Int. J Computational Science and Engineering (*Accepted*)

A User-Centric Cluster and Grid Computing Portal

Erik Elmroth*
Department of Computing Science and HPC2N,
Umeå University, Sweden
E-mail: elmroth@cs.umu.se
*Corresponding author

Mats Nylén
Department of Physics and HPC2N,
Umeå University, Sweden
E-mail: nylen@tp.umu.se

Roger Oscarsson
HPC2N,
Umeå University, Sweden
E-mail: roger@hpc2n.umu.se

Abstract: The HPC2N Grid portal is a user-centric environment that provides a homogeneous interface to a set of heterogeneous high-performance computing resources from standard web-browsers. The interface includes support for most everyday activities for a regular user, such as to submit, manipulate and delete jobs, monitor queues and job status, obtain user-, project-, and resource statistics and information, view job output, etc. This contribution reviews the portal functionalities and presents the design and implementation of the underlying system architecture. Some major design considerations, features and limitations are discussed and future extensions are outlined. The portal currently gives access to all major resources at HPC2N, in total comprising over 700 CPUs.

1. INTRODUCTION

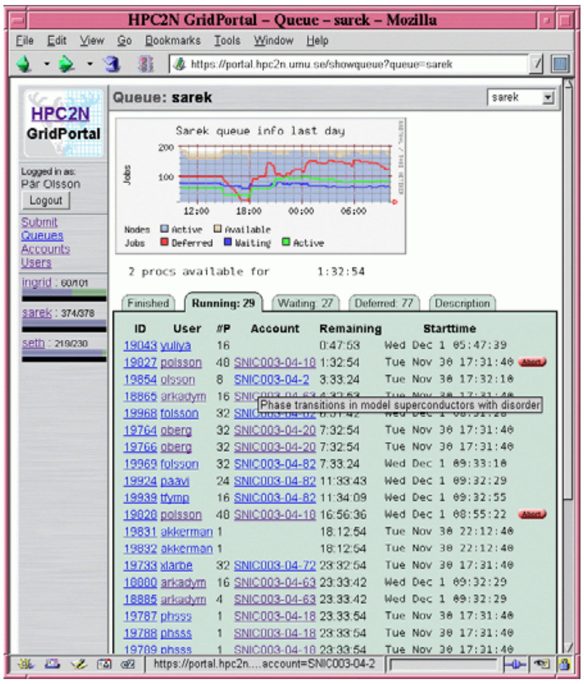
Various types of web-based portals are expected to become frequently used alternatives or complements to traditional interfaces to individual high-performance computing (HPC) systems or general Grid resources. Today, there are several projects developing different types of portals, or tools for constructing portals, ranging from general interfaces for sets of resources to more application-oriented or application-specific portals, often referred to as science portals. For examples, see [1, 3, 6, 9, 11-15, 17, 18, 20-22, 26, 28-31].

This contribution presents a general user-centric portal for accessing a set of HPC systems, possible to extend to resources accessed via Grid interfaces. The objective is to develop a portal that gives a homogeneous interface to a set of heterogeneous resources for most everyday system interaction of a regular HPC user. This includes support for submitting and manipulating jobs, monitoring queues and job status, obtain user-, project-, and resource statistics and information, to view job output, etc.

In the following we present the functionality, design, system architecture, and implementation (in Perl) of this portal, currently in use at the High Performance Computing Center North (HPC2N), a national HPC center in Sweden. The HPC2N resources accessible via the portal include different types of Linux clusters with 100 to 384 processors and an IBM SP system [4].

2. FUNCTIONALITY

After a standard login procedure with username and password, the user faces the portal front page, e.g., as shown in Figure 1.


HPC2N GridPortal – Queue – sarek – Mozilla
File Edit View Go Bookmarks Tools Window Help
https://portal.hpc2n.umu.se/showqueue?queue=sarek
HPC2N GridPortal
Logged in as:
Pär Olsson
Logout
Submit
Queues
Accounts
Users
ingrid : 60/101
sarek : 374/378
seth : 219/230
Queue: sarek
sarek
Sarek queue info last day
Jobs
200
100
12:00
18:00
00:00
06:00
Nodes ▨ Active ▨ Available
Jobs 🟥 Deferred 🟦 Waiting 🟩 Active
Finished
Running: 29
Waiting: 27
Deferred: 77
Description
Abort
Phase transitions in model superconductors with disorder
https://portal.hpc2n....account=SNIC003-04-2

2 procs available for 1:32:54

ID	User	#P	Account	Remaining	Starttime
19043	yuliya	16		0:47:53	Wed Dec 1 05:47:39
19827	polsson	40	SNIC003-04-10	1:32:54	Tue Nov 30 17:31:40
19854	olsson	8	SNIC003-04-2	3:33:24	Tue Nov 30 17:32:10
18865	arkadym	16	SNIC[illegible]	[illegible]	[illegible]
19968	folsson	32	SNIC[illegible]	[illegible]	[illegible]
19764	oberg	32	SNIC003-04-20	7:32:54	Tue Nov 30 17:31:40
19766	oberg	32	SNIC003-04-20	7:32:54	Tue Nov 30 17:31:40
19969	folsson	32	SNIC003-04-82	7:33:24	Wed Dec 1 09:33:10
19924	paavi	24	SNIC003-04-82	11:33:43	Wed Dec 1 09:32:29
19939	tfymp	16	SNIC003-04-82	11:34:09	Wed Dec 1 09:32:55
19828	polsson	40	SNIC003-04-10	16:56:36	Wed Dec 1 08:55:22
19831	akkerman	1		18:12:54	Tue Nov 30 22:12:40
19832	akkerman	1		18:12:54	Tue Nov 30 22:12:40
19733	xlarbe	32	SNIC003-04-72	23:32:54	Tue Nov 30 17:31:40
18880	arkadym	16	SNIC003-04-63	23:33:42	Wed Dec 1 09:32:29
18885	arkadym	4	SNIC003-04-63	23:33:42	Wed Dec 1 09:32:29
19787	phsss	1	SNIC003-04-18	23:33:54	Tue Nov 30 17:31:40
19788	phsss	1	SNIC003-04-18	23:33:54	Tue Nov 30 17:31:40
19709	phsss	1	SNIC003-04-10	23:33:54	Tue Nov 30 17:31:40

Figure 1. *Running jobs queue.*

In this window information is presented for the user's default resource, with the queue information for running jobs taking up most of the space, but there is also a graph showing a summary of job queue statistics and some links and menus for selecting other type of information or information about other resources.

The information presented for each running job is the job-ID, the user name, the number of processors used, the project account to be charged for the job, the start time, and the time remaining. The job-ID, the user name, and the account number can be expanded for more information. For example, it is possible to get a processor utilization graph for each running job. For jobs owned by the user being logged in, there is also an "abort" button for each job, that can be used to kill the running job.

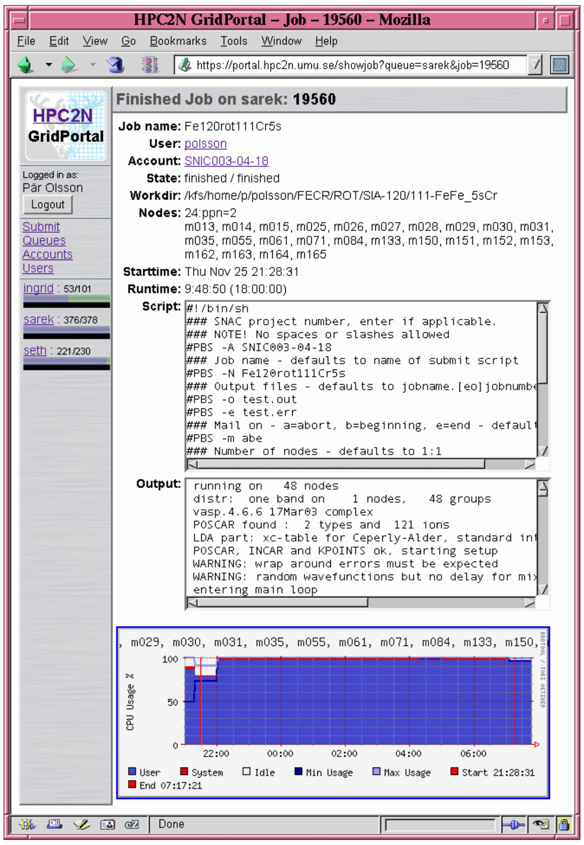
Above the queue information, there are four additional tabs of which three are for obtaining the corresponding lists for finished job and for jobs in the active and in-active part of the queue. The last tab is for obtaining a description of the resource of current interest. The two listings of queued jobs are rather similar to the list of running jobs, with the start time and remaining time replaced by the time requirement as specified at job submission and the time since the job was submitted. By expanding a queued job, job specific information is presented together with an explanation of why the job has not yet been started.

The list of finished jobs only includes the jobs owned by the user currently logged in on the portal. The list shows all jobs completed during the last few days, with each job's job-ID, finishing status, number of processors used, project account charged for the execution, the total time required for the job, and the date and time when it was completed. Detailed information about each single job can be obtained by expanding the job-ID, as shown in Figure 2. This gives information about which CPU's that have been used, the total CPU utilization, the complete script used for job submission, and the standard output and standard error (if non-empty). The graph showing the total CPU utilization can be expanded in a new window showing the CPU utilization for each individual CPU used.

From the submit link on the front page, the user is presented a job submission interface, as in Figure 3. This interface provides menus for specifying a resource (i.e., queue) which to submit a job to, which project account to charge, and if the resource capacity required is specified in terms of CPUs or nodes. The job submission script can be entered in a text field or uploaded from a file. Text fields are also available for assigning a job name, and for entering the number of CPUs or nodes, and the maximum time requirement for the job.

The account information can be accessed either per project only or organized per user. For example, Figure 4 shows the two projects in which a specific user participates. By expanding each tab, the more specific information, such as the project abstract and a list of project members is presented for each project. Similar information is also available on a per project basis.

System information for all resources is integrated in the portal. The overview of one resource is presented in Figure 5. More detailed information can also be presented.



https://portal.hpc2n.umu.se/showjob?queue=sarek&job=19560

Logged in as: Pär Olsson

ingrid : 53/101
sarek : 376/378
seth : 221/230

Finished Job on sarek: 19560

Job name: Fe120rot111Cr5s
User: polsson
Account: SNIC003-04-18
State: finished / finished
Workdir: /kfs/home/p/polsson/FECR/ROT/SIA-120/111-FeFe_5sCr
Nodes: 24:ppn=2
m013, m014, m015, m025, m026, m027, m028, m029, m030, m031, m035, m055, m061, m071, m084, m133, m150, m151, m152, m153, m162, m163, m164, m165
Starttime: Thu Nov 25 21:28:31
Runtime: 9:48:50 (18:00:00)
Script:

```
#!/bin/sh
### SNAC project number, enter if applicable.
### NOTE! No spaces or slashes allowed
#PBS -A SNIC003-04-18
### Job name - defaults to name of submit script
#PBS -N Fe120rot111Cr5s
### Output files - defaults to jobname.[eo]jobnumbe
#PBS -o test.out
#PBS -e test.err
### Mail on - a=abort, b=beginning, e=end - defaul
#PBS -m abe
### Number of nodes - defaults to 1:1
```

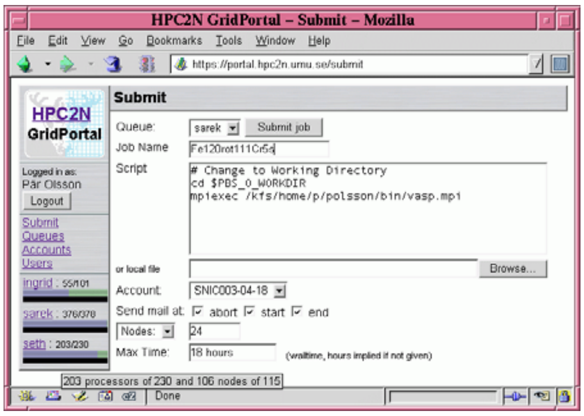
Output:

```
 running on   48 nodes
 distr:  one band on    1 nodes,   48 groups
 vasp.4.6.6 17Mar03 complex
 POSCAR found :  2 types and  121 ions
 LDA part: xc-table for Ceperly-Alder, standard in
 POSCAR, INCAR and KPOINTS ok, starting setup
 WARNING: wrap around errors must be expected
 WARNING: random wavefunctions but no delay for mi
 entering main loop
```

, m029, m030, m031, m035, m055, m061, m071, m084, m133, m150,

🟦 User 🟥 System ☐ Idle ■ Min Usage ▨ Max Usage 🟥 Start 21:28:31
🟥 End 07:17:21

Figure 2. *Details of a finished job.*



https://portal.hpc2n.umu.se/submit

Logged in as: Pär Olsson

ingrid : 55/101
sarek : 376/378
seth : 203/230

Submit

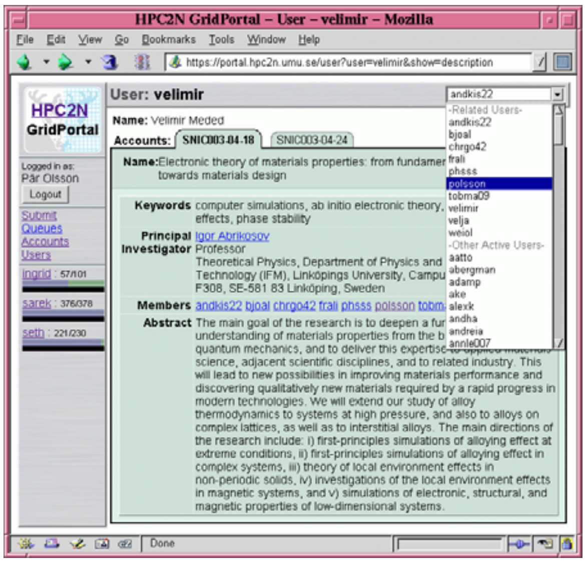
Queue: sarek
Job Name: Fe120rot111Cr5s
Script:

```
# Change to Working Directory
cd $PBS_O_WORKDIR
mpiexec /kfs/home/p/polsson/bin/vasp.mpi
```

or local file:
Account: SNIC003-04-18
Send mail at: ✓ abort ✓ start ✓ end
Nodes: 24
Max Time: 18 hours (walltime, hours implied if not given)

203 processors of 230 and 106 nodes of 115

Figure 3. *Job submission.*



https://portal.hpc2n.umu.se/user?user=velimir&show=description

Logged in as: Pär Olsson

ingrid : 57/101
sarek : 376/378
seth : 221/230

User: velimir

andkis22
-Related Users-
andkis22
bjoal
chrgo42
frali
phsss
polsson
tobma09
velimir
velja
weiol
-Other Active Users-
aatto
abergman
adamp
ake
alexk
andha
andreia
annie007

Name: Velimir Meded

Accounts: SNIC003-04-18 | SNIC003-04-24

Name: Electronic theory of materials properties: from fundamer
towards materials design

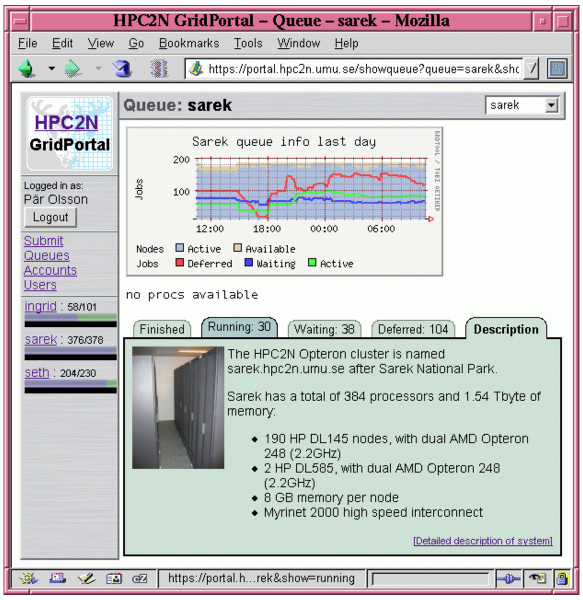
Keywords computer simulations, ab initio electronic theory,
effects, phase stability

Principal Investigator Igor Abrikosov
Professor
Theoretical Physics, Department of Physics and
Technology (IFM), Linköpings University, Campu
F308, SE-581 83 Linköping, Sweden

Members andkis22 bjoal chrgo42 frali phsss polsson tobm

Abstract The main goal of the research is to deepen a fur
understanding of materials properties from the b
quantum mechanics, and to deliver this expertise
science, adjacent scientific disciplines, and to related industry. This will lead to new possibilities in improving materials performance and discovering qualitatively new materials required by a rapid progress in modern technologies. We will extend our study of alloy thermodynamics to systems at high pressure, and also to alloys on complex lattices, as well as to interstitial alloys. The main directions of the research include: i) first-principles simulations of alloying effect at extreme conditions, ii) first-principles simulations of alloying effect in complex systems, iii) theory of local environment effects in non-periodic solids, iv) investigations of the local environment effects in magnetic systems, and v) simulations of electronic, structural, and magnetic properties of low-dimensional systems.

Figure 4. *A user’s project information.*



https://portal.hpc2n.umu.se/showqueue?queue=sarek&shc

Logged in as: Pär Olsson

ingrid : 58/101
sarek : 376/378
seth : 204/230

Queue: sarek

sarek

Nodes ▨ Active ▨ Available
Jobs 🟥 Deferred 🟦 Waiting 🟩 Active

no procs available

Finished | Running: 30 | Waiting: 38 | Deferred: 104 | **Description**

The HPC2N Opteron cluster is named sarek.hpc2n.umu.se after Sarek National Park.

Sarek has a total of 384 processors and 1.54 Tbyte of memory:

- 190 HP DL145 nodes, with dual AMD Opteron 248 (2.2GHz)
- 2 HP DL585, with dual AMD Opteron 248 (2.2GHz)
- 8 GB memory per node
- Myrinet 2000 high speed interconnect

[Detailed description of system]

https://portal.h...rek&show=running

Figure 5. *Sample resource description.*

3. INTERFACE DESIGN

A guiding principle in the interface design is that the portal should be as close to browser independent as possible. In order to ensure this, test and evaluation have been made using all of Internet Explorer, Mozilla, Safari, and w3m. The use of Internet Explorer version 5 or more recent ensures compatibility with approximately two thirds of all users [2]. The other three choices reflect the majorities of other users, Macintosh users, and users preferring text-based browsers.

3.1. General design considerations

The strive for generality and portability lead to some general design decisions including to completely avoid pixel-based positioning, mainly as this often leads to a need for pixel-based positioning and sizing to large extent. Cascading Style Sheets (CSS) [32] are useful, but since CSS often requires pixel based positioning and sizing, and not all browsers implements CSS fully and correctly, we have chosen to use CSS to some extent, but not everywhere it possibly could have been used.

For the actual web page layout, we have decided to use traditional HTML tables, as an alternative to frames. The decision not to use frames is based on the facts that they are difficult to bookmark, only can be sized in pixels or percentages, and they do either lead to the use of JavaScript to navigate or a proliferation of pages. JavaScript is only used for enhancement and not for necessary functionality in order to avoid some portability problems and to enable the use of text-based browsers. Graphics are also used restrictively for the same reason.

3.2. Page layout

The page layout must be designed with at least one fixed area, e.g., for navigation, and one dynamic area for various types of display and interaction. Based on studies of other portal initiatives in the HPC area and elsewhere, our decision is to have one narrow fixed area on the left hand side and a large dynamic area on the right, as this typically leads to a need for vertical scrolling instead of horizontal.

The fixed area is partitioned into four boxes for a logo, for login/logout, for navigational links, and for resource status information. The dynamic area is used for all other types of information, as illustrated in Figures 1 and 3 – 5.

We have imposed strong requirements on the size of the portal in order to optimize its functionality also for low-resolution displays. Hence, we have strived to use the available screen area efficiently, so that the user should not need to use both horizontal and vertical scrollbars.

4. BACK-END DESIGN

The back-end part of the portal server collects and organizes the information that is to be presented to the user. Figure 6 illustrates the overall function of the server. Each request from a client browser starts up the server CGI script. The front-end of the server interprets the requests and the back-end retrieves the requested information from various external sources. Finally, the information is formatted and sent back to the client browser.

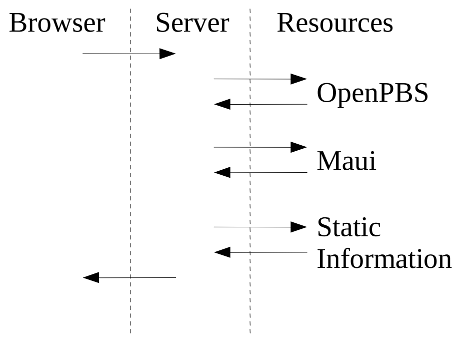


Figure 6. *Overall function of the portal.*

Data about batch queues, job status, etc, is retrieved using various batch system commands. These commands are issued to OpenPBS [23] or to the Maui [27] scheduler. The output from these commands contains the information about batch queues and jobs that the server needs in order satisfy the client requests.

Since the batch systems in use at HPC2N has no built-in support for retrieving information about finished jobs, i.e., jobs that have completed, the available information needs to be amended. This is accomplished by running a special script at the end of each batch job that stores the information about that job. The data saved includes some statistics about the job, the batch script, standard input, output and error.

Apart from the files maintained for the security solution described in Section 5 the portal server can be considered stateless, i.e., it maintains no session specific state over time or from one access to the next.

4.1. Caching

The portal server makes extensive use of both internal and external caching for improved performance and reduced interference with surrounding systems.

The methods that retrieve information from the batch system are typically invoked several times during each request to the portal server. For example, when creating the view shown in Figure 1, the back-end needs to retrieve some information about waiting and deferred jobs in addition to the running queue. In order to avoid unnecessary parsing and interaction with the batch system, the output from each command is *cached internally* after parsing.

Some of the commands to the Maui scheduler take substantial time to complete, e.g., the `showq` command for showing the complete batch queue. In order to improve responsiveness of the portal, output from such commands is *cached in an external file cache* whenever this is possible and meaningful. The files in the external file cache are updated regularly external to the server, using the UNIX `cron` facility. This allows for the portal server to access the file cache already available instead of querying the batch system for such commands.

4.2. Batch systems and queues

The methods for accessing batch queues and jobs are implemented in an object oriented fashion where the top-level objects are `Queue` and `Job`. These objects are then specialized to the batch system in use at HPC2N (OpenPBS with Maui as scheduler), with the additional support for the handling of data for finished jobs. Figure 7 shows the inheritance diagrams for these classes with specialization to the combination OpenPBS with the Maui scheduler. The extended functionality required for aggregating data for finished jobs is handled with additional PBS classes (the classes with names ending in "_f").

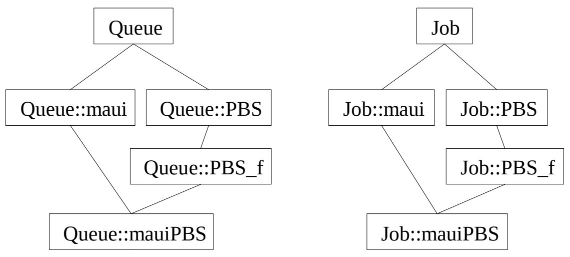


Figure 7. *Queue and Job object classes.*

Whenever the portal needs to interact with the batch system a `Queue` object is instantiated. The main methods in `Queue` are: `jobs`, `submit` and `delete`. The methods `submit` and `delete` are described in Section 4.3. The `jobs` method returns a list of batch jobs fulfilling some specified criteria, e.g., all finished jobs for a specific user. The initial call to the `Queue::job` method retrieves the information required from the batch system, from command output and from the file cache. This textual information is then parsed and stored in the internal cache. Subsequent calls during the processing of the request that requires pieces of this data can then utilize this internally cached information.

Each batch job in the list returned by the `Queue::jobs` method is an object of the `Job` type. The main method in the `Job` class is `get` that returns information about the job. The `Job::get` method takes as an argument the required information, examples include `user`, `starttime`, `account`, etc.

As an example, to generate the data needed for the display of the batch queue shown in Figure 1, showing all running jobs, as well as the number of waiting, deferred and finished jobs, the following actions are performed:

Instantiate an object `Queue::mauiPBS`. This creates a `Queue` object specialized to the situation at HPC2N, including the interface for the finished jobs.

Call the method `Queue::jobs`, requesting a list of all jobs in the state running. This call will obtain most of the required information by reading and parsing the `showq` output that is stored in the external file cache. After the parsing, this data is stored in the internal cache, and the list of running jobs is returned.

Loop over all jobs in the list, retrieving the data using the `Job::get` method, each call returning one field of data, e.g., the account number, or the start time for the job. During these calls all the data is already in the internal cache.

Retrieve the number of jobs in the waiting, deferred and finished state, by calling the `Queue::jobs` method once for each of the states and counting the length of the list. At this stage there is no cached information about the finished jobs. In order to retrieve this information the back-end needs to look at the files generated as described above.

If the user has requested information about an individual job, the `Queue::jobs` can be called with a single job-ID as argument, and then the `Job::get` method is called for detailed information about that job. For jobs in the batch system, i.e., jobs with status running, waiting or deferred, this includes obtaining output from various OpenPBS commands, whereas for finished jobs the data comes from the files stored at the end of each job.

The type of information available for a job depends on the state of the job. For waiting and deferred jobs there is not much detailed information available. For running jobs the start time and node allocation is known, which can be combined with the statistics described below to generate graphs. For finished jobs, the files stored at the end of the batch job include standard input, output and error, as well as the batch script. Information such as start time, end time and node allocation is also stored. All of this information is available to the portal back-end and can be presented to the user.

4.3 Job submission and deletion

When a user requests the deletion of a batch job, the back-end calls the `delete` method in the `Queue` class. This method then calls the appropriate batch queue command, i.e., `qdel` in the case of OpenPBS, and the job gets deleted.

When a user submits a batch job through the portal, the portal server collects the information that the user has specified in the fields on the web interface shown in Figure 3. The server then invokes the `Queue::submit` method which assembles the various pieces of information, job name, account number, maximum execution time, etc, into a batch script suitable for submission. Finally, the back-end submits this script, using `qsub` in the case of OpenPBS.

4.4. Statistics and graphs

Statistics are made available to the portal server from a few sources. First of all, detailed statistics about individual nodes are accumulated during operation of the HPC2N clusters using Ganglia [16]. Secondly, statistics about finished batch jobs are kept track of using the mechanism described above. Thirdly, Ganglia has been customized to log certain cluster-wide data, e.g., the number of available nodes.

To assemble the statistics needed to put together graph like the one shown in Figure 2, for a finished batch job, the different statistics has to be combined. From the information stored at the end of the batch job, the start time and number and identity of the allocated nodes can be found. We can then search the data logged by Ganglia to determine the CPU usage as function of time for these nodes.

The transformation of the assembled statistics into the graphs that are presented to the user is accomplished with RRDtool [24]. Examples of graphs generated are given in Figures 1, 2, and 5.

4.5. Static information

In addition to the information that is gathered from dynamical data-sources like the batch system or the statistics, the portal also makes use of data that is more static in nature. Examples of this are user and project information, static information about the available resources, etc. Much of this data is updated very infrequently.

The user and project information is maintained externally to the server in flat-file databases and XML-files containing detailed information about projects. These sources are accessed by the server and parsed whenever it is requested.

Static information about the resources is kept in separate files that contain HTML-text that can be included directly into the web-pages presented to the user.

5. SECURITY ISSUES

All communication between the portal server and browser is performed using the https protocol. Figure 8 illustrates the basic procedure for validating a user. First the username and the password are sent from the browser to the server, and then the server verifies these using Kerberos authentication, i.e., the portal server issues a `kinit` command to the HPC2N Kerberos server. If the user/password combination is valid the server sets a `session-ID` cookie in the browser consisting of a random string of 12 hexadecimal digits followed by the username and the full name of the user. The fields in the cookie are separated by a colon ('`:`'). The Kerberos ticket returned by the Kerberos server is not used and is discarded.

In order to keep track of the active sessions, the server also maintains a directory with one file for each active session. Each file name consists of the username and the same random string as the cookie of the corresponding active session. The file is created during the login procedure as indicated in Figure 8. Every time a user initiates an action, the server checks that the browser's cookie corresponds to an active session. If the cookie is invalid or non-existent the server returns the login page to the browser.

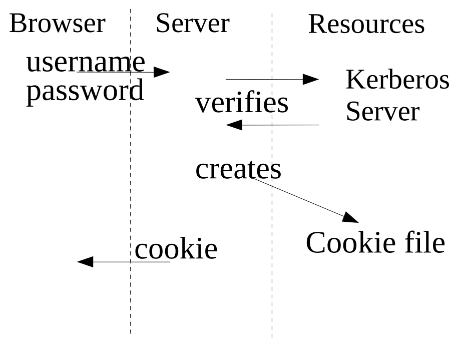


Figure 8. *Steps to initiate a session.*

In order to deactivate sessions that remain inactive for too long, the server touches the file on each action. A `cron` job then deletes any files that have not been touched for 40 minutes, thereby deactivating the session.

In principle, all actions performed by the portal server on behalf of the user should be executed on his/her user account. This requires the use of a setuid program to run commands. The setuid program is kept quite simple and uses the same security mechanism as the server, i.e., it checks that the user and cookie corresponds to a cookie file for an active session. For many actions, however, the result is independent of the user that executes them. In these cases the portal server executes the actions under its own user account, avoiding the use of the setuid program and thereby saving a significant amount of time.

6. DESIGN CONSIDERATIONS AND FUTURE WORK

We are currently performing a number of extensions and modification to the HPC2N portal. One significant task is to make it possible to access not only individual resources but also sets of resources via Grid interfaces, as provided, e.g., by the Globus Toolkit [10] or NorduGrid ARC [19]. This includes modifications for enabling interaction with the Grid middleware interfaces and not only resource specific interfaces, but it also have implications that will improve the current functionality. New functionality can take advantage of Grid-specific features not provided by the individual resources, such as Grid resource brokers [7, 8] and Grid-wide accounting information [5, 25].

A major current task is a redesign of the security solution to be based on the Grid Security Infrastructure (GSI). Even though this will impose the requirement on the resources to allow GSI-based access using X.509 certificates it has a number of advantages. For example, the resource will not have to trust the portal, it will be sufficient to trust the user's (delegated) credentials passed via the portal. It will also enable access to the resources via GSI-SSH and thereby be more general than the current solution when it comes to batch queue access. The current solution requires that the batch queue can be accessed via network interfaces, which is not the case for, e.g., LoadLeveler. Moreover, GSI-SSH based access can be the basis for a general approach to all kind of batch queue access, including job submission, deletion, modification, and queue listing. Another advantage is that it provides a convenient solution to the problem of accessing file systems that are not mounted on the portal server, typically local file systems on the remote clusters.

We plan to make the infrastructure more general with respect to different batch systems by reorganizing the object-oriented class hierarchy for `Queue` and `Job.` This includes making a more generic framework for accessing data for finished jobs. As most schedulers do not keep information about these jobs, and our current solution basically is designed for OpenPBS with Maui, a more standardized solution will benefit the access to other resources including resources accessed via Grid interfaces.

We are also addressing the intricate problem of providing relevant time information to the user in cases where the user, the portal, and the resources are in different time zones (and the user possibly moving around or being unaware of the physical location of the portal server). Ideally, the portal should always provide time information relative to the user's current time zone, but as the browsers do not provide this information as part of their requests, this problem has to be solved, or possibly circumvented by only presenting relative time information.

Finally, the extension of the typical usage scenario to include resources from more than one site has lead to the start-up of a new initiative for a distributed database solution for maintaining and appropriately combining all user and project information.

7. CONCLUDING REMARKS

We have presented a user-centric portal for general access to a set of high-performance computing resources. The portal provides a homogeneous interface to a set of heterogeneous resources, which by the outlined extensions can include Grid resources. The current prototype is used for accessing the resources of HPC2N, including over 700 CPUs in total. Our aim has been to provide a general interface for the every-day usage of a wide group of users, i.e., we have not tried to provide a more application-oriented portal for a specific application area or a portal with support for setting up specific types of computations such trivially parallel parameter sweeps or data analysis applications. Our view is that such environments can be provided as future extensions of this portal or as stand-alone components built on the infrastructure developed here.

In addition to providing a portal for our needs in the near future, our aim has been to gain the experience required for developing a more portable portal for a wider spectrum of infrastructures. This work has, as many of the related projects, been highly dependent on the actual configuration of the underlying infrastructure. By extending our work as proposed in Section 6, to include also general Grid infrastructure, our long term plan is both to support a large variety of infrastructures and to provide front-end functionality that can be combined with a flexible back-end for easy incorporation of resources with new configurations. In this presentation we propose such a front-end design and a proof-of-concept back-end implementation.

The design of the overall architecture is made with focus on the user-perspective, and by taking into account a whole range of issues, including support for a variety of browsers, heterogeneity of resources and their interfaces, interface and web page layouts, overcoming performance bottlenecks in batch systems, etc. This contribution presents these solutions and outlines future directions of this research.

8. ACKNOWLEDGMENTS

We are grateful to many colleagues for their support and contributions that have helped improving this work. In particular we wish to thank Bo Kågström, Markus Mårtensson, Peter Poromaa and Åke Sandgren.

This work has in part been funded by The Swedish Research Council (VR) under contracts 343-2003-953 and 2004-5787.

9. REFERENCES

[1] G. Aloisio and M. Cafaro. Web-based access to the Grid using the Grid Resource Broker portal. *Concurrency Computat.: Pract. Exper.*, 14, pp. 1145-1160, 2002.

[2] Browser News.
http://www.upsdell.com/BrowserNews/.

[3] M. Dahan, M. Thomas, E. Roberts, A. Seth, T. Urban, D. Walling, J.R. Boisseau: Grid Portal Toolkit 3.0 (GridPort). HPDC 2004: 272-273.

[4] N. Edmundsson, E. Elmroth, B. Kågström, M. Mårtensson, M. Nylén, Å. Sandgren, and M. Wadenstein. Design and Evaluation of a TOP100 Linux Super Cluster System. *Concurrency Computat.: Pract. Exper*., 16, pp. 735-750, 2004.

[5] E. Elmroth, P. Gardfjäll, O. Mulmo, and T. Sandholm. An OGSA-based Bank Service for Grid Accounting Systems. In *State-of-the-art in Scientific Computing. Springer-Verlag, LNCS, (accepted).*

[6] E. Elmroth, P. Johansson, B. Kågström, and D. Kreßner. A Web Computing Environment for the

SLICOT Library. In *Proc. The Third NICONET Workshop*, pp. 53-61, 2001.

[7] E. Elmroth and J. Tordsson. A Grid Resource Broker Supporting Advance Reservations and Benchmark-based Resource Selection. In *State-of-the-art in Scientific Computing. Springer-Verlag, LNSC, (accepted).*

[8] E. Elmroth and J. Tordsson. An Interoperable Standards-based Grid Resource Broker and Job Submission Service, *e-Science 2005. First IEEE Conference on e-Science and Grid Computing*, IEEE Computer Society Press, USA, (accepted)

[9] D. Gannon, G. Fox, M. Pierce, B. Plale, G. von Laszewski, C. Severance, J. Hardin, J. Alameda, M. Thomas, J. Boisseau. Grid Portals: A Scientist's Access Point for Grid Services (DRAFT 1). Sept. 19 2003, GGF working draft.

[10] Globus. http://www.globus.org.

[11] GridPort. http://www.gridport.net.

[12] GridSphere. http://www.gridsphere.org.

[13] T. Haupt, P. Bangalore, and G. Henley. Mississippi Computational Web Portal. *Concurrency Computat.: Pract. Exper,*. 14, pp. 1275-1287, 2002.

[14] S. Krishnan, R. Bramley, D. Gannon, M. Govindaraju, R. Indurkar, A. Slominski, and B. Temko. The XCAT Science Portal. *Supercomputing 2001*, (2001).

[15] G.J. Lewis, G. Sipos, F. Urmetzer, V. N. Alexandrov, P. Kacsuk: The Collaborative P-GRADE Grid Portal. International Conference on Computational Science (3) 2005: 367-374.

[16] M. Massie, B.N. Chun, and D.E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, Vol. 30, Issue 7, July 2004.

[17] A. Natrajan, A. Nguyen-Tuong, M.A. Humphrey, M. Herrick, B.P. Clarke, and A.S. Grimshaw. The Legion Grid Portal. *Concurrency Computat.: Pract. Exper.*, 14, pp. 1365-1394, 2002.

[18] C. Németh, G. Dózsa, R. Lovas, P. Kacsuk: The P-GRADE Grid Portal. ICCSA (2) 2004: 10-19.

[19] Nordugrid. Advanced Resource Connector (ARC). http://www.nordugrid.org/middleware/.

[20] J. Novotny. The Grid Portal Development Kit. *Concurrency Computat.: Pract. Exper.*, 14, pp. 1129-1144, 2002.

[21] J. Novotny, M. Russell, and O. Wehrens. GridSphere: a portal framework for building collaborations. *Concurrency Computat.: Pract. Exper.*, 16, pp. 503-513, 2004

[22] M.E. Pierce, C. Youn, and G.C. Fox. The Gateway computational Web portal. *Concurrency Computat.: Pract. Exper.*, 14, pp. 1411-1426, 2002.

[23] Portable Batch System. http://www.openpbs.org.

[24] RRDtool. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool.

[25] T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo. An OGSA-Based Accounting System for Allocation Enforcement across HPC Centers. *The 2nd International Conference on Service Oriented Computing (ICSOC04),* ACM, 2004.

[26] K. Schuchardt, B. Didier, and G. Black. Ecce - a problem-solving environment's evolution toward Grid services and a Web architecture. *Concurrency Computat.: Pract. Exper.*, 14, pp. 1221-1239, 2002.

[27] Supercluster.org. Center for HPC Cluster Resource Management. http://www.supercluster.org.

[28] T. Suzumura, H. Nakada, M. Saito, S. Matsuoka, Y. Tanaka, and S. Sekiguchi. The Ninf Portal: An Automatic Generation Tool for the Grid Portals. *Proceedings of Java Grande 2002*, pp. 1-7, 2002.

[29] M. Thomas and J.R. Boisseau. Building Grid Computing Portals: The NPACI Grid Portal Toolkit. Texas Advanced Computing Center, Univ. of Texas at Austin.

[30] M. Thomas, M. Dahan, K. Mueller, S. Mock, C. Mills, and R. Regno. Application portals: practice and experience. *Concurrency Computat.: Pract. Exper.*, 14, pp. 1427-1443, 2002.

[31] M. Thomas, S. Mock, J. Boisseau, M. Dahan, K. Mueller, and D. Sutton. The GridPort Toolkit Architecture for Building Grid Portals. *Proc.10th IEEE Intl. Symp. On High Perf. Dist. Computing*, August 2001.

[32] W3C. Cascading Style Sheets, level 2. CSS2 Specification. http://www.w3.org/TR/1998/REC-CSS2-19980512/