

From Clusters to the Fabric: The Job Management Perspective*

Thomas Röblitz, Florian Schintke, Alexander Reinefeld
Zuse Institute Berlin (ZIB)
Takustr. 7, D-14195 Berlin, Germany
{roebnitz,schintke,reinefeld}@zib.de

Abstract

Clusters provide an outstanding cost/performance ratio, but their efficient orchestration, i.e. their cooperative management, maintenance, and use, still poses difficulties. Moreover, many sites operate multiple clusters, each possibly running under a different cluster management system.

In this paper, we present an architectural scheme for the coordinated management of multiple clusters in a fabric. Our scheme allows different cluster management systems to interact with each other via adaptors, thereby providing interoperability within a single administrative entity, the fabric. Using adaptors, various jobs (grid jobs, local jobs, system maintenance jobs) can be served by the same methods, and existing cluster management software (like LSF, PBS, CCS, etc.) can be extended by additional functions without much modification effort.

1. Introduction

Clusters received much attention during the past few years and one should therefore think that their use and operation is a well-understood issue by now. However, some large-scale projects like Cern's LHC experiments or life science and earth observation projects require resources in a scale that was never reached before [1]. In respect to their excessive demands on the computing infrastructure these projects are equipped with a relatively low budget for computing, storage, and networking. Clusters made of common-off-the-shelf technology provide a welcome alternative to costly high-performance computers. The vastly grown requirements, however, clearly identify the neuralgic shortcomings in the current state-of-the-art of the cluster technology, namely the lack of scalability, poor fault tolerance, and high maintenance overhead [8].

Many clusters are not used as standalone systems, but are integrated in a larger computing fabric, that is an aggregation of clusters. Clusters are often incrementally installed or upgraded over time, resulting in heterogeneous hardware, system software, and application tools, making it difficult to manage them in a consistent way.

At the next higher level, many farms are used as computing nodes in a Grid environment. Here the mentioned problems become even more obvious, because the single clusters should (ideally) function autonomously without human intervention. The automated operation is not only necessary for reducing human administration effort, but also to limit possible sources of errors.

Figure 1 illustrates a scheme for the automatic fabric maintenance that we jointly developed and implemented with other researchers in the fabric management workpackage of the European *DataGrid* project¹. A fabric comprises several separate clusters with (possibly) different cluster management systems. Three kinds of jobs may enter the fabric: grid jobs (from the grid level above), local user jobs (injected from the left), and maintenance jobs (injected by the system itself).

Status information on the clusters' target configuration, the *goal state*, is stored in a configuration database. The *actual state* is obtained by monitoring tools. Both states are compared in the *Fault Detection and Recovery System*, which checks for mismatches and initiates the necessary maintenance actions to fix them. Note that these actions are passed via the *Installation System* to the *Job Management* which schedules the maintenance jobs just like any other ordinary user job—but with specific requirements, of course. In addition, local user jobs may be submitted for execution on specified clusters, or grid jobs may be injected from remote via the *Grid Access* component.

For the described scenario, a fabric management system must cope with additional tasks that are not commonly found in today's cluster management systems:

* This research was funded by the EU DataGrid project and by an IBM Faculty Award to the third author.

1 <http://www.eu-datagrid.org/>

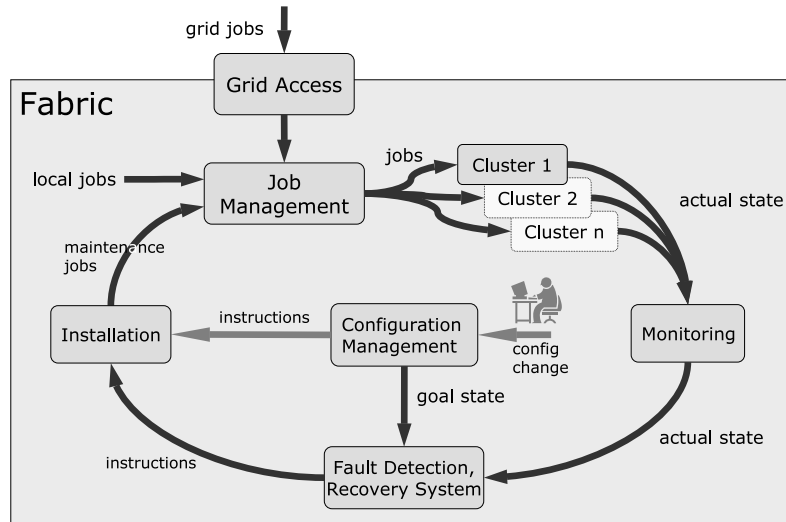


Figure 1. Management cycle for the automated maintenance of clusters in a fabric.

- the interaction with different cluster management systems,
- the support of jobs coming from various sources,
- the provision of additional services for the grid layer.

The remainder of this paper is organized as follows. In the next section we review similar work by others. Thereafter we present the architecture of our fabric management system and discuss its practical implementation in Section 4. Performance results are shown in Section 5. We conclude the paper with a summary of our results.

2. Related Work

The Maui² scheduler [2] provides scheduling algorithms and static configuration features required for our system. Some cluster management systems already provide hooks to integrate external schedulers like Maui, but the integration is done differently for each system. Our system, in contrast, interfaces to various cluster management systems and schedulers via adaptors, without the need to implement separate adaptation interfaces for each combination.

A variety of cluster configuration suites, like OSCAR³, SCMS [9], NPACI Rocks [6] exists, but none of them supports an automated cluster management cycle or the scheduling of maintenance actions.

Additionally our system allows to filter and annotate status information for the flexible integration with other cluster computing services, thereby providing the basis for data-aware scheduling or system re-configuration on demand.

This would be difficult to do if only the Maui scheduler is used.

3. Architecture of a Consistent Fabric Management System

Most cluster management systems consists of three components:

- a *server* that receives job requests and provides a central access point to clients, users and the scheduler,
- a *scheduler* that implements access policies, requirements matching, and determines an efficient execution order,
- a *node manager* that reports the status of the nodes to the server and controls the node access.

In most systems, the scheduler interacts with the server to retrieve status information from the nodes to make its scheduling decisions. Sometimes, the scheduler interacts with the node managers directly to obtain more detailed status information that would not be available from the server. One such example is the combination of OpenPBS with the Maui scheduler.

Our job management system assumes that the scheduler only interacts with the server. We illustrate different aspects of this interaction in the following sections. Section 3.1 discusses how a specific scheduler can interact with a specific server. Thereafter in Section 3.2 we use this approach to manage several clusters with one scheduler. Section 3.3 discusses models for scheduler replacement. With our approach scheduling features can be added to cluster management systems in a non-intrusive way. We describe this in

2 <http://www.supercluster.org/maui>

3 <http://oscar.sourceforge.net/>

Section 3.4 at the hand of scheduling features that are missing in several cluster management systems.

3.1. Introducing an Abstraction Layer between Scheduler and Server

Connecting the scheduler directly to the server, as shown in the left hand side of Fig. 2, gives the best interaction performance because status information and job commands are sent directly without passing other components. The disadvantage is, however, that servers and schedulers must be adapted to interact with each other, resulting in $c \times s$ code adaptations for c cluster management systems and s schedulers. This makes the development of middleware on top of cluster management systems and the development of generic extensions for these systems a tedious task. To avoid

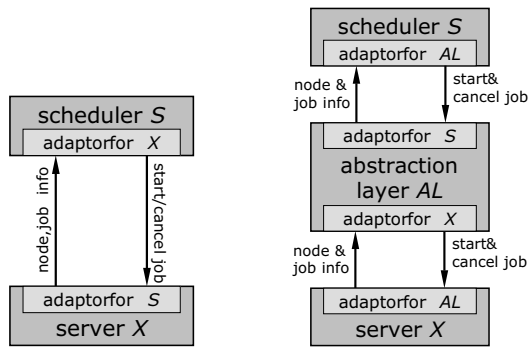


Figure 2. Left: Direct connection between scheduler and server. Right: Connection via an abstraction layer.

this, we introduced an *abstraction layer* (right hand part of Fig. 2) which reduces the customization efforts to $c + s$. Of course, the abstraction layer causes additional overhead, but it is low as shown in Section 5. When the source code of the server or the scheduler is not available, adding an abstraction layer may be the only possible solution. In addition, it helps hiding site-specific facilities or features like internal load balancing across clusters, resource brokerage, status information filtering, etc. The abstraction layer can be kept generic by using different adapters for different servers/schedulers.

3.2. Consistent Management of Multiple Clusters

The abstraction layer described in the previous section also allows to operate several clusters with a single scheduler as illustrated in Fig. 3. Here, job management actions

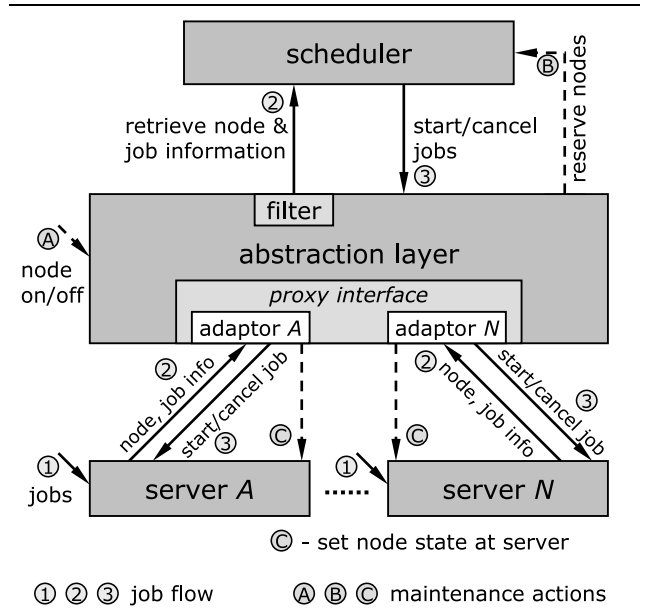


Figure 3. Support for job management and maintenance actions across several clusters.

are depicted by solid arrows while maintenance tasks use dashed lines.

Incoming jobs are submitted to the server (step 1). The scheduler periodically asks for the current status of nodes and jobs, and the abstraction layer gathers this information by sending requests to the server. When finished, it sends the (filtered) information back to the scheduler (step 2). The scheduler determines a schedule and sends the decision to the server through the abstraction layer (step 3).

Only four operations are necessary to link scheduler and server by an abstraction layer: start job, cancel job, node info, and job info. Although the set of operations seems to be obvious, its composition was motivated by the *Wiki interface* of the Maui scheduler.

- *start job* triggers the execution of a job on the assigned nodes.
- *cancel job* terminates the execution of a job.
- *node info* delivers status information on the nodes that the scheduler takes into account for job placement. Typically, the information consists of a node identifier, the status of the node, the number of processors, the available/maximal disk space or memory, the network connectivity, etc.
- *job info* retrieves status information about jobs. Usually, the information comprises an identifier, an owner, the jobs current state, the number of requested nodes, and the duration.

The execution of maintenance actions is illustrated by dashed arrows in Fig. 3. First, the abstraction layer receives a request to switch a node on or off (step A). If it accepts the request, it reserves the specified nodes for the given time interval (step B) and sets the status of the nodes during that interval to *stopped* at the servers (step C).

3.3. Using External Schedulers

Only one scheduler must be used to decide whether or when a job can be started. Hence the built-in scheduler of the cluster management systems must be disabled. Depending on the features provided by the cluster management system, we distinguish two models for replacing the built-in scheduler: the *disabled scheduler model* and the *execution queue model*.

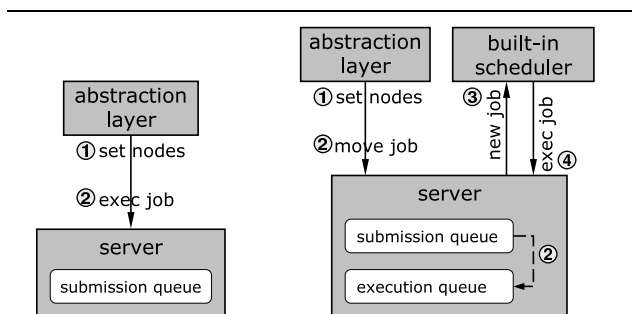


Figure 4. Left: Disabled scheduler model. Right: Execution queue model.

Disabled Scheduler Model. This model is the preferred one, because we only need to disable the built-in scheduler, as shown in the left hand part of Fig. 4. In addition, the cluster management system must be able to satisfy the following requirements:

- The server must provide a method for starting a specific job.
- The nodes where a job is executed can be defined via an interface.

To start a job, the abstraction layer (AL) sets the nodes where the job should start according to the information given by the scheduler in the *start job* command (step 1). Next, the AL triggers the job execution (step 2).

Execution Queue Model. When the built-in scheduler cannot be disabled the execution queue model shown in the right part of Fig. 4 is used. For this model, the following features are required by the cluster management system:

- It must be possible to set up an execution queue that is only accessible with administrator permissions.
- The submission queues must be able to be set on *hold* or *stopped* and the built-in scheduler only starts jobs in the execution queue.
- Jobs can be moved from submission queues to the execution queue with administrator permissions.
- The nodes to execute a job can be defined via an interface.

To start a job, the AL sets the corresponding nodes according to the information given by the scheduler in the *start job* command (step 1). Next, the AL issues a *move job* command to the cluster management system's server (step 2). In the next iteration, the built-in scheduler notices the 'new' job in the execution queue (step 3) and starts its execution (step 4).

Although, the model facilitates the replacement of the built-in scheduler, it requires specific features of a cluster management system. Moreover, because jobs are in hold-status first and then moved to another queue, users might get confused when viewing the queues. Retrieval of sound status information gets complicated, because several queues must be taken into account.

3.4. Extending the Functionality of Cluster Management Systems

Most cluster management systems, like the Portable Batch System (PBS) [3], LoadLeveler (LL) [4], Sun Grid Engine (SGE)⁴, Load Sharing Facility (LSF) [10], or the Computing Center Software (CCS) [5] provide a common set of scheduling features like fifo, backfill, etc. (Table 1). They only differ in their support for advanced scheduling capabilities like advance reservation. While users of stand-alone clusters may be able to cope with these insufficiencies, Grid users are forced to confine themselves to the common set of basic scheduling features. Our approach, in contrast, allows to add those functions in the abstraction layer that may not be available in certain management systems.

Note that no cluster management system or scheduler listed in Table 1 supports the modification of node states like *up* and *down*, which is necessary to support the planning and execution of maintenance actions.

Maintenance Actions. As outlined in Section 1 automating the administration of large clusters is an important issue. Administrative tasks that may affect jobs running on the same node, must be taken into account by the scheduler by planning the maintenance task just like an ordinary job.

4 <http://www.sun.com/software/gridware/>

mgmnt system	fifo	BF	AR	SMA
CCS 4.0	yes	yes	yes	no
LL 3.1	yes	yes	no	no
LSF 5.1	yes	yes	yes	no
OpenPBS 2.3	yes	no	no	no
PBSPRO 5.2	yes	yes	yes	no
SGE 5.3	yes	no	no	no
scheduler				
Maui 3.2.6	yes	yes	yes	no

Table 1. Scheduling features of some cluster management systems and schedulers (BF - backfill, AR - advance reservation, SMA - support for maintenance actions).

A simple but effective method is to disable all affected nodes during the task. To schedule a maintenance action, the *admin* component contacts the resource management for a specific or flexible time slot on a set of nodes. The scheduler decides and schedules the node state change. If the request was successful, the administrative task can be performed during the agreed time slot.

Advance Reservations. Our scheme of handling maintenance actions needs the capability to request time slots in the future for the coordinated planning of system maintenance. As shown in Table 1 not all cluster schedulers support advance reservations. Hence we replace them by a more powerful one. This non-intrusive approach to replace schedulers in cluster management systems was described in Section 3.3.

Filtering and Annotating Control Information. In the abstraction layer, also format conversions are done to ensure the proper functioning of the system. This is done using the filtering capability of the abstraction layer which also provides a flexible and generic method to annotate status information of queues, nodes and jobs. For example, node status information may be annotated with data obtained from the monitoring component or the configuration database (see Fig. 1). Similarly, queue and job status information may be manipulated to let the scheduler assume a single (partitioned) cluster that is composed out of many separate elements in reality.

4. Implementation

We implemented the described architectural framework in the DataGrid project. The abstraction layer contains four threads (Fig. 5):

- *server interface*: comprises the status retrieval operations (*getNode*s, *getJob*s, *getQueue*s), the job control operations (*startJob*, *cancelJob*) and the maintenance actions support operation *setNodeState*.
- *scheduler interface*: serves as a proxy to a scheduler. It accepts the status requests (*getNode*s, *getJob*s, *getQueue*s), the job control requests (*startJob*, *cancelJob*) and the reservation requests (*createReservation*, *cancelReservation*).
- *maintenance interface*: switches nodes virtually off (*node-off*) or cancels such requests (*cancel*). These requests are handled using operations of the server interface (*setNodeState*) and the scheduler interface (*createReservation*, *cancelReservation*).
- *runtime control*: controls the operations and the configuration of the abstraction layer itself during runtime. The whole system can be stopped or restarted and configuration information can be read.

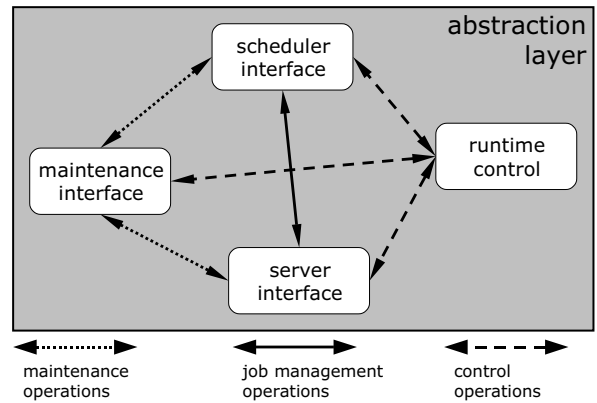


Figure 5. Components of the abstraction layer.

The scheduler interface is currently used by the Maui scheduler, but any other scheduler could be used as well. We provide different adaptors like an OpenPBS API and a script adaptor for the server interface. While the first uses API functions of OpenPBS, the latter executes external shell scripts for performing the actions. We also support LSF and Condor via script adaptors. The scheduler interface calls methods of the adaptors via a common interface and receives results accordingly. Hence, the scheduler is not aware of the cluster management system it steers.

The script adaptor allows rapid adaptation to new cluster management systems. For debugging purposes, for example, we connected a simulator with a script adaptor in just a few hours.

5. Performance Evaluation

The adaptation layer causes a marginally higher CPU overhead compared to environments where the scheduler and cluster management systems interact directly. To determine this overhead, we measured the performance of `getNode`s, `getJobs` and `startJob`. We examined the overhead for the following system setups:

- *Maui* ↔ *OpenPBS (M-PBS)*: Maui uses OpenPBS directly. This will give the best performance.
- *Maui* ↔ *AL* ↔ *OpenPBS scripts (M-AL-script)*: Maui uses the abstraction layer which uses the script adaptor for OpenPBS. This combination is expected to cause the highest overhead since it involves several system calls to `fork` and `exec`.
- *Maui* ↔ *AL* ↔ *OpenPBS API (M-AL-API)*: Maui uses the abstraction layer which uses an adaptor linked to the OpenPBS library. The performance of this combination should be somewhere between the two others.

The benchmarks were executed on a Linux cluster with 16 dual Intel Xeon nodes with 2.2 GHz, 512 KB second level cache and 1 GB of memory. The cluster uses a GigaBit switch and FastEthernet for each node. For all scenarios, we submitted 100 jobs to the cluster management system and recorded the timings of the requests `getNode`s, `getJobs` and `startJob` in the scheduler as well as in the AL. The results are shown in Table 2.

Scenario	getNode		getJobs		startJob	
	M	AL	M	AL	M	AL
<i>M-PBS</i>	121	-	180	-	71	-
<i>M-AL-script</i>	46	44	318	309	110	109
<i>M-AL-API</i>	13	11	236	227	67	66

Table 2. Performance for certain scheduler requests (times in ms) – M - Maui, AL - abstraction layer.

Interestingly the `getNode`s requests are faster with our system than with Maui and OpenPBS. This is due to the fact that the Maui scheduler fetches node attributes from each node rather than relying on information that is cached on the server. The scheduler in our system, in contrast, does not request the information from each node separately. The values for the other requests follow the expected behavior.

6. Conclusion

We presented an architectural framework for the coordinated management of multiple heterogeneous clusters in a fabric. Our scheme, which is included in the EU Data-Grid software package, virtualizes cluster management software and allows them to interact with each other via adaptors. Using adaptors, different kinds of jobs (grid jobs, local jobs, maintenance jobs) can be served by the same methods, and existing cluster management software can be extended by adding further functions.

Acknowledgements

We thank our student staff members K. Pauls, J. Bardins and J. Meltzer who helped with the implementation.

References

- [1] S. Bethke, M. Calvetti, H. Hoffmann, D. Jacobs, M. Kasemann, and D. Linglin. Report of the steering group of the LHC computing review. Technical Report, CERN European Organization for Nuclear Research, February 2001.
- [2] B. Bode, D. Halstead, R. Kendall, and Z. Lei. The Portable Batch Scheduler and the Maui scheduler on Linux Clusters. In *Usenix Conference*, Atlanta, GA, October 2000.
- [3] R. Henderson. Job scheduling under the Portable Batch System. In *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 279–294. Springer-Verlag, 1995.
- [4] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. Skovira. *Workload Management with LoadLeveler*. IBM Redbooks, November 2001.
- [5] A. Keller and A. Reinefeld. Anatomy of a resource management system for HPC clusters. In *Annual Review of Scalable Computing*, volume 3. 2001.
- [6] P. Papadopoulos, M. Katz, and G. Bruno. NPACI Rocks: Tools and techniques for easily deploying manageable Linux clusters. *Concurrency and Computation: Practice and Experience*, 15(7-8):707–725, June-July 2003.
- [7] A. Reinefeld and V. Lindenstruth. How to build a high-performance compute cluster for the Grid. In *2nd International Workshop on Metacomputing Systems and Applications (MSA2001)*, Valencia, Spain, September 2001.
- [8] P. Uthayopas, J. Manesilp, and P. Ingongnam. SCMS: An integrated cluster management tool for Beowulf cluster system. In *Proceedings of the International Conference on Parallel and Distributed Proceeding Techniques and Applications 2000 (PDPTA'2000)*, Las Vegas, Nevada, USA, June 2000.
- [9] S. Zhou, X. Zheng, J. Wang, and P. Delisle. Utopia: A load sharing facility for large, heterogenous distributed computer systems. *Software – Practice & Experience*, 23(12):1305–1336, December 1993.