BEESYCLUSTER AS FRONT-END FOR HIGH PERFORMANCE COMPUTING SERVICES

PAWEŁ CZARNUL

Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology
Narutowicza 11/12, 80-233 Gdansk, Poland

(received: 14 May 2015; revised: 19 June 2015;
accepted: 26 June 2015; published online: 1 October 2015)

Abstract: The paper presents the BeesyCluster system as a middleware allowing invocation of services on high performance computing resources within the NIWA Centre of Competence project. Access is possible through both WWW and SOAP Web Service interfaces. The former allows non-experienced users to invoke both simple and complex services exposed through easy-to-use servlets. The latter is meant for integration of external applications with services made available from clusters or servers. Details of services such as APIs used for development as MPI, OpenMP, OpenCL as well as queuing systems are hidden from the user. The paper describes both the WWW and Web Service interfaces extended for use with files of large sizes. Mechanisms for selection of devices for execution of services are described along with experiments including remote invocations.

Keywords: BeesyCluster, high performance computing, services, Web Service interface

1. Introduction

In recent years, we have witnessed constant growth of High Performance Computing (HPC) systems. This applies to single node systems as well as clusters both of which have increased their parallelization potential by packing more processing units. Each node now features one or more multicore CPUs with an increasing number of cores. For instance, the Intel Xeon Processor E5-2699 v3 features 45M Cache, is clocked at 2.30 GHz, 3.6 GHz turbo and has 18 cores with the possibility to run 36 threads efficiently. Apart from CPUs capable of executing the general purpose code efficiently, servers and workstations make use of accelerators and coprocessors. For instance, the Intel Xeon Phi Coprocessor 7120A features 16 GB, is clocked at 1.238 GHz, has 61 cores with the possibility to run 244 threads efficiently. NVIDIA Tesla K80 features 24 GB of memory and 4992 CUDA cores and offers up to 2.91 TFlop/s of double precision floating point performance. Such devices have transformed single computers into parallel

systems. Furthermore, nodes packed with such devices can still be combined into clusters offering even more computing power. The TOP500 list [1] features Tianhe-2 (MilkyWay-2) at the top with 3120000 cores offers 33862 TFlop/s performance. It features Intel Xeon E5-2692 CPUs clocked at 2.200 GHz with Intel Xeon Phi 31S1P coprocessors. It is now more than ever that such resources need easy-to-use interfaces for deployment, usage and publishing of applications to users who act as clients of HPC services.

The goal of this paper is to present ways of exposing and consuming computational services through the BeesyCluster system which can be regarded as a middleware and front-end to a collection of clusters and servers. Services are published from clusters or servers registered in the BeesyCluster system by users of the latter. From a consumer point of view this allows calling such services without the need for taking care of low level details on the cluster side.

2. Related Work

Efficient usage of such cluster resources is possible through one of the two ways: running ready-to-use domain software or in-house development of custom parallel codes. Examples of the former include: General Atomic and Molecular Electronic Structure System (GAMESS) – a general ab initio quantum chemistry package[1], Gaussian – a package for electronic structure modeling[2], ArcGIS – a package for geospatial data, information management and analysis[3]. For the latter, there are several Application Programming Interfaces (APIs) available. For instance, OpenMP [2] allows parallelization of sequential applications by inserting directives indicating parallel regions of code and usage of a library of functions that assist in thread level parallelism for shared memory systems. Pthreads is the traditional low level API for thread management and synchronization that can be used for programming many core systems with shared memory. Nowadays parallel programming for GPUs is usually performed with one of the following APIs: NVIDIA CUDA [3], OpenCL [4] or OpenACC [5]. NVIDIA CUDA proposes the application structure as a grid composed of blocks which contain threads where parallelism comes from scheduling many independent blocks. OpenCL proposes a similar approach – the grid is called NDRange which is composed of work groups each of which contains a number of work-items. OpenCL allows mapping an application not only to GPUs (from various vendors) but also multicore CPUs. Programming Intel Xeon Phi [6–9] is possible using many APIs such as OpenMP [10], OpenCL [4] and Message Passing Interface (MPI) [11]. MPI has been and is still used as the leading MPI for programming distributed memory systems *i.e.* clusters of machines connected with fast interconnects. MPI allows both processes running on nodes of a cluster as well as threads within

1. http://www.msg.ameslab.gov/gamess/
2. http://www.gaussian.com/
3. http://www.esri.pl/

processes. Communication routines involve both point-to-point and collective routines involving both data exchange and synchronization such as barrier.

Programs can be run on a cluster from a command-line on per user basis. Alternatively, execution of applications from many users can be managed by queuing systems. Typically used systems include Portable Batch System (PBS)[4], TORQUE[5], Load Sharing Facility (LSF)[6].

It should be noted that clusters, often managed by various institutions can be combined into grid systems. In these cases, a grid middleware constitutes a layer that provides a uniform interface to resources hosted by various Virtual Organizations (VOs). Examples of such middlewares are Globus Toolkit[7], UNICORE[8] or gLite (Lightweight Middleware for Grid Computing)[9]. Grid middlewares expose services such as file and data management as well as job submission. Grid middlewares can use meta schedulers which incorporate queuing systems as job managers. For instance, in Globus Toolkit, the Grid Resource Allocation Manager (GRAM) can use PBS for job scheduling.

Additionally, web based interfaces or clients have been proposed for easy management of these functions, hiding low level details and exposing interfaces to specific applications hosted on clusters. As an example, GridSpace[10] is an environment that allows users to define and perform experiments that use underlying compute and storage resources [12]. It allows collaboration of scientists and can manage resources using PBS. The UNICORE Rich client is a graphical interface for remote management of files and job submission to a grid with the possibility to define workflows [13]. Vine Toolkit [14, 15] offers graphical centralized access to computational and storage resources, a file manager, application submission forms, status indication as well as visualization of results. In the PL-Grid system, users can use a graphical interface for accessing a remote machine and running applications there[11]. For instance, the following applications can be run in this way: Abaqus, Matlab, Avogadro, PMV. In Amazon Elastic MapReduce applications such as Ganglia, Hadoop publish interfaces as web sites[12].

4. http://www.pbsworks.com/PBSProduct.aspx?n=PBS-Professional&c=Overview-and-Capabilities

5. http://www.adaptivecomputing.com/products/open-source/torque/

6. http://www-03.ibm.com/systems/platformcomputing/products/lsf/

7. http://toolkit.globus.org/toolkit/downloads/latest-stable/

8. http://www.unicore.eu/

9. http://grid-deployment.web.cern.ch/grid-deployment/glite-web/

10. https://gs2.plgrid.pl/userguide

11. https://docs.cyfronet.pl/display/PLGDoc/UI+-+GUI

12. http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-connect-ui-console.html

3. BeesyCluster Interfaces and Use Cases

BeesyCluster is a middleware that allows users to access distributed resources such as clusters, servers and workstations. BeesyCluster offers two interfaces:

1. An easy-to-use Web interface with a drag&drop file manager, specialized editors for code development, team work environment. BeesyCluster offers single sing-on into many system accounts registered on clusters. Applications accessible to the user can be published as services and proper privileges can be granted to either individual users or user groups to invoke such services. The latter can be combined into workflow applications which are optimized, scheduled and executed [16, 17]. The typical usage of this interface is described in Section 3.1.
2. Web Services which expose functions such as invocation of a command on a cluster, invocation of a service, file upload/download. These are described in Section 3.2.

As a component of the NIWA Centre of Comptetence[13], BeesyCluster is a front-end and middleware to cluster systems allowing batch processing of high performance computing jobs such as demanding numerical computations.

A BeesyCluster instance[14] deployed at the Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland has already been used for teaching high performance computing systems and integration of services in distributed environments [18]. Features such as file manager, compilation, running applications in parallel on a cluster, team work features, an editor for integration of services into a workflow application, running workflows [19] have been used. BeesyCluster, contrary to many other grids middlewares, uses ssh in order to access system accounts on distributed resources. The following sections describe extensions of the NIWA special deployment of BeesyCluster[15] and their features.

3.1. WWW Interface and Usage

The Web interface available in BeesyCluster offers the following functions to users:

- Single sign-on access to an array of clusters, workstations and servers on which the user has been granted system accounts;
- A file manager allowing management of files and directories, copying, compilation, running, invocation of any system commands. Figure 1 presents a file manager with a context menu launched for one of files on a cluster. Figure 2

13. http://niwa.gda.pl/

14. https://beesycluster.eti.pg.gda.pl:10030/ek/AS_LogIn

15. available at https://mayday-apl.task.gda.pl:55443/ek/Main in the internal network of the Gdansk University of Technology

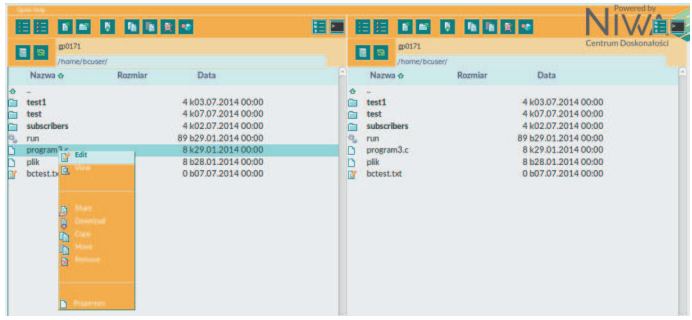

Figure 1. BeesyCluster's file manager

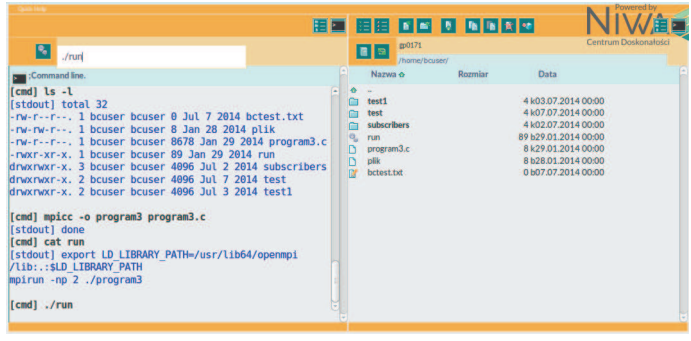

Figure 2. Compilation of a parallel application using *mpicc* and running using a script in BeesyCluster's file manager

```
[cmd] mpicc program3.c -o program3
[stdout] done
[cmd] cat run
[stdout] export LD_LIBRARY_PATH=/usr/lib64/openmpi
/lib:.:$LD_LIBRARY_PATH
mpirun -np 2 ./program3

[cmd] ./run
[stdout]
Hi, I am process 0, the result is 2.516266
```

Figure 3. Result of running a parallel application using *mpirun*

presents a compilation of a parallel application using *mpicc* while Figure 3 the result of running the application using *mpirun*.

- Hiding queuing systems such as PBS or LSF on clusters;
- Team work support including exchange of messages, definition of projects, milestones, a shared board *etc.*;
- Publishing applications as services and assignment of rights to invoke such services either to individual users or groups;
- Wiki module.

3.2. Web Service Interface

The Web Service BeesyCluster interface described in [20] has been extended with support for remote invocation of not only arbitrary commands on a cluster but also services made available by other users as well as handling files of large

sizes that might be passed as input. The key operations exposed within Web Services include:

1. *String[] logIn(String[] credentials, long loginagentID, long signerID)* – for logging in and obtaining an authenticator used for subsequent Web Service calls:
   - *credentials* – identification data,
   - *loginagentID* – id of an identification module (algorithm),
   - *signerID* – id of a signer module (algorithm),
   - *returned value* – authenticator used for subsequent Web Service calls.
2. *String runCommand (String[] authenticator, int clid, String command)* – running a given command on the given cluster:
   - *authenticator* – returned previously by *logIn*,
   - *clid* – id of a cluster to run the command on,
   - *command* – command to launch on the cluster,
   - *returned value* – standard output from the command.
3. *String runService (String[] authenticator, String service, String inputData)* – running a service with a given name made available before and for which access should have been granted to the calling user:
   - *authenticator* – returned previously by *logIn*,
   - *service* – name of the service to be called,
   - *inputData* – input data to the service,
     returned value – standard output from the service (the service may also write down output files).
4. *int enqueueJob (String[] authenticator, int clid, String jobPath, int minCPU, int maxCPU, String resultPath, String email)*:
   - *authenticator* – returned previously by *logIn*,
   - *clid* – id of a cluster to queue the command on,
   - *jobPath* – path to the executable,
   - *minCPU* – minimum number of CPUs to use,
   - *maxCPU* – maximum number of CPUs to use,
   - *resultPath* – path to results,
   - *email* – used for sending information on the status of the job,
   - *returned value* – 0 – success, 1 – error.
5. *int uploadAttachments(String[] authenticator, int clid, DataHandler[] attachments, String[] remoteFileNames)* – for uploading large data files to be used as potential input to commands or services:
   - *authenticator* – returned previously by *logIn*,
   - *clid* – id of a cluster to upload data to,
   - *attachments* – data to be uploaded (files),
   - *remoteFileNames* – names of remote files that the data should be written to.

4. Typical Scenarios for Scientific Computations

Within the NIWA Centre of Competence project, BeesyCluster offers the following use cases for exposing high performance computing services to outside clients:

1. Publication and invocation of a service through WWW. As shown in Figure 4 user *niwauserpublisher* publishes a parallel application as a BeesyCluster service and subsequently makes it available to user *niwauserclient*. Figure 5 presents launching of the application performed by user *niwauserclient*. In this case, the application is run on the system account of user *niwauserpublisher* but in a safe sandbox. It should be noted that BeesyCluster features a module that allows publishing of applications contained in packages in UNIX systems as BeesyCluster services [21]. Descriptions of applications from the packages are published within descriptions of the services in BeesyCluster automatically.

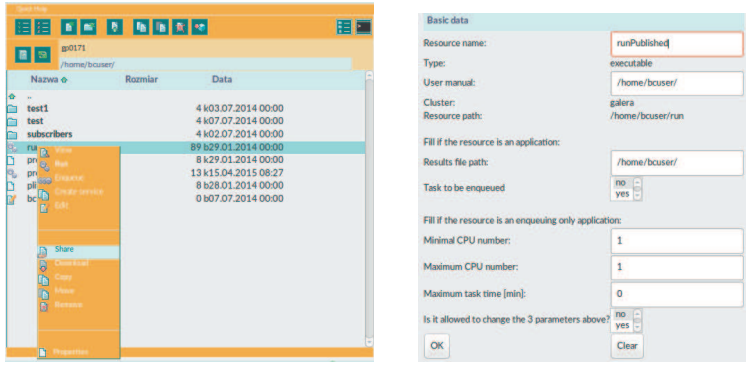

Figure 4. Sharing a service in BeesyCluster

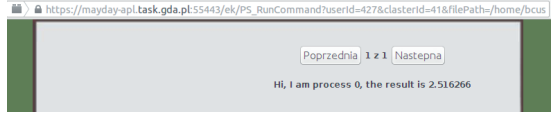

Figure 5. Invocation of a service published in BeesyCluster

2. Invocation of a service through SOAP Web Services. Any high performance computing application can be made available as a named service as shown in Figure 4 and subsequently accessed from an external application using the *runService* operation of a SOAP Web Service. Within the NIWA project, this approach is used for publishing a service linked to the KOALA library which provides many operations for the graph theory including graph definition and analysis. In this particular case, for KOALA, another proxy server has been established in front of the BeesyCluster server in order to publish an easy-to-use interface to domain users. Behind the proxy, either *runCommand* or *runService* are called.

It should also be noted that services deployed in BeesyCluster (either sequential or parallel applications) can be used in workflows outside of BeesyCluster. Such a workflow could be defined in BPEL and encompass other systems such as KASKADA [22] or Wiki-WS [23].

It can be seen that the proposed architecture is flexible as it defines services at many levels that can also be combined into workflow applications at various levels. Such architecture is presented in Figure 6.

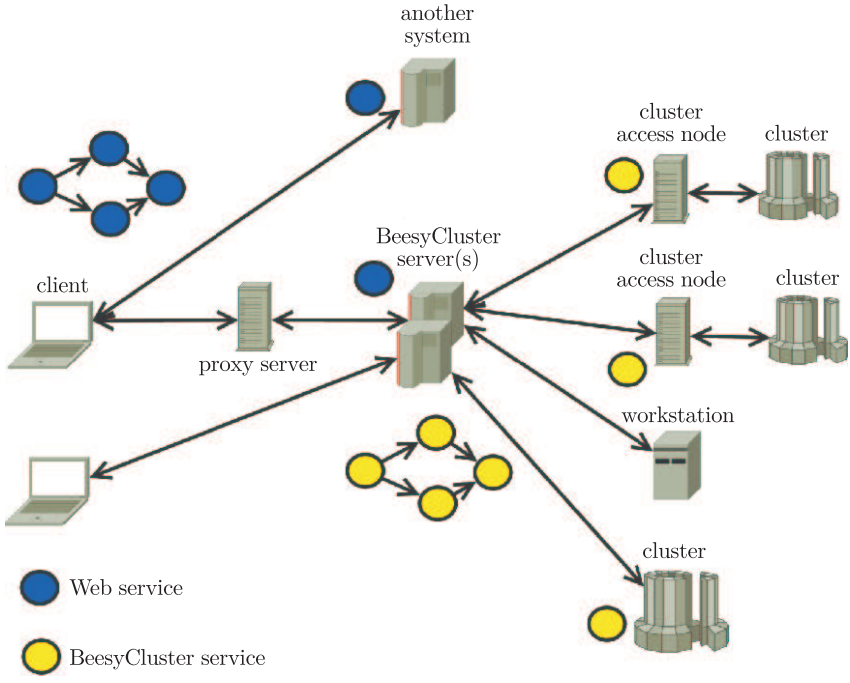

Figure 6. Architecture of solution with proxies

This flexibility comes at the price of increased communication latency. Table 1 lists execution times of various configurations (without and with a proxy, *i.e.* for the architecture shown in Figure 6) for listing contents of a directory using the WWW interface for a total of 360 bytes, measured within the FireFox browser. The request adds HTTP headers and the value of a cookie that is decoded by BeesyCluster in order to authenticate the user.

Table 1. Execution times for various configurations – WWW interface

Configuration	Execution time [ms]
Command line – same node	4 ms
Client – BeesyCluster server – cluster access node – cluster node	347 ms
Client – proxy server – BeesyCluster server – cluster access node – cluster node	369 ms

5. Conclusions and Future Work

The paper presented the BeesyCluster middleware as a front end for high performance computing services in the NIWA Centre of Competence project. Two interfaces: WWW and Web Services were presented that were extended for use within the project, especially for calling named services from BeesyCluster using SOAP Web Services as well as handling large sized data files. Future work based on the proposed solution could include coupling such services into higher level workflows that incorporate many systems from various domains as well as optimization of resource selection within BeesyCluster for running high performance computing applications with consideration of other factors such as power consumption.

Acknowledgements

This work was carried out as a part of the Centre of Competence for Novel Infrastructure of Workable Applicationsin "NIWA" project, Operational Program Innovative Economy 2007–2013, Priority 2 "Infrastructure area R&D".

References

[1] TOP500 Supercomputer Sites http://www.top500.org/

[2] Chapman B, Jost G and Pas R v. d. 2007 *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*, The MIT Press

[3] Sanders J and Kandrot E 2010 *CUDA by Example: An Introduction to General-Purpose GPU Programming*, Addison-Wesley Professional

[4] Stone J E, Gohara D and Shi G 2010 Opencl: A parallel programming standard for heterogeneous computing systems, *Computing in Science and Engineering* **12** 66

[5] Wienke S, Springer P, Terboven C and Mey D 2012 *Openacc: first experiences with real-world applications*, Proceedings of the 18th international conference on Parallel Processing; Euro-Par'12, Springer-Verlag 859

[6] Chrysos G 2012 *Intel® xeon phi™ coprocessor – the architecture; the first intel® many integrated core (intel® mic) architecture product*, Intel Corporation https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-codename-knights-corner

[7] Rahman R 2013 *Intel® xeon phi™ coprocessor vector microarchitecture*, Intel Software and Services Group https://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-vector-microarchitecture

[8] Fang J, Sips H, Zhang L, Xu C, Che Y and Varbanescu A L 2014 *Test-driving intel xeon phi*, In: Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering. ICPE'14, ACM 137

[9] Rosales C 2013 *Porting to the intel xeon phi: Opportunities and challenges*, In: Extreme Scaling Workshop (XSCALE13)

[10] Schmidl D, Cramer T, Wienke S, Terboven C and Müller M 2013 *Assessing the performance of openmp programs on the intel xeon phi*, *Springer Berlin Heidelberg*, In Wolf F, Mohr B, Mey D, eds.: Euro-Par 2013 Parallel Processing; Volume 8097 of Lecture Notes in Computer Science 547

[11] Forum M P I 2012 *MPI: A Message-Passing Interface Standard Version 3.0*

[12] Malawski M, Bubak M and Nabrzyski J 2011 *GridSpace scripting environment – from common component architecture to cloud components*, CCA'11: Cloud Computing and Its Applications

[13] Demuth B, Schuller B, Holl S, Daivandy J, Giesler A, Huber V and Sild S 2010 *The unicore rich client: Facilitating the automated execution of scientific workflows*, IEEE Sixth International Conference on e-Science 238

[14] Russell M, Dziubecki P, Grabowski P, Krysinski M, Kuczynski T, Szejnfeld D, Tarnawczyk D, Wolniewicz G and Nabrzyski J 2007 *The vine toolkit: A java framework for developing grid applications*, In Wyrzykowski R, Dongarra J, Karczewski K, Wasniewski J, eds.: PPAM; Volume 4967 of Lecture Notes in Computer Science, Springer 331

[15] Szejnfeld D, Dziubecki P, Kopta P, Krysinski M, Kuczynski T, Kurowski K, Ludwiczak B, Piontek T, Tarnawczyk D, Wolniewicz M, Domagalski P, Nabrzyski J and Witkowski K 2010 *Vine toolkit – towards portal based production solutions for scientific and engineering communities with grid-enabled resources support*, Scalable Computing: Practice and Experience **11**

[16] Czarnul P 2015 *Integration of Services into Workflow Applications*, Chapman & Hall/CRC Computer and Information Science Series, Taylor & Francis http://www.taylorandfrancis.com/books/details/9781498706469/

[17] Czarnul P 2013 Modeling, run-time optimization and execution of distributed workflow applications in the jee-based beesycluster environment, *The Journal of Supercomputing* **63** 46

[18] Czarnul P 2014 Teaching high performance computing using beesycluster and relevant usage statistics, *Procedia Computer Science* **29** 1458

[19] Czarnul P 2006 *Integration of compute-intensive tasks into scientific workflows in beesycluster*, In Alexandrov V, Albada G, Sloot P, Dongarra J, eds.: Computational Science – ICCS 2006; Volume 3993 of Lecture Notes in Computer Science, Springer 944

[20] Czarnul P, Bajor M, Frączak M, Banaszczyk A, Fiszer M and Ramczykowska K 2006 *Remote task submission and publishing in beesycluster: Security and efficiency of web service interface*, *Springer*, In Wyrzykowski R, Dongarra J, Meyer N, Wasniewski J, eds.: Parallel Processing and Applied Mathematics; Volume 3911 of Lecture Notes in Computer Science 220

[21] Czarnul P and Kurylowicz J 2010 *Automatic conversion of legacy applications into services in beesycluster*, 2nd International Conference on Information Technology (ICIT) 21

[22] Krawczyk H and Proficz J 2010 *KASKADA – multimedia processing platform architecture*, In Tsihrintzis GA., Virvou M, eds.: SIGMAP 2010 – Proceedings of the International Conference on Signal Processing and Multimedia Applications; SIGMAP is part of ICETE – The International Joint Conference on e-Business and Telecommunications, SciTePress 26

[23] Krawczyk H and Downar M 2012 *Commonly accessible web service platform-wiki-ws*, In: Intelligent Tools for Building a Scientific Information Platform, Springer 251