Grendel: A bioinformatics Web Service-based architecture for accessing HPC resources

Adam Hunter, David Schibeci, Hong Liang Hiew, Matthew Bellgard

Centre for Bioinformatics and Biological Computing (CBBC)
Murdoch University
South St, Murdoch, Western Australia 6150

a.hunter@cbbc.murdoch.edu.au

Abstract

Bioinformatics is an important application area for grid computing. Bioinformatics applications exported over a grid have the potential to facilitate transparent access to high performance computing (HPC) resources to a range of stakeholders. These stakeholders include application users, application developers, system administrators as well as service providers. In this paper we describe a Web Service based system called Grendel. Grendel acts as a single access point to HPC and is used to explore grid-enablement issues for bioinformatics tools. It uses a language and platform independent mechanism for remotely interacting with the HPC environment. In this paper, we provide details of Grendel's architecture as well as lessons learnt in developing and deploying the system.

Keywords: high performance computing; resource management; transparent access; bioinformatics

1 Introduction

Bioinformatics is an important application area for grid computing. In bioinformatics, the relevant issues related to resource sharing over a grid include processing power, large-scale data access and management, security, application integration, data integrity and curation, control/automation/tracking of workflows, data format consistency and resource discovery.

The requirements for bioinformatics tools to access high-performance computing (HPC) resources are different from tools in other areas of research. A significant number of bioinformatics tools exist as stable third party applications. These applications are largely static since they have already gone through extended deployment and testing phases and are in late maintenance stage of their development lifecycle. As such, users of the system do not require access to an application development environment to compile custom applications as one might encounter in other HPC problem domains. Under these circumstances, a service-oriented architecture (SOA) is a natural way to offer these bioinformatics applications. For a fuller description of SOA, see for example Zimmermann (2003) or Atkinson (2003).

There are presently some international efforts in building Web service based bioinformatics environments. Examples of these include BioMoby (Wilkinson 2002) and myGrid (Stevens 2003). However, it is still unclear how to best transparently provide high performance resources such as compute clusters to users. Most bioinformatics users are concerned with biology research, and it is unreasonable to expect them to interact with a typical HPC job submission environment.

We developed a system called Grendel, with the aims of providing Bioinformatics researchers transparent access to basic computational resources used in their research. Grendel is a platform and language independent Web Service based system for distributed resource management. Grendel provides a single entry point for computational tasks while keeping the actual resources transparent to the user. It also tracks the different stages of the tasks as they are executed, to allow proper monitoring and auditing.

Compared to providing users direct access to the bioinformatics applications or processing resources, the Grendel architecture results in lower system administrative overhead in terms of user management, a simple mechanism for adding additional programs and a single controlled access point to HPC resources. The architecture is also platform and language independent so as to be accessible via a scripting environment as well as via client applications across multiple platforms. It provides an abstraction layer between the HPC environment and the non-technical users of the system to allow them to leverage the resource effectively. We describe the Grendel architecture in the following sections.

2 System Architecture

Grendel was developed using a Web Service architecture. The Grendel Web Services were developed in Java using the Apache Axis SOAP library and deployed using the Apache Jakarta Tomcat 5 Servlet/JSP Container. The Grendel Web Service consists of a simple API that allows the client to submit jobs, query job status and retrieve job results.

In the background, the computational resources are managed by the Sun Grid Engine (SGE, http://gridengine.sunsource.net/). The SGE handles the steps involved in job execution and return results. Access to the SGE is provided through a wrapper, that acts as an exchange between the Web Service and the job execution environment.

For security purposes, the exchange between the client and the service is protected using the Secure Socket Layer (SSL). This ensures that information about the job as well as the raw data used is free from unauthorized access.

Figure 1 shows the overall architecture of the Grendel system.

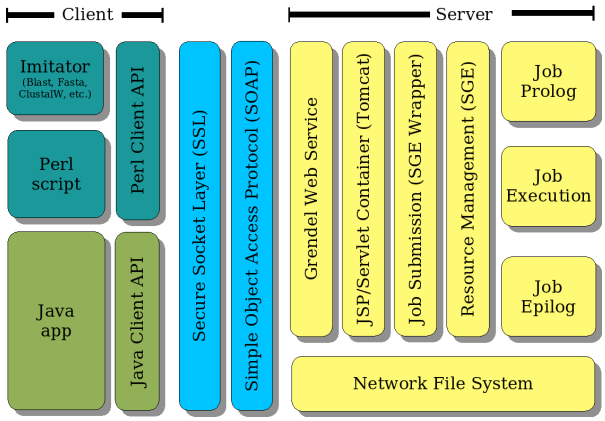

Figure 1: Grendel's overall architecture

The tools that are exported via Grendel are each defined by an XML document. For example, for the bioinformatics applications known as Blast we may have an XML document called 'blast.xml'. In this file all the parameters of Blast are defined along with other information about the program such as where it is located, which parameters are mandatory, parameter ordering and any other information required to invoke the application. During each stage of service request, job submission, job execution and return result, the same XML document is passed from one component to the next. Each component populates the XML document with details of the task. All components can use the information in this document for auditing of different kinds. Passing an XML document through the system in this manner provides a tracking and auditing mechanism that we refer to as a Bioinformatics Analysis Audit Trail (BAAT) subsystem. At this stage, BAAT documents are not validated against a set DTD or schema, since we have yet to expound the full use of the BAAT subsystem. The BAAT schema is an issue for future development.

Figure 2 illustrates an example BAAT XML document.

Complementing the Java Web Service are client APIs written in Java and Perl. The Java client is develoepd using Apache Axis SOAP library, while the Perl client is developed using the SOAP::Lite Perl module. The client APIs simplify the process of interacting with Grendel for client developers.

A Perl Imitator script called 'xmlGrendelCommandLine' was also written to imitate the function of tools available within Grendel. Multiple symbolic links are created to the Imitator script with the same names as tools available within Grendel. The Imitator uses the BAAT XML file as a template, and fills the parameter values according to the parameter attributes and the command line parameters given by the user. It uses the Perl client API to submit the job as described in the XML file as well as retrieve results when they are done.

```
<baat>
  <job toolName="blast" toolPath="/usr/local/grendel/bin/blastWrapper">
    <inputFile>36891100164648.zip</inputFile>
    <outputFile/>
    <parameterList>
      <parameter inputFile="no" mandatory="yes" rank="1" switch="-p"
switchUse="both" value="blastx"/>
      <parameter inputFile="no" mandatory="yes" rank="2" switch="-d"
switchUse="both" value="nr"/>
      <parameter inputFile="yes" mandatory="yes" rank="3" switch="-i"
switchUse="both" value="ordered_contig_ORFs_130.fas"/>
    </parameterList>
    <parameterAliasList/>
    <grendel id="11001649243680">
      <option>
        <name>dummy</name>
      </option>
    </grendel>
    <priority>-20</priority>
    <executionHost>node8</executionHost>
    <batchNumber>497873</batchNumber>
    <status>C</status>
    <startTime>2004/11/11 17:35:23</startTime>
    <stopTime>2004/11/11 17:47:30</stopTime>
    <submittedTime/>
    <submitUser>ymotro</submitUser>
    <submitLabel/>
  </job>
</baat>
```

Figure 2: Example BAAT file

Using an Imitator that pretends to be various bioinformatics applications in this manner has several advantages. Firstly, once new tools are installed in the compute environment, the Imitator allows new tools to be made available to clients via Grendel by simply creating an XML file and a symbolic link. Secondly, Imitated tools, are less strict on parameter ordering than the original tools. Lastly, the Imitator is a convenient place to add support a batch mode facility to allow users to submit multiple jobs at the same time.

3 Results and Discussions

The system as described above has been deployed successfully for a period of 18 months. It has successfully incorporated a total of 40 bioinformatics tools into the system. Figure 3 shows a subset of the tools currently supplied through Grendel. This system has been used by researchers in many research projects, with successful outcomes. For example, see papers by Appels (2004), Bellgard (2004) and Kulski (2003).

In developing and deploying the Grendel system, we have learnt some valuable lessons. The most important of this are that 1) we should keep to as simple an architecture as possible, and 2) we should follow defined standards in developing our system. We have found specific instances where follow the two approaches above have made our work much simpler. For example, the communication between the Java backend and the Perl imitator uses standard SOAP and should therefore not give much interoperability problems. However, the implementations of SOAP in Java and Perl do have certain incompatibilities. For this reason, keeping to simple operations and exchanges becomes very important. This will be true as we expand Grendel to including implementations using other programming languages as well.

```
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 bl2seq -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 blast -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 blastall -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 blastn -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 blastp -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 blastx -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 chips -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 clustalw -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 consense -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 extractseq -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 fastacmd -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 fastf -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 fasts -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 fastx -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 fasty -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 formatdb -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 genscan -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 getorf -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 glimmer -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 glimmerm -> xmlGrendelCommandLineß
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 mpiblast -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 multiblast -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 mview -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 needle -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 neighbour -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 prettyplot -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 repeatmasker -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 repeatmasker3 -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 sankoff -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 seqboot -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 signalp -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 ssearch -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tblastn -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tblastx -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tfasta -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tfastf -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tfasts -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tfastx -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tfasty -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 trnascan -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 tRNAscan-SE -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 webblast -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 wu-blast -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 wu-blastall -> xmlGrendelCommandLine
lrwxrwxrwx    grendel  grendel   21 Nov 12 13:16 xcompare -> xmlGrendelCommandLine
```

Figure 3: Example directory listing showing links to Imitator tool. Transparency to users is provided by creating a symbolic link with the same name as the original tool the users are used to.

We also find the philosophy of keeping things simple and following standards to have great advantages in other aspects of extending the system. It leads us to having an architecture that allows us to include new tools relatively easily by using the imitator concept. As different versions of software are released, we are less proned to major issues in deploying these new versions. Even to the extent of standards undergoing changes, we can accommodate the changes with a simple architecture.

Using a simple Web Services architecture is appropriate in our bioinformatics environment. Most of the advantages of easy coupling and de-coupling of operations naturally comes with a Web Service architecture. The advantages of developing grids using a service-oriented architecture have been reported elsewhere in the literature (eg. see Atkinson 2003).

However, as with all systems, there are certain aspects of the system that currently needs improvement. For example, the system is tied very intricately with the Sun Grid Engine, and we would like to refactor the implementation to de-couple the system from any particular process execution engine.

In this paper, we have described a system that deals with process-resource sharing, security, maintenance and control/automation/tracking of workflows. Future work in this system would involve extensions to include further grid-features such as data distribution and resource discovery.

At this stage, the system can interface with other systems through basic SOAP messaging, and SSL connection with mutually recognised certifying authorities. The interface with specific grid middlewares such as Globus and Nimrod are not fully explored yet. We anticipate this to be done soon.

The most obvious grid-middleware candidate for further tests and development is the soon-to-be-released Web Service compliant Globus4/WSRF (http://www-unix.globus.org/toolkit/docs/development/4.0-drafts/GT4Facts/index.html). We are currently awaiting the release of GT4 to continue in this direction. Until that release, we are wary of extending our interfaces that may move us away from a Web Service architecture, for reasons already outlined above. However, possible initial implementation problems in GT4 aside, we anticipate that WSRF (Web Service Resource Framework) will allow us to continue in the Web Service directions as we have described.

4 Conclusions

This paper describes as Web Service-based architecture to enable bioinformatics tools to access HPC resources. The Grendel system we described have achieved our aims of:

- Providing transparent access to HPC resources for bioinformatics applications and to users.
- Allowing easy coupling, decoupling and management of the bioinformatics tools with the HPC resources.
- Allow the tracking and auditing of the operations in this system.

We follow an approach of keeping the system as simple as possible, and adopting standards as much as we can to cater for unpredictabilities in the environment we work in. We have found this approach to be critical in our development and deployment of the system. We will keep to this approach in further extensions to this system.

5 References

Atkinson M., DeRoure D., Dunlop A., Fox G., Henderson P., Hey T., Paton N., Newhouse S., Parastatidis S., Trefethen A. and Watson P. (2004), Web Service Grids: An Evolutionary Approach, Report UKeS-2004-05, UK e-Science Technical Report Series, available from http://www.nesc.ac.uk/technical_papers/uk.html.

Appels R, Francki M, Cakir M, Bellgard M. (2004), Looking through genomics: concepts and technologies for plant and animal genomics. Funct Integr Genomics. 4(2):71-3.

Bellgard M, Ye J, Gojobori T, Appels R. (2004), The bioinformatics challenges in comparative analysis of cereal genomes-an overview. Funct Integr Genomics. 4(1):1-11.

Kulski JK, Lim CP, Dunn DS, Bellgard M. (2003), Genomic and phylogenetic analysis of the S100A7 (Psoriasin) gene duplications within the region of the S100 gene cluster on human chromosome 1q21. J Mol Evol. 56(4):397-406.

Stevens RD, Robinson AJ, Goble CA. (2003), myGrid: personalised bioinformatics on the information grid. Bioinformatics. 19 Suppl 1:i302-4.

Wilkinson MD, Links M. (2002), BioMOBY: an open source biological web services proposal. Brief Bioinform. 3(4):331-41.

Zimmermann O, Tomlinson MR and Peuser S. (2003), Perspectives on Web Services: Applying SOAP, WSDL, and UDDI to Real-World Projects, Springer-Verlag.