A Computational and Data Scheduling Architecture for HEP Applications

A. Papaspyrou, L. Schley, R. Yahyapour,
Information Technology Division, Robotics Research Institute, University of Dortmund, Germany
M. Ernst, P. Fuhrmann, M. Radicke,
DESY, Hamburg, Germany

Abstract

This paper presents a new approach on extending the LCG system architecture to enhance job scheduling for data intensive applications in the High Energy Physics (HEP) community. First, a brief introduction to the current LCG/gLite-based implementation is given, and its inherent problems are discussed. Then, an extended architecture is proposed which adds coordinated scheduling capabilities on top of the existing middleware for both computation and storage nodes, explaining their enhancements for scheduler interaction to improve the overall turnaround times and system throughput. Here, standard Grid components for the computational part and the dCache software package for data management are used. Finally, an outlook is given on how this scheduling framework can be adapted to other application scenarios like e.g. the climate research community, reaching a broader acceptance in Grid communities beyond the HEP community.

The HEPCG project is funded by the German Ministry of Education and Science as part of the national e-science initiative D-Grid and is jointly carried out by the Robotics Research Institute (Section Information Technology) at University of Dortmund and DESY.

INSIDE LCG

The LCG system (`http://lcg.web.cern.ch/lcg/`) is a widely-used grid middleware system for large-scale computation tasks in the High Energy Physics community, partly based on the Globus Toolkit [6]. Its architecture depicts two basic resource types, that is computing elements (CEs) and storage elements (SEs). Typically, a CE consists of a batch management system implementation (e.g. Torque, `http://www.clusterresources.com`) and a local grid scheduler, which extends the CE to a schedulable resource. The SE is build on a storage management system – typically dCache [4] –, which provides extensive information and management services and seamless integration of hierarchical storage management systems (HSM) as a back-end solution. These components and their integration into the overall service architecture of LCG are depicted in Figure 1.

A typical task in the LCG system is a long-running computation job which depends on input data and creates output data, both often very large. To submit a task to the grid, the user must create a specification for it, describing certain CE requirements (processor architecture, memory size, etc.) and data dependencies (needed files, output data size etc.). For this purpose, the LCG environment provides the Job Description Language (JDL) [7]. Such job descriptions (along with other data) are passed to a Resource Broker (RB) [8], the current workload management system (WMS) in the LCG infrastructure. The RB analyzes the task's JDL specification and runs a match-making algorithm to find an appropriate CE for its execution. However, a rather simple approach is used here: basically, the selection of the CE depends on the "nearness" to a SE with the required data—if a SE on a certain site holds the required files, the algorithm will select the CE on the same site for computation. However, this rather loose coupling of job management and data management arises several problems.

Because data transfers are considered expensive (as shown in the previous example, they are avoided at all cost), the turnaround time of a job can be very long. For example, if a job is submitted to a CE with an overfull queue, the waiting time from enqueuement to completion can exceed the waiting time for a network transfer of the data to an idle CE site with an appropriate SE. Obviously, this approach leaves room for improvement: if replica management would be taken into account during the matchmaking process, the turnaround time for jobs could be reduced.

Another point is the use of HSM systems as a backend to dCache. In many cases, HSMs utilize tertiary storage systems like robot-driven tape libraries/silos. Access to files which are stored on tape is considerably slower than reading equivalent files from disk, with speed differences in orders of magnitude. To this end, systems like dCache are introduced, which allow levelled access to the HSM, providing large hard disk caches for faster access. However, in the dCache system, *all* files appear as available from disk at any given time. Only on file access, the storage location (disk or tape) can be determined. Thus, the opening system checks whether the file is already on disk and, if not, restages it from tape. Hence, if a computation job reaches the head of its CE's queue and is started, it will hang until the file is completely restored from tape. For space-shared CEs, the remaining jobs in the queue have to wait despite the partition used by the job staying idle and waiting for I/O.

Apparently, the lack of interaction between RB, CE and SE and the absence of advanced reservation strategies on the data and computation nodes leave room for improvement. To this end, we propose the following extensions to the LCG system to address these issues.

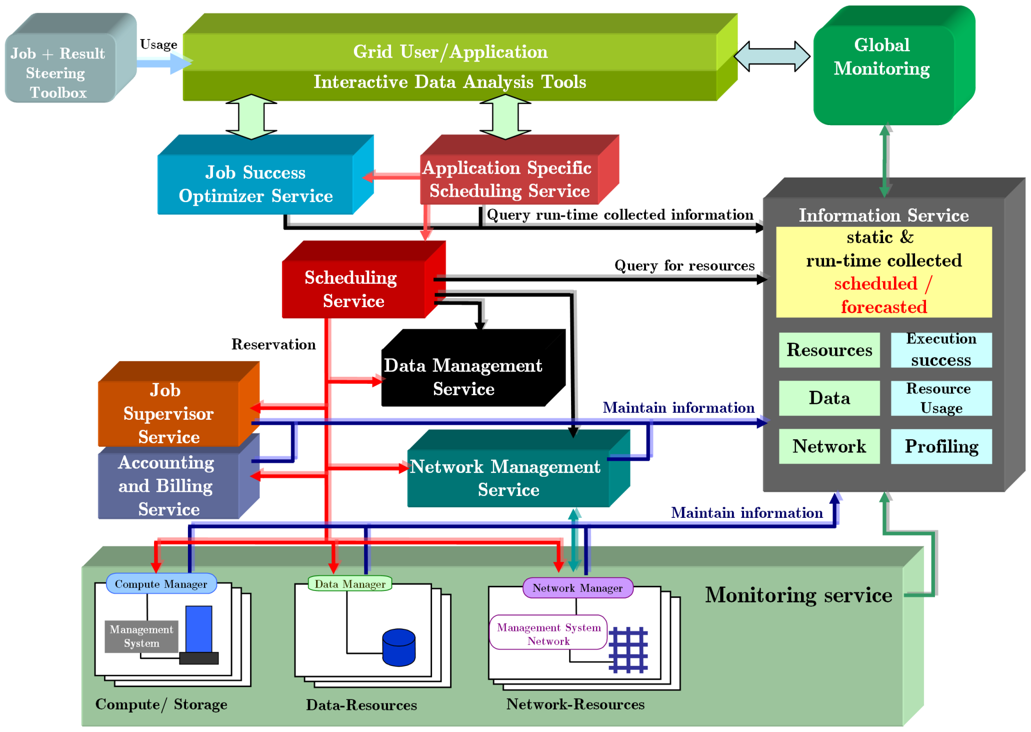

Figure 1: LCG Services.

EXTENSION POINTS

It is clear that, in grid environments, computation and data requirements of jobs are closely coupled: the input data must be available prior to job execution, and a certain amount of free space must be available during and after job computation. Therefore, both data and compute resources must be scheduled in a more comprehensive way. That is, file availability *and* accessibility must be known or at least predictable prior to job allocation and, furthermore, local (on-site) job and data scheduling must be integrated more tightly. To this end, the RB must be extended to a broader Workload Management System (WMS), which in turn makes use of several other services to accomplish the aforementioned goals.

Components

First, a metadata catalogue is used for accessing static, pre-scheduled or forecasted/estimated information on the usage of CEs, SEs and their available services. For this purpose, the Metadata and Discovery System (MDS) [1] of the LCG environment is used. The MDS provides data on registered resources like CEs, SEs and other services in a centralized directory. Here, information on the type of a resource can be obtained, for example the processor architecture of a CE. Thus, the MDS can be used for the matchmaking process in the WMS. Still, dynamic information like a CEs queue length must be obtained elsewhere, e.g. from a local job scheduler component on the CE itself.

Second, a centralized file catalogue is necessary to determine the availability and, if applicable, the location of data sets and their replicas within the grid. This is accomplished by the LCG File Catalogue (LFC), which provides a global file system namespace with logical file names (LFNs) for the data sets and resolves them to physical locations (PFNs) on data resources (SEs, that is). This functionality is needed to locate input data for a computation job in order to execute it on-site or to create a replica elsewhere. Then, however, the job itself must be postponed until the replication process is completed. Contrary to the current RB strategy of executing the job always on-site, the proposed WMS utilizes the LFC for relocation of data to an off-site CE.

Third, it must be possible to make advance reservations for computational jobs on such an off-site CE. However, the currently used GRAM [2] does currently not support this functionality at all: jobs which arrive at CEs are immediately enqueued in the underlying batch management systems without checking whether all job constraints (such as input file availability) can be satisfied. Therefore, a local job scheduler must be introduced to provide reservation functionality. To this end, a two-step negotiation process will be used: in a first step, offers for computation are requested by the WMS. Then, an obligatory agreement on a certain offer is made. However, the underlying batch management system must have at least basic support for advanced reservation. Otherwise, agreements can be fulfilled on a best-effort basis only. During this negotiation process, the WMS must also take data management into account, as necessary input files can be currently unavailable and must

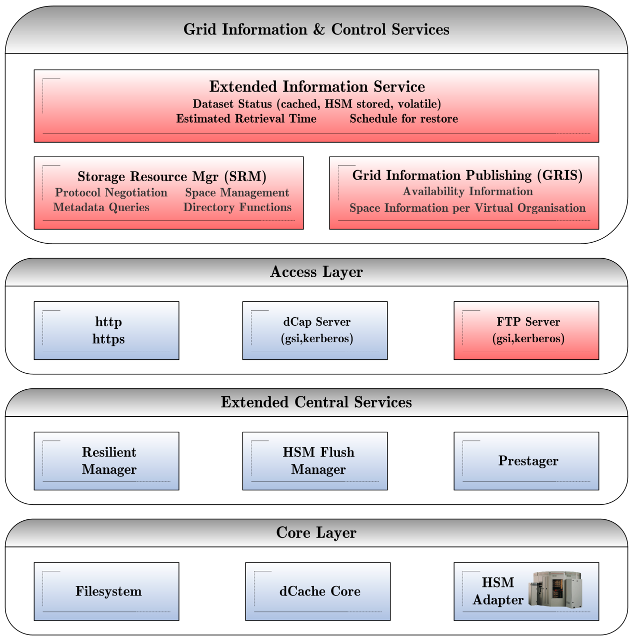

Figure 2: Internal architecture of dCache

be replicated in the first place.

Forth, data management has to cooperate with both the WMS and the job scheduler at each CE in terms of providing estimations on file staging times, reserving space and managing data transfers. The current SEs do not support this at all, neither do most of the underlying HSMs. To address the aforementioned problem of visible, but unavailabe files, a local data scheduler will be established which manages file recovery and space reservation from tertiary storage and a prediction engine which provides information on file status and expected recovery time from tertiary storage. Therefor, the dCache system [3] (cf. figure 2) will also be extended to provide more detailed information on a file's current status. Then, the local job scheduler will be able to determine the start time for a task depending on the availability of its required input data. Again, the prediction engine highly depends on the underlying HSM functionality and will in the worst case only allow best-effort services.

Orchestration

A job submission with data and space requirements for input and output to the new WMS can then be orchestrated as follows (cf. Figure 3). It is assumed that all CEs and SEs are appropriately registered in the MDS and all files can be resolved by using the LFC. Furthermore, the underlying systems (batch and data management) are expected to support the aforementioned advanced functionality. Note that it is not trivial to perform "good" matchmaking; the used strategy is a matter of policy in the scheduling algorithm. Here, one example strategy will be presented.

After receiving a job description, the WMS utilizes the MDS to determine a subset of CEs according to the user-specified requirements (e.g. processor architecture, number of nodes, main memory). Then, for every CE from this subset M_{CE}, the corresponding (on-site, that is) SE is checked whether the necessary files are already available. This is done by querying the LFC for the required files (or replicas of these) and matching the found SEs with the CE subset. Let this subset of SEs be M_{SE}. Now, the CEs are sorted ascending to their computation queue length, which is determined by utilizing the GRAM and the local job scheduler. The SEs are queried whether the presented files are accessible immediately or if a restaging process is necessary first, which is done by using the extended dCache interface and the local data scheduler. At this point, the scheduler could directly select the CE with the shortest queue length and immediate file availability at the on-site SE and submit the job to it. Let this CE be C_1.

However, depending on the waiting time $t_w(C_1)$ of the aforementioned CE (depending on the queue length l_q), this is not necessarily the best selection possible. Since l_q can be arbitrarily large, those SEs should be taken into account where the file is not on disk and calculate the restaging time t_r. Now, if $t_w(C_1) > t_r(S_n), S_n \in M_{\text{SE}}$, it might be beneficial to select one of those CEs where the file on the SE is not already available, but restaging time is short. To this end, the matching SEs are also sorted ascending to their restaging time. Then, the the pair (C_k, S_k) with the shortest overall waiting time will be selected. The job can then be submitted to C_k, and during the CE wait time, the restaging process is started on S_k.

Obviously, this solution is still not optimal, taking into account that the estimations made in the described process (e.g. file restoration time) are not accurate—this will of course have a negative impact on the quality of the schedule. Also, *creating a new replica* on an idle CE/SE site could result in a shorter waiting time than those reachable with the approach from above. This would require to model networks between sites to be able to incorporate file transfer time estimates into the scheduler's calculations. However, in many realistic scenarios often no precise information about the service quality in networks is available. There are, of course, means which can ameliorate this situation. For example, the NWS system [9] measures and records the available bandwidth between two nodes periodically. This data is then used to predict the expected average bandwidth in the future based on historic patterns. In cases where reservation of bandwidth is not feasible, there are still possibilities to shape network traffic. Still, abiding agreements on an certain QoS level cannot be settled normally [5]. Regarding network latency, which is important for applications requiring large numbers of small network packets (e.g. streaming), the situation is akin. Besides that, there is always a certain amount of background (not grid workload-related, that is) traffic on a network, which lowers both bandwidth and latency. However, due to the lack of reservation capabilities, the impact of background traffic is not predictable at all. Even when predictions expect high network availability and the known future utilization is low,

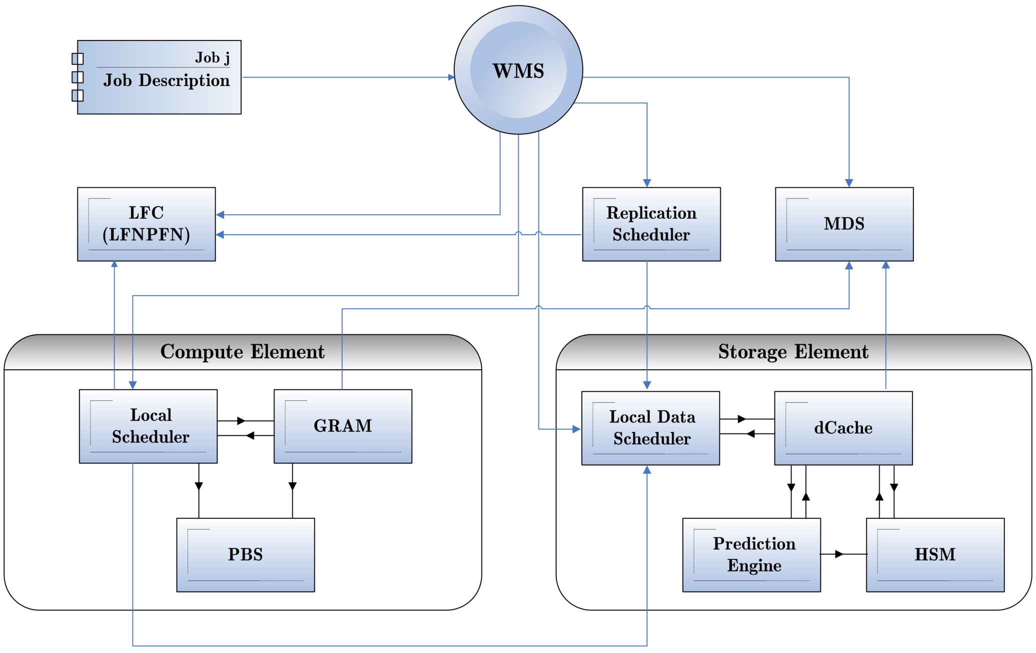

Figure 3: Extended HEPCG Scheduling Architecture.

a single data-intensive file transfer can suddenly invalidate a whole schedule.

CONCLUSION

Summarizing, we presented a new architecture for computational and data scheduling for HEP applications. First, we discussed the actual state of the LCG grid software and its drawbacks. Then, we proposed extensions to existing elements in order to develop schedulable resources which support advanced reservation. Also, the orchestration of the overall architecture was described.

Nevertheless, the proposed scheduling architecture is not restricted to high energy physics, but can also be adapted to other scenarios with data intensive HPC demands. For example, this is the case for the Collaborative Climate Community Data and Processing Grid (C3Grid, `http://www.c3grid.de`), which focuses on the exploration of the human living space to recognize and understand global climate coherences. To this end, complex models are used to describe and simulate earth's behavior, in turn detailed input is used, which can cover large spaces of time and high precision data generated by satellites. Concluding, the C3Grid community has similar computational and data demands as the HEP community, so that our proposed architecture and techniques could be adopted.

REFERENCES

[1] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselmann. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the 10th IEEE International Symposium on High-Performance Distributed Computing*, San Francisco, August 2001. IEEE Computer Society Press.

[2] Karl Czajkowski, Ian Foster, Carl Kesselman, and et. al. A Resource Management Architecture for Metacomputing Systems. In Dror G. Feitelson and Larry Rudolph, editors, *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82, Orlando, March 1998. Springer-Verlag.

[3] Mathias de Riese, Patrick Fuhrmann, Michael Ernst, Alex Kulyavtsev, Vladimir Podstavkov, and Martin Radicke. dCache, The Book. Technical report, Deutsches Elektronen-Synchrotron, 2003.

[4] Michael Ernst, Patrick Fuhrmann, Martin Gasthuber, Tigran Mkrtchyan, and Charles Waldman. dCache, a distributed data storage caching system. In *Proceedings of Computing in High Energy and Nuclear Physics*, pages 101–104, Beijing, September 2001. Science Press.

[5] Ian Foster, Markus Fidler, Alain Roy, Volker Sander, and Linda Winkler. End-to-End Quality of Service for High-End Applications. *Computer Communications*, 27(14):1375–1388, September 2004.

[6] Ian Foster and Carl Kesselman. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprint for a Future Computing Infrastructure*, pages 259–278. Morgan Kaufman, San Mateo, 1st edition, 1998.

[7] Fabrizio Pacini and Stefano Beco. JDL Attributes. Technical report, European Data Grid, October 2003.

[8] Antonio Retico. LCG Resource Broker – Generic Configuration Guide. Technical report, LHC Computing Grid, October 2005.

[9] Rich Wolski, Neil Spring, and Jim Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, October 1999.