.g., LRC-LMU).

A feature of aCT worth mentioning here is the way aCT fills the available resources. Since aCT is intended to run jobs on resources that are shared with other users, the number of slots available per site can vary a lot from day to day. Hence, it is not possible to find an optimal number of jobs to submit to one site on any given day. Instead, aCT looks at the number jobs running on a site at any given time and makes sure that the number of jobs in queue is limited as shown in Equation 1.

$$N_{queued} = aN_{running} + b \tag{1}$$

Here, $N_{running}$ is the number of jobs in state *running* and N_{queued} is the upper limit on the



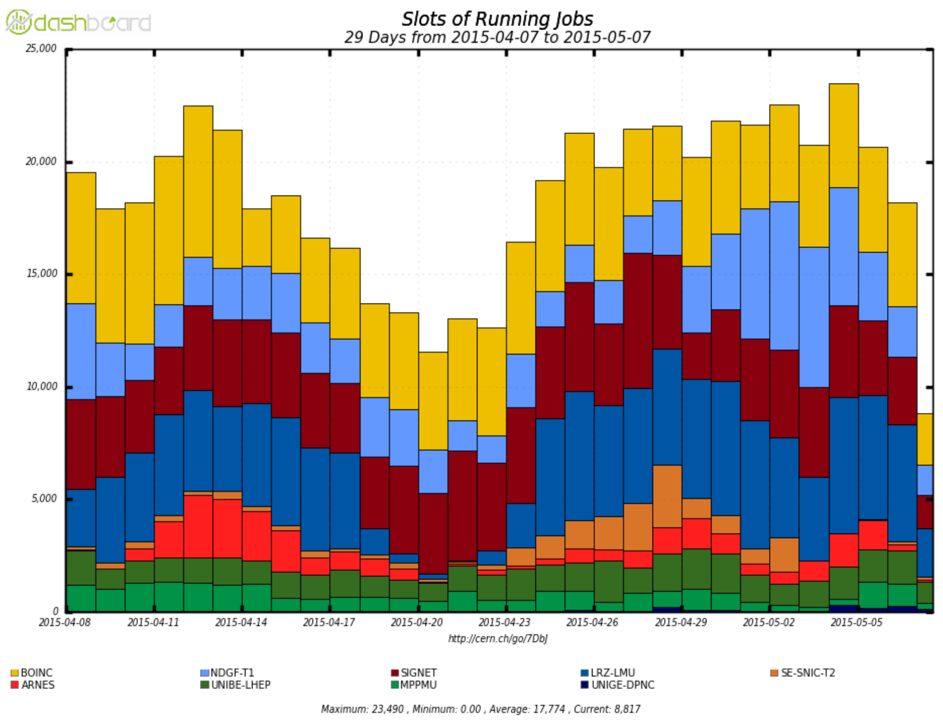


Figure 5. Snapshots of completed jobs per day (top) and slots of running jobs (bottom) sorted by site running ARC CE.

number of jobs in state *queuing*. This means that there are at least b jobs queueing on a site if it accepts new jobs. Additionally, the number of queued jobs are kept to at least a times the number of running jobs, to make sure that there are enough jobs preparing and staging input data so that they are ready when running jobs finish. In current production, $a = 0.15$ and $b = 100$ as experience has shown this to yield a reasonably good job throughput while reasonably sized queues.

3.3. Load Balancing Norwegian HPCs

While the last example is still on the planning-stage it shows one of the key bennefits of aCT; the job brokering capability. ELIXIR.NO is the Norwegian node to ELIXIR [18], aiming to provide resources to bioinformatics users through the Norwegian e-Infrastructure for Life Science (NeLS) project. Figure 6 shows the member institutions at the bottom and the geographic distribution of the resources to the left. To give access to the resources each of the shown sites have installed or are planning to install Galaxy [19], an open, web-based portal for data intensive biomedical research. However, Galaxy only supports one cluster per instance, meaning that no load-balancing between sites is available out of the box. As shown to the right in Figure 6 a possible solution to this problem is to introduce one or more aCT instances in the setup so that Galaxy portals at each site submits jobs through aCT which in turn submits the jobs to ARC CEs at the different sites, depending on the resource availabilities at the sites.

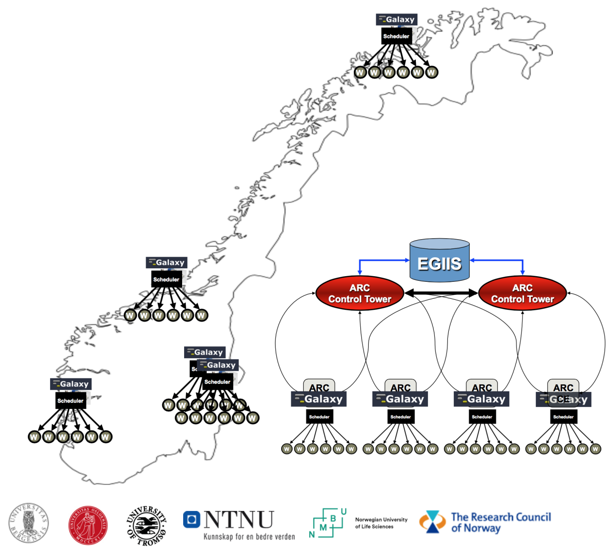

Galaxy
Scheduler
EGIIS
ARC Control Tower
ARC Control Tower
ARC
NTNU
Kunnskap for en bedre verden
Norwegian University of Life Sciences
The Research Council of Norway

Figure 6. Envisioned use of aCT in the ELIXIR.NO project, showing how aCT can be used as a broker for 5 HPC sites in Norway.

Figure 7 shows in more detail the possible setup for running Galaxy jobs through aCT. The red boxes highlight where changes to Galaxy and aCT are needed. Here an aCT plugin is added to Galaxy to allow users to submit to aCT. This plugin stores job descriptions in a separate aCT Job database controlled by Galaxy and the Galaxy engine within aCT pulls jobs from this database and adds them to the ARC table. Compared with the ATLAS setup in Figure 4 the main difference is that the application table is no longer governed by aCT itself, but rather by multiple Galaxy instances.

4. Conclusion and Future Directions

This paper has outlined the features and the architecture of the ARC Control Tower. Three example setups have been given, showing use-cases where aCT can be used as a flexible generic distributed job manager, from the simplest case to the currently in-production case and showing a potential future use-case of aCT as a national job broker.

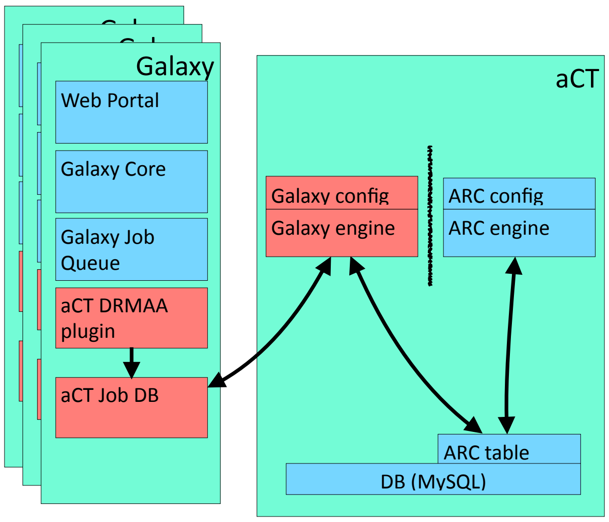


Figure 7. Envisioned Galaxy aCT setup. Multiple Galaxy instances write job descriptions to local databases that the Galaxy engine pulls into aCT. When jobs are finished, aCT updates the corresponding Galaxy databases.

Acknowledgements
This work supported in parts by the Nordic e-Infrastructure Collaboration (NeIC).

References
[1] Romberg M. The UNICORE Grid Infrastructure. *Sci. Program.*, 10(2):149–157, April 2002.
[2] Laure E et al. Programming the Grid with gLite. In *Computational Methods in Science and Technology*, page 2006, 2006.
[3] Thain D, Tannenbaum T, and Livny M. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
[4] Ellert M et al. Advanced Resource Connector middleware for lightweight computational Grids. *Future Gener. Comput. Syst.*, 23(1):219–240, 2007. doi: 10.1016/j.cam.2006.05.008.
[5] `http://diracgrid.org`. DIRAC (Distributed Infrastructure with Remote Agent Control).
[6] Alves A A et al. The LHCb Detector at the LHC. *JINST*, 3:S08005, 2008.
[7] Maeno T et al. The Future of PanDA in ATLAS Distributed Computing. *J. Phys.: Conf. Ser.*, 2015.
[8] Aad G et al. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST*, 3:S08003, 2008.
[9] Sfiligoi I. glideinWMSa generic pilot-based workload management system. *Journal of Physics: Conference Series*, 119(6):062044, 2008.
[10] Filipčič A, Cameron D, and Nilsen J K. Dynamic Resource Allocation with the arcControlTower. *J. Phys.: Conf. Ser.*, 2015.
[11] Cameron D et al. ATLAS@Home: Harnessing Volunteer Computing for HEP. *J. Phys.: Conf. Ser.*, 2015.
[12] Globus Alliance. Extended Resource Specification Language (XRSL). Technical report, Globus Alliance, 2009.
[13] Andersen M S, Cameron D, and Lindemann J. ARC SDK: A toolbox for distributed computing and data applications. *J. Phys.: Conf. Ser.*, 513(3):032015, 2014.
[14] Sciacca F G et al. The ATLAS ARC ssh back-end to HPC. *J. Phys.: Conf. Ser.*, 2015.
[15] Hostettler M, Filipčič A, and Walker R. ATLAS computing on the HPC Piz Daint machine. *J. Phys.: Conf. Ser.*, 2015.
[16] Mazzaferro L et al. Bringing ATLAS production to HPC resources - A use case with the Hydra supercomputer of the Max Planck Society. *J. Phys.: Conf. Ser.*, 2015.
[17] Anisenkov A et al. AGIS: The ATLAS Grid Information System. *J.Phys.Conf.Ser.*, 396:032006, 2012.
[18] `http://www.bioinfo.no/elixir`. ELIXIR Norway.
[19] Goecks J et al. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, 11(8):R86, 2010.