BES++: HPC Profile Open Source C Implementation

Arkaitz Ruiz Alvarez
arkaitz@cs.virginia.edu
University of Virginia

Christopher Smith
csmith@platform.com
Platform Computing

Marty Humphrey
humphrey@cs.virginia.edu
University of Virginia

Abstract

While existing resource management software systems each have distinct and advanced capabilities, the way in which a user submits a basic job is generally similar across resource mangement systems. Recognizing this, the HPC Profile Working Group in the Open Grid Forum (OGF) has recently created a set of standards to define a common Web-services-based interface to resource management systems, thereby significantly enhancing interoperability. We describe the design and implementation of BES++, our open source implementation of the OGF HPC Profile. BES++ supports LSF, PBS and SGE resource managers and provides a C interface to further extend this software. In addition to complying with the OGF HPC Basic Profile specification, we support emerging OGF HPC Profile Extensions such as File Staging and Advanced Filter. We support job forwarding from one BES++ server to another, thereby enhancing potential transparency to the client. In addition, BES++ currently offers proof-of-concept support for legacy client tools such as qsub *by translating legacy scripts into invocations of our BES++ client. We evaluate the effectiveness of BES++ through microbenchmarks, assessment of correctness/interoperability, and ability to parse and translate legacy scripts. To our knowledge, this research is the first to comprehensively report on the challenges, issues, and evaluation of the implementation of the OGF HPC Profile specifications.*

1. Introduction

There are several resource management software systems to manage HPC computational clusters: e.g., the Load Sharing Facility [14] (LSF), the Portable Batch System [5] (PBS), Sun Grid Engine [9] (SGE) and, recently, Microsoft Windows HPC Server 2008 [12]. These resource managers add the jobs submitted by the users to a queue(s). Then, the resource manager processes the queue, assigning to each job available resources and dispatching the job for execution. Each resource management system has distinct and advanced capabilities to differentiate itself in the marketplace. Each software uses its own format for job description, submission and management. Thus, an organization with more than one cluster must often support multiple, different interfaces to largely similar backend resources.

The *HPC Basic Profile* [20] is a specification developed within the Open Grid Forum (OGF) aimed at solving the basic use case [17] of a generic HPC system. The *HPC Basic Profile* can be implemented by virtually every resource manager since it is based on a basic set of common functionalities that most resource management software provides. This specification is based on the *OGSA Basic Execution Service* (BES) [10] and the *Job Submission and Description Language* (JSDL) [2]. The BES-Factory port type offers an abstraction of the most basic interactions with a resource manager: create a job, check the status and attributes of a job, delete the job and check the status of the cluster. The JSDL document is an extensive specification for job description and submission. The subset ("profiling") of the JSDL document defined in the *HPC Basic Profile* is sufficient for basic use of a computational cluster.

In this paper, we describe the design and implementation of BES++, our open source implementation of the OGF HPC Profile. Our BES++ server is a *HPC Basic Profile*-compliant system that translates the job submission and management requests into the specifics of a particular resource management software. We currently offer interfaces to LSF, PBS, and SGE. We have architected the software for extensibility by providing a small set of functions (interfaces) that are the only routines that need to be implemented to adapt BES++ to an existing or new resource management system. To facilitate community involvement, we have recently started a BES++ sourceforge project [16] (with an open source GPL license).

In addition to complying with the OGF HPC Basic Profile specification, we support emerging OGF HPC Profile Extensions such as File Staging [21] and Advanced Filter [13], further enhancing the utility of the implementation. We support job forwarding from one BES++ server to another, in which a client can submit to a particular BES++ server and allow/request that BES++ server to forward the job to another BES++ server as warranted (e.g., to receive a faster execution on a lightly-loaded resource). Subsequent client interactions regarding the job (e.g., checking its state) take place as easily and as quickly as if the job had not been forwarded.

In addition, the use of BES++ does *not* require that potential users learn an entirely new way to submit jobs, which could significantly limit its impact and effectiveness. We report on prototype support in which an existing PBS user can continue on a limited basis to use the same PBS scripts but still benefit from the interoperability provided by BES++. By invoking our suite of replacement client-side binaries (e.g., by placing into $PATH a directory containing our replacements for *qsub, qstat, qdel, etc.* such that our tools are invoked), the originally-intended PBS job could execute on a different back-end such as an LSF-managed system. While we are continued to expand this functionality (e.g., parse/translate more keywords), we believe the proof-of-concept reported in this paper has long been sought by the HPC user community and is an important step forward.

We evaluate the effectiveness of BES++ through microbenchmarks, assessment of correctness/interoperability, and ability to parse and translate legacy scripts. For example, in our limited evaluation of our prototype support for legacy clients, we evaluated a small but representative set of real PBS scripts currently being used by researchers at the University of Virginia and found that 23.81% of the existing scripts could be translated and submitted via our system without *any* loss of functionality and 100% of the scripts could be submitted such that the job be executed but perhaps with some of the non-critical functionality not supported (e.g., the HPC Profile has no support for sending email upon completion of the job). To our knowledge, this research is the first to comprehensively report on the challenges, issues, and evaluation of the implementation of the OGF HPC Profile specifications.

2. Related Work

Our BES++ software is based on the *HPC Basic Profile* [20] and its extensions [4, 13, 21]. A "compliant" implementation *must* implement the Basic Profile but the implementation of one or more of the Extensions is optional. Specific requirements for the HPC Profile have been created to guide the effort in the Open Grid Forum [17]. In addition to our software, there are other grid projects [7, 15, 6, 11] that implement the profile (Note: to our knowledge, none of these projects have attempted to assess their specific HPC Profile implementation as per this paper).

There has been other efforts to attempt to reduce the complexity of remote job submission through common protocols and APIs. Most notably, Globus GRAM [8] offers a similar service for submitting jobs to computational resources. The goal of the HPC Profile effort is to reduce the complexity via a minimal functionality of the HPC Basic Profile with additional capabilities provided via optional extensions. In contrast to GRAM, the HPC Profile effort began as a community-driven activity in the Open Grid Forum, which we believe has resulted in unique and broad community involvement from academia and industry (e.g., the number of independent implementations).

Job forwarding in general is similar to the the concept of metascheduling for the grid (e.g., [18]). While we are *not* attempting sophisticated metascheduling in BES++, we believe our job forwarding capabilities can be significantly leveraged by such metascheduling algorithms. We are currently investigating creating an easy-to-use plugin archecture for adding such third-party metaschedulers.

3. BES++

The BES++ server and client implements the *HPC Basic Profile*, which is based on the following standards: the *OGSA Basic Execution Service* (BES), for the operations related to job scheduling and management, and the *Job Submission Description Language* (JSDL), which specifies the format in which jobs are submitted. The BES-Factory port type is sufficient to implement the required operations to support the basic use case of HPC systems. BES-Factory includes the following operations: *CreateActivity, GetActivityStatuses, TerminateActivities, GetActivityDocuments* and *GetFactoryAttributeDocuments*. An *activity* represents the concept of *job* in traditional resource managers such as LSF, PBS, or SGE. Thus, this port type provides the means to start, delete and query the status of jobs running in a computational cluster. We can also dynamically obtain the state of the cluster, which includes the jobs that exist in the queue and the current availability of the computational nodes.

We use the gSOAP [19] Web service development toolkit for the implementation of the client and server. This toolkit provides an automated way of mapping the schema descriptions of the standards used into C language constructs. Our server and client are coded in C, with some Python scripts.

To comply with the HPC Profile specifications, BES++ uses X.509 certificates [1] for server authentication and communication encryption (TSL/SSL). Within the BES++ server, client authentication is done via Pluggable Authentication Modules (PAM) using the username and password token that the client includes in the request (WS-Security [3]). Note that this does not mean that clients authenticate only via username/password; rather, BES++ also implements X.509 client certificate authentication of the HPC Basic Profile. Operations that require additional credentials, such as file staging, are described in Section 3.3.2.

BES++ implements the existing HPC Profile Extensions that are currently under discussion and standardization in

```
<CreateActivity>
 <ActivityDocument><JobDefinition>
  <JobDescription>
   <JobIdentification>
    <JobName>TestJob</JobName>
   </JobIdentification>
   <Application><HPCProfileApplication>
     <Executable>grep</Executable>
     <Argument>"HPCBP"</Argument>
     <Argument>"*"</Argument>
     <WorkingDirectory>/home/user/tmp/</WorkingDirectory>
     <Output>/home/user/tmp/output.txt</Output>
     <Input>/home/user/tmp/input.txt</Input>
     <Error>/home/user/tmp/error.txt</Error>
     <UserName>user</UserName>
     </HPCProfileApplication></Application>
   <Resources>
     <CandidateHosts>
       <Hostname>centurion002</Hostname>
     </CandidateHosts>
     <TotalCPUCount>1</TotalCPUCount>
   </Resources></JobDescription>
   </JobDefinition></ActivityDocument>
</CreateActivity>
```

Figure 1. **A request for creating activity "TestJob"**

the Open Grid Forum [13, 4, 21]. These extensions enhance the profile by adding new features such as support for file staging or the use of various type of credentials.

Our implementation is Open Source and is licensed under the terms of the GNU General Public Licence version 2. The code and documentation is available for download [16]. Our current version requires gSOAP 2.7.9f and Python 2.3.

3.1. BES++ client

Our BES++ client is a command line tool that provides the user a simple interface to any *HPC Basic Profile* compliant systems. The usage of the tool is described as follows:

```
besclient <auth options> CApath <endpoint>
      <username> <command> [<commandargs>]
```

BES++ client can be used with systems that authenticate users exclusively via username/password or with systems that also use X.509 client certificates. In both cases the communication is encrypted via SSL and the server provides a X.509 certificate that is validated by our client. The username and password are included in the SOAP Header and are mapped in the server to a UNIX account. We use WS-Addressing to describe the server's endpoint. The commands that the BES++ client can perform are the ones described in the BES-Factory port type, which could require additional arguments. For example, a *CreateActivity* operation requires a JSDL activity description file (e.g., Figure 1), or a *GetActivityStatuses* operation requires an endpoint reference (EPR) that identifies an activity.

The output of this command line tool is the server's response to the client's request:

- *create*: a new activity identifier and the confirmation of a correct submission.
- *status*: the status of an activity.
- *activity*: a XML document with the description of an activity.
- *delete*: the confirmation of the deletion of an activity.
- *factory*: a XML document which lists all the attributes of the BES-Factory (an *HPC Basic Profile* server).

For any operation that failed, a SOAP Fault element with the type and the description of the fault. This fault may have been originated by an incorrect request from the client or a server error.

3.2. BES++ server

The multithreaded BES++ server listens for incoming requests and sends them to the head node of a computational cluster (which may be running in other physical machine). The server interprets the XML request and converts it to a valid resource manager command or API call. We can compile the server to communicate with three different resource managers: LSF, PBS or SGE. The implementation of the server is easily extensible to other resource management systems. Figure 2 shows the general interface which should be implemented by a module in order to add a new resource manager to the list of supported software.

Currently, the LSF and PBS modules use the C interface that both resource managers provide to programmatically interact with the queues, except for the job submission. The implementation for SGE, on the other hand, uses scripts that call the command line tools (qsub, qstat and qdel) and parse the output text before returning this information to the BES++ server. A developer who wishes to extend the BES++ server could choose to take advantage of the resource manager's programmatic interface (if it is available) or use the basics of the SGE implementation and adapt the scripts to the specifics of the resource management software. Table 1 shows the mapping of each BES-Factory port type operation to the different resource managers.

The BES++ server must present a valid X.509 certificate to the client. All the communication is encrypted using SSL. The server may or may not require the client to have a X.509 certificate, depending on the command line arguments when we start the server. However, the client is always required to submit a WS-Security UsernameToken in the SOAP header. This username is the UNIX account which the server will use to perform the operation requested. The BES++ server runs with root privileges, so it can set the UID of the process to the identified user's and interact with the resource manager using the user's credentials. Our implementation checks the validity of the user's account using PAM, although this mechanism can be easily substituted.

```
int initialize(struct soap* s, char* server);
/* GetFactoryAttributes */
struct clusterInfo* getClusterInfo(
                              struct soap* s);
/* CreateActivity */
int submitJob(struct soap* s, struct jobcard*
              jobinfo, char** error_message);
/* GetActivityStatuses */
int isJobFinished(char* status);
int isJobStarted(char* status);
int isJobPending(char* status);
int getJobStatus(long int jobid, char* user,
                 char** status);
/* GetActivityDocuments */
int getJobInfo(struct soap*, long int jobid,
               struct jobcard** jobInfo);
JobList* getJobsList(struct soap*, Filters*);
/* TerminateActivities */
int deleteJob(long int jobID, char* result);
/* Debugging an logging */
void printError(char* userMessage);
int getErrNo();
```

Figure 2. C interface to implement for supporting a resource manager

Table 1. BES-Factory port type mapping to resource manager API or commands

Operation	LSF	PBS	SGE
CreateActivity	bsub	qsub	qsub
GetActivityStatus	lsb_openjobinfo	pbs_statjob	qstat
GetActivityDocument	lsb_openjobinfo	pbs_statjob	qstat
DeleteActivity	lsb_signaljob	pbs_del_job	qdel
GetFactoryAttributes	ls_gethostinfo	pbs_statserver	qstat

3.3. Extensions to the Basic Profile

In the OGF HPC Profile effort, “Extensions” add functionality that is out of the scope of the basic use case that the *HPC Basic Profile* represents. In this section, we focus on the three key Extensions we have implemented in BES++.

3.3.1 File Staging

For many users, the ability to stage files in (before the computation happens) or out (after) is important. Thus, our the HPC Profile File Staging Extension [21] defines a standardized XML format for specifying such file movement. Our BES++ server implements this specification and is is able to stage files using the following protocols and tools: FTP, HTTP, scp and GridFTP. The BES++ server is able to find and use the necessary credentials for file staging in the user's request. Our BES++ server identifies and processes the *DataStaging* elements in a user's request, which are part of the JSDL specification but not required by the *HPC Basic Profile*. The elements in this complex type which we support are: *FileName, DeleteOnTermination, Source* and *Target*. The definition of these elements is the same as in the JSDL specification. The *Source* and *Target* elements are URIs of the type: http://, ftp://, scp:// or gridftp://.

Some of the protocols we use for file staging are currently unsupported by resource managers. Thus, our BES++ software is responsible for adding the necessary file staging capabilities. A common characteristic of every resource manager we support is that the job submission can be done via a shell script. This script includes the executable and the arguments the user has submitted via *HPC Basic Profile*. In the submission process, we surround this command in the script with commands that perform the file staging. We have implemented these commands in a Python script located in a well-known directory.

3.3.2 Activity Credential

The support for File Staging described in section 3.3.1 would not be possible without providing a mechanism for the user to give the BES++ server the appropriate credentials. We implement the *HPC Common Case Profile: Activity Credential V. 0.1* extension [4]. This specification defines a new element to be included in the XML request for the *CreateActivity* operation. This new element (*ActivityCredentials*) includes a list of *Credentials*. Each *Credential* includes a *UsernameToken* with a password, and a *AppliesTo* element that points to the *DataStaging* element for which this credential should be used.

Figure 3 provides an example of an *ActivityCredentials* element, in which we will use the scp username/-password provided for the files that come from or go to the machine www.deli.deusto.es (again, note that this is over SSL so username/password is not sent in cleartext). HTTP (wget) does not require any credentials and can be used only for staging in files. For GridFTP, we have included a username and password to a MyProxy service at myproxy.ncsa.uiuc.edu. The credentials obtained from this service are then consumed by the globus-url-copy command.

3.3.3 Advanced Filters

The HPC Basic Profile operation *GetFactoryAttributesDocument* returns a list of all the jobs currently on the system and a list of resources that the factory contains (which includes a short description of the resource and its status). Because the response to this operation might be unnecessarily large, particularly if the client is only interested in a subset of the jobs or resources, the *HPCBP Advanced Filter Extension, V 0.1* [13] provides way for the client to set the level of detail that the server will use in the response to *GetFactoryAttributesDocument*.

```
<cred:ActivityCredentials>
 <Credential>
  <UsernameToken>
   <Username>userDeusto</Username>
   <Password>passDeusto</Password>
   <AppliesTo>
    scp://www.deli.deusto.es
   </AppliesTo>
  </UsernameToken>
 </Credential>
 <Credential>
  <UsernameToken>
   <Username>userMyproxy</Username>
   <Password>passMyproxy</Password>
   <AppliesTo>
    myproxy://myproxy.ncsa.uiuc.edu
   </AppliesTo>
  </UsernameToken>
 </Credential>
</cred:ActivityCredentials>
```

Figure 3. Example of the Activity Credential Extension

The *AdvancedFilter* element in a XML request is a complex type composed by a series of elements:

- *UserName*: includes only jobs executing under this username.
- *State*: include only jobs in the state given.
- *ActivityIdRange*: include only jobs whose numerical id is within a given range.
- *DateTimeRange*: include only jobs submitted within a certain date and time range.
- *CompactResources*: include only the complete description of all resources, free resources or used resources.

The application of filter(s) may result on the server returning no list of jobs or resources. In this case, the server still returns the value of some general attributes, the total number of jobs in the cluster and the total number of CPUs available. A use case in which this feature is useful can be found in Section 4, in which we only need to collect the number of available CPUs in each cluster (a BES Factory).

4. Job Forwarding

To enhance the value of BES++, we have created support for job forwarding from one BES++ server to another, in which a client can submit to a particular BES++ server and allow/request that BES++ server to forward the job to another BES++ server as warranted (e.g., to receive a faster execution on a lightly-loaded resource). Subsequent client interactions regarding the job (e.g., checking its state) take place as easily and as quickly as if the job had not been forwarded. Note that the concept of Job Forwarding is anticipated and enabled by the HPC Profile standards efforts, but there will be no specific "standard" that defines how to do this (because its functionality is internal to a particular HPC Profile server).

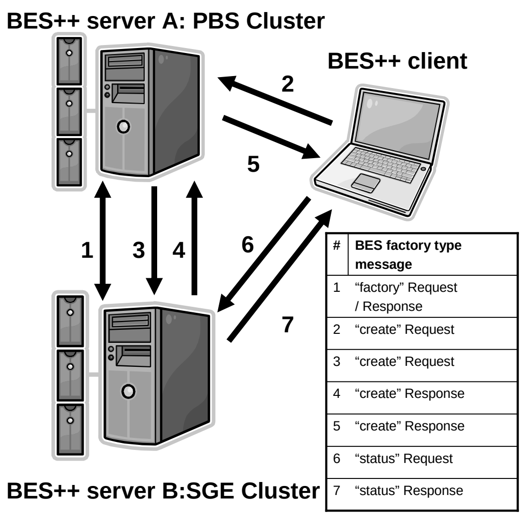
Figure 4 shows the interactions that take place between a BES++ client and two BES++ servers, in which the client creates a new activity and checks its status:

1. The BES++ servers check each other's status at regular intervals using *GetFactoryAttributesDocument*. The response includes the total number of CPUs available, the status of each node and the total number of jobs running in each cluster. This can be configured to determine this information only in direct response to a client submission.
2. The BES++ client submits a *CreateActivity* request to BES++ server A in the PBS cluster.
3. The PBS cluster is overloaded, so the BES++ server A forwards the *CreateActivity* petition to the BES++ server B.
4. The BES++ server B submits the job to the SGE cluster and returns the response with the job id (https://besppB.server.org:443/jobIDinSGE) to BES++ server A.
5. BES++ server A returns the response to the client. Now the client knows that the job id points to BES++ server B.
6. The client checks the status of the job submitted with the BES++ server B (*GetActivityStatus*).
7. BES++ server B returns the job status in the SGE queue to the client.

This capability is meant to show proof-of-concept and relies on a number of simplifying assumptions. For example, this makes a number of assumptions that are likely to be valid within a particular enterprise but unlikely to be true in general (e.g., the participating BES++ servers can all authenticate and authorize based on the single credential, the participating BES++ servers share a common file system such as an NFS tree, etc.) In addition, the current job forwarding decision is based on the number of available CPUs in each cluster. We plan to expand this functionality in the future. Even with the restrictions that we have mentioned, we feel that this extension to the BES++ server can greatly increase the flexibility for job scheduling.

5. Legacy clients

Since the BES++ server is an *HPC Basic Profile*-compliant system, other *HPC Basic Profile*-compliant clients can be used. This scenario still imposes a restriction



#	BES factory type message
1	“factory” Request / Response
2	“create” Request
3	“create” Request
4	“create” Response
5	“create” Response
6	“status” Request
7	“status” Response

Figure 4. Job forwarding

to the user: her jobs should be described in XML format and submitted by using Web services. We feel that we should avoid imposing such a restriction in an existing HPC enterprise. Thus, we have pursued support for *legacy clients*. We consider a *legacy client* to be the set of binary tools that are part of the resource management software and support the user’s direct interaction with it: qsub, qstat and qdel are the basic tools. qsub submits a new job described by a script file (which is a command line argument) and returns the id of the created job. qstat and qdel have a job id as their input and return the status of the given job or cancel it, respectively. We have developed new corresponsing tools for PBS that offer a similar interface to the user, but internally they use the BES++ client. Thus, a user can still use a legacy script file composed to be submitted to the PBS system and use it with our qsub. The program converts the job script, like the one shown in Figure 5, to a valid *CreateActivity* request. This process can be seen as the inversion of the process that the BES++ server does, which takes a JSDL document, composes a valid PBS script file and submits it to the resource manager. Our new qsub will submit this file to the default BES++ server (which may or may not forward the job submission), parse the response and return a job id formatted in the legacy syntax. Our qstat and qdel for PBS work similarly.

HPC Basic Profile is an specification that covers a basic use case of an HPC system. Even with the additional extension we have implemented, it is not currently feasible to cover all the options that qsub (in LSF, PBS or SGE) accepts as input. Therefore, the interface of our qsub is identical if the script file uses the options for the features that are allowed in the *HPC Basic Profile*. Otherwise, we report to the user that her job can not be mapped to a valid *CreateActivity* request due to the use of a certain unsupported option. As part of our evaluation described in the next section, we show that we are currently able to sufficiently support a large percentage of our test scripts, and we plan to expand this support in the near future.

```
#!/bin/bash
#PBS -N Prueba
#PBS -o /home/user/tmp/output.txt
#PBS -e /home/user/tmp/error.txt
#PBS -u user
#PBS -l nodes=1
#PBS -l host=centurion002

cd $PBS_O_WORKDIR
grep "HPCBP" *
```

Figure 5. A PBS script file that BES++ translates to a request equivalent to Figure 1

6. Evaluation

In this section we evaluate our BES++ implementation. *Note that our goal in this section is to evaluate BES++; it is not our goal to evaluate the HPC Profile set of specifications.* We first present the performance for each of the BES++ server operations. Second, we show the correctness/interoperability of the BES++ server in relation to other compliant software. Third, we evalute the BES++ support for legacy clients. We do not specifically report the performance of job forwarding because we have found it to be consistent with a linear combination of our measurements for the individual operations on a single BES++ service (i.e., the cost to “decide” to forward does not require significant duration).

6.1. Microbenchmarks

We show the microbenchmarks results on Figure 6. We have run 100 times a set of experiments which include a request for each of the BES-Factory port type operations. This microbenchmark was performed against four BES++ servers: one BES++ server for each resource manager type (PBS and SGE) and each client authentication method (X.509 certificates and username/passsword). The durations reported are from the perspective of our client endpoint. Adding X.509 client certificate authentication does not impact significantly the running time of each operation. The biggest impact is the type of interfaces to the resource manager used: C library calls are significantly faster

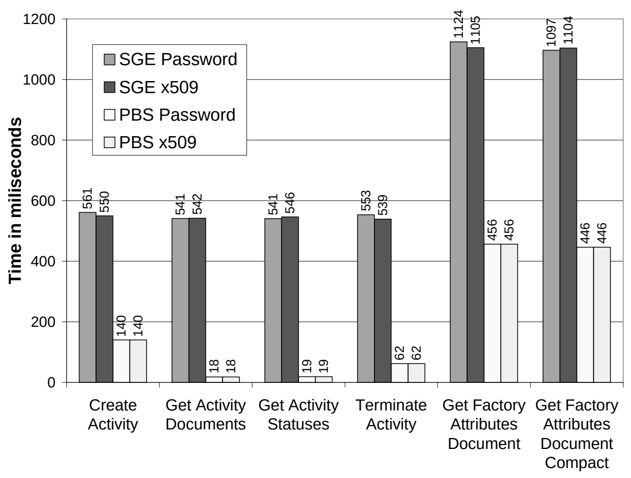


Figure 6. **Performance of BES++ operations**

than forking and calling the commands of the resource manager. BES++ server for SGE calls a Python script that performs the execution of qsub, qstat and qdel and parses the output text. Developing an interface for a resource manager similar to our interface to SGE is significantly easier than using C library calls (if available). The trade-off is the overhead imposed by this scheme: around 500 ms more for each incoming request. Overall, we consider that the performance of our BES++ server satisfies the requirements, since the cost of performing operations (submit, delete, etc.) on jobs running on a cluster is typically negligible (although not always) compared to the running time of the jobs.

6.2. Correctness/Interoperability

We have demonstrated the correctness and interoperability of BES++ most recently at the Supercomputing 2007 conference. The BES++ server was tested against six other implementations of the *HPC Basic Profile*. These implementations were: University of Virginia .NET implementation, Microsoft HPC Group, EGEE2/OMII-Europe CREAM-BES, Nordugrid/-KnowARC A-REX, Forschungszentrum Juelich UNICORE and GridSAM/OMII-UK. The interoperability testing included a test case for each of the BES-Factory port type operations. Our BES++ server successfully interacted with each of the other clients for each test case.

A useful contribution from the University of Virginia is the *HPC Basic Profile Compliance Tester*, which allows to test a BES++ server under multiple operations.This web-based tester generates a series of test cases for each operation. Each test case sends a correct request (for testing the correct operation) or an incorrect one (for testing the correct generation of SOAP Faults and error descriptions). The response is then analyzed for compliance with the schema. Our BES++ server successfully passes this compliance test. Overall, we assert through the results of this testing that our software offers a high level of interoperability with other software that implement the specification.

6.3. Legacy clients

In this section we analyze the support for legacy clients such as PBS's qsub. Figure 7 includes the performance of our qsub tool described in Section 5. The use of this tool, compared to the BES++ client, adds between 200 and 250 ms to the total execution time. If the user wants to substitute PBS's qsub with this tool and take advantage of a system based on *HPC Basic Profile*, the increased time will vary between 500 and 750 ms, depending on the implementation of the server (Python like SGE or C like PBS). We believe this time penalty is not significant enough to prevent widespread use of this tool.

As we have mentioned in Section 5, *by design*, the *HPC Basic Profile* does not cover all possible options of qsub, so there certainly will exist PBS scripts for which BES++ cannot provide this transparency and ease of use. However, to attempt to test BES++'s ability to support "ordinary" scripts, we obtained a small but representative set of real PBS scripts currently being used by researchers at the University of Virginia on central PBS resources. Table 6.3 contains the result of our analysis. First, we have counted the options used and rank them from the most used to the least used. We can see that most of the scripts included the *nodes*, *ppn* (processors per node) and *walltime* options. We can classify these options in three groups: options that map directly to elements in the specification (*supported*), options that can not be expressed with the profile but do not affect the correct execution of the job (*warnings*) and options that can not be expressed with the profile and will cause execution errors (*errors*). An example of a supported options is *ppn*, since we have the element *TotalCPUCount*. An example of a warning is the option *m* (send mail a user on certain conditions). If we ignore this option the job will continue to execute correctly, but some additional functionality (notifying the job owner) is lost. Finally, some options can not be expressed using the profile, such as job interdependences. Table 6.3 shows that 23.81% of the jobs use exclusively supported options. This percentage can grow to 62% if we implement support for walltime (the remaining 38% of the jobs that use mailing options that are warnings.) *Walltime* is in the JSDL specification but not in the *HPC Basic Profile*. If we ignore the warnings, the percentage of the jobs that can be submitted grows to 100% in our sample. While we recognize that in many situations something like "walltime" cannot be ignored, we note that in general this does not mean that user can use our qsub to, in effect, run a job forever – in general, the server still enforces such constraints.

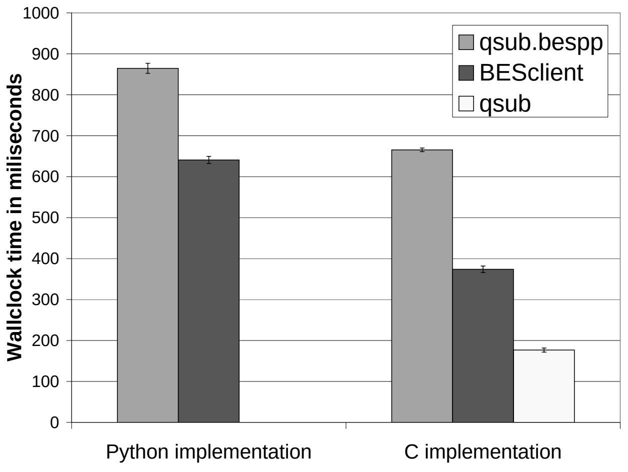


Figure 7. **Performance of job submission**

Table 2. **Most common PBS QSUB options**

Option	% of scripts
nodes	95.24%
ppn	90.48%
walltime	76.19%
j	42.86%
o	38.10%
m	38.10%
M	38.10%
N	4.76%

Options used	% of scripts
Supported	23.81%
Supported with warnings	100%
Unsupported	0%

7. Conclusions

While the use of HPC resources continues to increase, there is also an increasing necessity for improved resource management and sharing. The OGF HPC Profile effort offers a new and lightweight approach to interoperability between HPC resources. The BES++ software is a comprehensive implementation of the *HPC Basic Profile* that currently supports LSF, PBS and SGE, includes several extensions to the profile, features an extensible architecture and has added capabilities such as job forwarding and support for legacy clients. We have evaluated BES++ and shown that we can achieve flexible job forwarding mechanisms. Our emerging support for legacy clients has strong potential to increase adoption – in our tests, we found that 23.81% of the existing PBS scripts that we tested could be translated and submitted via our BES++ support for legacy clients without *any* loss of functionality and 100% of the scripts could be submitted such that the job be executed at reduced functionality. In the future, we plan to continue this research to provide the HPC community an open source, extensible and interoperable standards-based implementation for support of HPC resources.

References

[1] C. Adams and S. Farrell. RFC2510: Internet X. 509 Public Key Infrastructure Certificate Management Protocols. *Internet RFCs*, 1999.

[2] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification. *GGF GFD*, 56, 2005.

[3] B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manferdelli, H. Maruyama, et al. Web Services Security (WS-Security).

[4] J. Basney, B. Dillaway, M. Humphrey, and G. Wasson. HPC Common Case Profile: Activity Credential, Version 0.1. https://forge.gridforum.org/projects/ogsa-hpcp-wg/.

[5] A. Bayucan, R. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. Portable Batch System: External reference specification. *MRJ Technology Solutions, May*, 1999.

[6] J. Bradley, C. Brown, B. Carpenter, V. Chang, J. Crisp, S. Crouch, D. De Roure, S. Newhouse, G. Li, J. Papay, et al. The OMII Software Distribution. *Proceedings of the UK e-Science All Hands Meeting*, pages 748–753, 2006.

[7] D. Erwin et al. UNICORE:a Grid computing environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410, 2002.

[8] I. Foster and C. Kesselman. The Globus project: a status report. *Heterogeneous Computing Workshop, 1998.(HCW 98) Proceedings. 1998 Seventh*, pages 4–18, 1998.

[9] W. Gentzsch et al. Sun Grid Engine: Towards Creating a Compute Power Grid. *Proceedings of the First IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002.

[10] A. Grimshaw, S. Newhouse, D. Pulsipher, and M. Morgan. OGSA Basic Execution Service Version 1.0. *Open Grid Forum*, 2006.

[11] W. Lee, A. McGough, and J. Darlington. Performance Evaluation of the GridSAM Job Submission and Monitoring System. *UK e-Science All Hands Meeting*, pages 915–922.

[12] Microsoft Comporation. Windows HPC Server 2008. http://www.microsoft.com/windowsserver2003/ccs/default.aspx.

[13] S. Newhouse. HPCBP Advanced Filter Extension, Version 0.1. https://forge.gridforum.org/projects/ogsa-hpcp-wg/.

[14] Platform Computing. The Load Sharing Facility. http://www.platform.com.

[15] O. Smirnova, P. Eerola, T. Ekelof, M. Ellert, J. Hansen, A. Konstantinov, B. Konya, J. Nielsen, F. Ould-Saada, and A. Waananen. The NorduGrid Architecture and Middleware for Scientific Applications. *Lecture Notes in Computer Science*, 267:264–273, 2003.

[16] C. Smith and A. Ruiz Alvarez. BES++ project. http://sourceforge.net/projects/bespp.

[17] M. Theimer, C. Smith, and M. Humphrey. HPC Job Scheduling: Base Case and Common Cases. https://forge.gridforum.org/projects/ogsa-hpcp-wg/.

[18] S. Vadhiyar and J. Dongarra. A metascheduler for the Grid. *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 343–351, 2002.

[19] R. van Engelen and K. Gallivan. The gSOAP toolkit for Web services and peer-to-peer computingnetworks. *Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002*, pages 117–124, 2002.

[20] G. Wasson. HPC Basic Profile, Version 1.0. https://forge.gridforum.org/projects/ogsa-hpcp-wg/.

[21] G. Wasson and M. Humphrey. HPC File Staging Profile, Version 0.1. https://forge.gridforum.org/projects/ogsa-hpcp-wg/.