Distributed BLAST in a Grid Computing Context

Micha Bayer and Richard Sinnott

National e-Science Centre, e-Science Hub, Kelvin Building, University of Glasgow, Glasgow G12 8QQ
{michab,ros}@dcs.gla.ac.uk

Abstract. BLAST is one of the best known sequence comparison programs available in bioinformatics. It is used to compare query sequences to a set of target sequences, with the intention of finding similar sequences in the target set. Here, we present a distributed BLAST service which operates over a set of heterogeneous grid resources and is made available through a Globus toolkit v.3 grid service. This work has been carried out in the context of the BRIDGES project, a UK e-Science project aimed at providing a grid based environment for biomedical research. Input consisting of multiple query sequences is partitioned into sub-jobs on the basis of the number of idle compute nodes available and then processed on these in batches. To achieve this, we have implemented our own Java-based scheduler which distributes sub-jobs across an array of resources.

1 The BRIDGES Project

The BRIDGES project (Biomedical Research Informatics Delivered by Grid Enabled Services [1]) is a core project of the UK's e-Science Programme [28] and is aimed at developing grid-enabled bioinformatics tools to support biomedical research. Its primary source of use cases is the Cardiovascular Functional Genomics Project (CFG), a large collaborative study into the genetics of hypertension (high blood pressure). Hypertension is partly genetically determined, and investigation of the genes and biological mechanisms responsible for high blood pressure is of great importance.

BRIDGES aims to aid and accelerate such research by applying grid-based technology. This includes data integration tools but also grid support for compute intensive bioinformatics applications such as BLAST. The service described in this paper is based primarily on the use case of microarray chip annotation, in which microarray reporter sequences have to be compared against annotated sequence data (e.g. from the human genome). These are highly compute-intensive tasks, involving several hundred thousand input sequences and very large target databases, and may take on the order of several weeks to compute on a single processor machine. An additional use case are BLAST runs of small batches of several tens of sequences against standard databases such as the NCBI nt database [5].

2 Parallelising BLAST

BLAST - the Basic Local Alignment Search Tool [3] is a widely used search algorithm that is used to compute alignments of nucleic acid or protein sequences with the goal of finding the *n* closest matches in a target data set. BLAST takes a heuristic (rule-of-thumb) approach to a computationally highly intensive problem and is one of the fastest sequence comparison algorithms available, yet it still requires significant computational resources. It does therefore benefit greatly from being run in a grid computing context providing it is parallelised, i.e. single jobs must be partitioned into independent sub-jobs that can be run on remote resources concurrently. In the case of BLAST (and possibly similar sequence comparison programs) parallelisation can be achieved at three different levels [6, 7]:

1. A single query can be compared against a single target sequence using several threads in parallel, since there are O(*nm*) possible alignments for a query sequence of length *n* and a target sequence of length *m*. This approach is implemented by default by the BLAST executable itself.
2. Input files with multiple query sequences can be parsed to provide individual query sequences or blocks of sequences, and these can then all be compared against multiple identical instances of the target data file in parallel.
3. With input files containing only a single query sequence, the target data can be segmented into *n* copies for *n* available compute nodes, and then multiple identical instances of the query sequence can each be compared to a different piece of the target data in parallel.

There are several existing implementations which take approach 2 [7, 8] or 3 [9, 10, 11] but none of these suited the particular requirements in this project. One of the better known implementations of approach 3 is mpiBLAST [10], an MPI-based implementation of BLAST in which the target database is segmented into a number of fragments that the input is then compared against concurrently. It was initially considered to include mpiBLAST on our back end resources so that single query input can benefit from parallelisation but after preliminary tests it was excluded from the design for the following reasons:

- The speedup described by the authors of mpiBLAST [10] appears to be closely linked to the particular set of conditions their tests were run under and could not be reproduced under the conditions on our clusters. Instead, execution times actually increased when more than 10 processors were used in the trial runs.
- mpiBLAST requires the target database to be segmented and the fragments to be formatted ahead of the actual BLAST run. This is a compute-intensive task which takes approximately 30 minutes for the NCBI's *nt* database [5], a standard nucleotide database widely used for BLAST searches. This is a significant overhead which is unsuitable for single, short jobs that may take only minutes to compute. The alternative of having a predetermined number of preformatted database fragments ready for computation on the cluster is unworkable because of the requirements of mpiBLAST itself. The software requires *n* + 2 processors to be available if *n* database fragments are to be used, and in practice this means that jobs will have to be queued until *n* processors are available, leading to very significant delays which usually outweigh any potential performance gains.

The final design therefore only included parallelization of BLAST at the level of the input data.

3 BRIDGES GT3 BLAST service

There is a growing number of grid based implementations of BLAST [12, 13, 14, 15], based on various grid middleware but we here present the first all-Java, OGSA[22]-based implementation, based on version 3 of the Globus toolkit. Globus v.3 is based on the web services programming model but services have been extended to operate in a grid computing context, with service data, client notification mechanisms and statefulness as added functionality. Our GT3 based implementation of BLAST is used in conjunction with our own metascheduler that allows jobs to be farmed out to remote clusters and the results to be combined.

3.1 Basic design

A GT3 core based grid service is used as a thin middleware layer on top of our application (Figure 1). The service itself has deliberately been kept basic to allow easy porting to other platforms since grid computing is still very much in flux. It therefore only uses limited GT3 functionality, with no client notification or service data having been implemented. The only significant GT3-specific functionality used is the service factory pattern. This is a useful feature because it provides an easy way of handling concurrency in the context of this application, with a service instance representing a single job submitted by a single user. The actual BLAST computation is carried out by NCBI BLAST which is freely available from NCBI [5], and BLAST executables are preinstalled on all compute resources since these are relatively unchanging components and therefore regular stage-in of the executable at runtime would be a waste of resources.

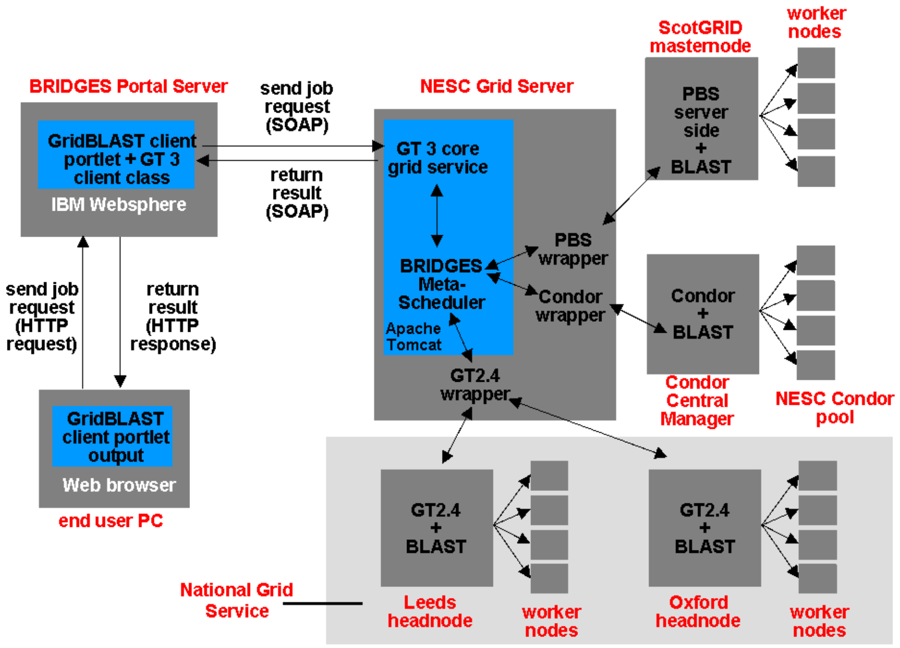


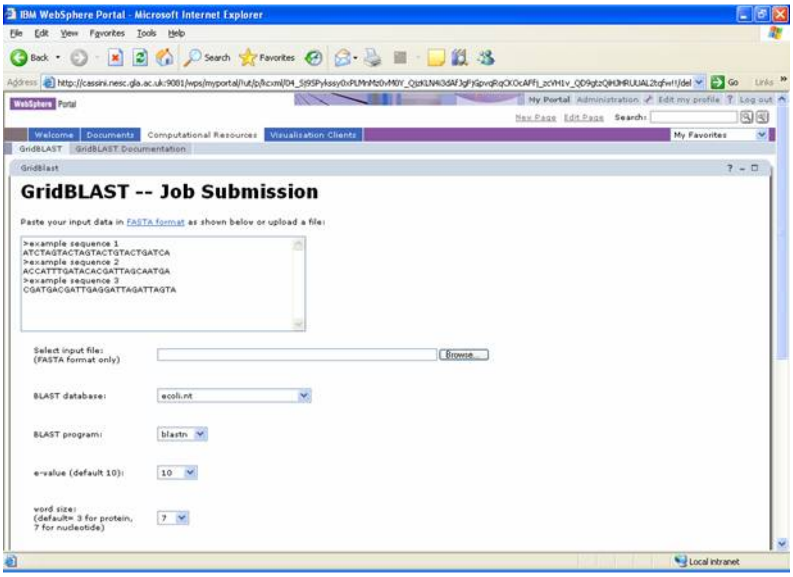
Fig. 1. Schematic of system architecture

3.2 Deployment and configuration

The grid service is deployed using Apache Tomcat running on a Linux server. To allow easy modification of the set of compute resources available to the service, resource details are held in an XML configuration file which is read by the service at runtime and an array of resources is initialised accordingly. The details of interest include the type of batch submission system, the domain name of the resource's head node, the number of compute nodes, memory etc. There is currently no provision of dynamic resource discovery.

3.3 User interface to service

A web portal has been designed for the BRIDGES project to allow users easy web based access to resources and in order to avoid having to install grid software on end user machines. This has been implemented using IBM Websphere, currently one of the more sophisticated portal packages. Users log in with standard username and password pairs and are then presented with a job submission portlet.


IBM WebSphere Portal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address Go Links
WebSphere Portal
My Portal Administration Edit my profile Log out
New Page Edit Page Search:
Welcome Documents Computational Resources Visualisation Clients
My Favorites
GridBLAST GridBLAST Documentation
GridBlast
Browse...
Local intranet

GridBLAST -- Job Submission

Paste your input data in FASTA format as shown below or upload a file:

```
>example sequence 1
ATCTAGTACTAGTACTGTACTGATCA
>example sequence 2
ACCATTTGATACACGATTAGCAATGA
>example sequence 3
CGATGACGATTGAGGATTAGATTAGTA
```

Select input file: (FASTA format only)

BLAST database: ecoli.nt

BLAST program: blastn

e-value (default 10): 10

word size: (default= 3 for protein, 7 for nucleotide) 7

Fig. 2. BRIDGES Web portal showing the job submission portlet for the distributed BLAST service

3.4 Scheduler and input data segmentation

For the purpose of this project we decided to provide our own scheduler since at the time of designing the application grid meta-schedulers were rare and none were available that satisfied our requirements with respect to the particular combination of OS and batch submission system on our resources (*n.b.* "resource" here denotes a computational back end such as a compute cluster).

Our scheduler can take an unlimited number of resources as an argument and uses the following algorithm to distribute subjobs across these:

```
parse input and count no. of query sequences

poll resources and establish total no. of idle nodes
set no. of sub-jobs to be run to no. of idle nodes

calculate no. of sequences to be run per sub-job n (=
no. of idle nodes/no. of sequences)

while there are sequences left

          save n sequences to a sub-job input file

if the number of idle nodes on the whole grid is 0
```

```
        make up small, predetermined number of sub-jobs and
        evenly distribute these into queues across re-
        sources

    else

        send $i$ subjobs to the resource, where $i$ is the num-
        ber of idle nodes at a resource

    when all subjob results are complete, combine them into
    single result file using the original input sequence
    order

    return combined output file to the user
```

Thus, the system will always make use of the maximum number of idle nodes across resources if multiple query sequences exist. Load balancing is achieved by assigning only as many sub-jobs to a resource as there are idle nodes, and by making all input files roughly the same size.

3.5 Compute resources and wrapper classes

Grid computing environments usually feature a heterogeneous mixture of back end resources [23] and their associated operating systems, and the resources available to this project are typical in this respect. The requirement arising from this is that the grid metascheduler must be capable of submitting jobs to the resources via a number of wrapper components that feature shared functionality. The minimum functionality that a basic job submission system requires consists of 3 elements: job submission, job monitoring and job cancellation.

Our design satisfied these requirements through the use of an abstract *Resource* class which was then extended with wrappers that provided the above functionality for each type of back end batch system. We implemented Java wrappers for the Condor [19] and PBS [18] batch systems as well as for Globus 2 server side installations [24]. The Resource class contains abstract methods for job submission, job monitoring and cancellation, with the wrappers providing concrete implementations of these for the specific back end systems.

In the case of the PBS and Condor batch systems we used the respective client packages provided with PBS and Condor and wrapped the relevant commands there as native processes in Java, using the java.lang.Runtime and java.lang.Process classes. In the case of the wrapper for Globus 2, we used the Java Commodity Grid Kit version 1.1 [25], which provides a convenient and easy-to-use high-level API for job submission to GT2 resources.

The actual resources available to the project included:

- Our local Condor pool at the National e-Science Centre Glasgow, a small cluster of 21 single processor desktop machines
- ScotGRID [17], a 250 processor compute cluster located at Glasgow University

- The recently formed National Grid Service (NGS) of the UK [26], a grid consisting of currently 6 compute clusters distributed over the UK, intended to support the needs of the scientific computing community in the UK

3.6 Access to resources

A typical feature of compute grids are complex sociological and economic issues revolving around the use and access of grid resources. In particular, resource access and usage are potentially sensitive issues which require careful consideration and a set of tools designed to implement those policies.

Policies and authentication

In this project's resource set, access policies differed significantly between resources. The Condor pool owned by NeSC Glasgow is effectively open to any user and therefore does not require any authentication mechanism, but there are restrictions on the kind of jobs that can be executed there which are enforced through providing application-specific rather than generic user interfaces only (see section on user front ends above). ScotGRID requires user registration through the support staff, and users then log on to the service using their *ssh* credentials. The UK National Grid Service allows any person in academia access after having registered with a local representative of the certification authority. The identity of the user is backed up by a digital certificate and private key issued by the certification authority, and access to the resources is permitted through a user proxy that has been created from the user's credentials.

One of the project requirements was that user authentication should not cause any additional learning or usability overheads for the users. Biology end users range widely in computer literacy and therefore systems providing a single mechanism for users of all abilities should aim at the lowest level of literacy. The process of obtaining and caring for digital certificates for the National Grid Service is currently still rather involved and requires familiarity with command line tools and a generally advanced level of computer literacy. It was therefore decided to remove digital certificates from the end user environment altogether and replace them with simple username and password authentication at a central project web portal (see below). Authentication at NGS resources is instead being carried out by means of a host proxy generated from the grid server's host credentials (Fig. 3). The host's identity is then mapped locally to a project account in the local gridmap file. Thus, all jobs run under the project's identity on the NGS resources, and the logging and monitoring of user activity has to be moved up one level into the domain of the BRIDGES support staff.

Authorisation

Once a user is logged on, they have access to the complete set of tools available on the project portal. The finer grain control of what back end resources associated with a tool are accessible for a given user is implemented through the grid authorisation software PERMIS [27]. The identity of the user submitting the job can be extracted from the portal context, and is passed on with the job request (Fig. 3). The grid server sends a lookup request to a dedicated PERMIS authorisation server maintained by the project team, where secure attribute certificates are used to store information about the roles a user is authorised for. This way of using PERMIS is different from the standard method but allows us to obtain user authorisations from any code base, and does not rely on the query coming from a GT3.3 service as specified by the current PERMIS model. There, the PERMIS service is queried directly by a GT3.3 service which contains the actual methods to be authorised, and which is tied into PERMIS through its deployment descriptor. Here, we are using the PERMIS service as a loosely coupled lookup service instead. In order to allow this, a fictitious service name needs to created and a number of fictitious methods attached to it that PERMIS can then query. The code can then iterate over the set of fictitious methods to establish what a user is allowed to do.

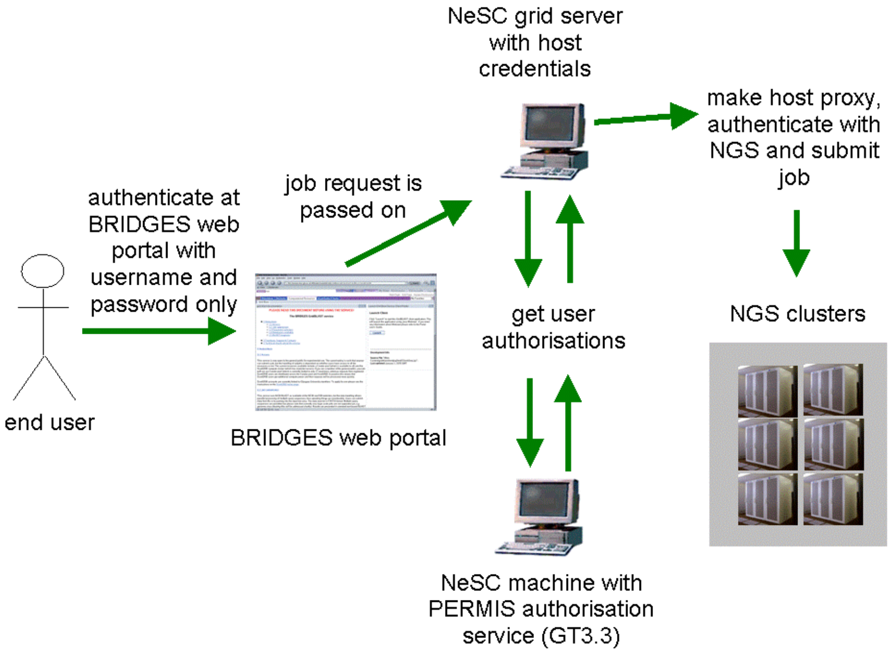

NeSC grid server with host credentials
make host proxy, authenticate with NGS and submit job
authenticate at BRIDGES web portal with username and password only
job request is passed on
get user authorisations
NGS clusters
end user
BRIDGES web portal
NeSC machine with PERMIS authorisation service (GT3.3)

Fig. 3. BRIDGES security infrastructure for job submission onto the NGS nodes

3.7 Target databases

Grid based file transfer is potentially time-consuming and the favourable option is to keep frequently used data cached locally, close to the computation. In the case of BLAST this is readily achievable since most users blast against a small number of publicly available target databases (available as flat text files for ftp download, e.g. from NCBI [5] or EBI [4]). Typically, the databases are in the order of gigabytes, with some of them growing exponentially. The data can be stored on the local NFS of the compute resource and updated regularly through automated scripts.

3.8 Future work

Grid computing is in its early stages and the middleware toolkits available to developers are still undergoing major architectural changes. Since the beginning of this project, the Globus Alliance has released several minor versions of GT3 and now a new major version, GT4 [29]. The current service should now be ported to GT4 to avoid problems regarding support etc. GT4 is a move towards greater homogeneity in the field of web/grid services and has the support of several major industry partners.

References

1. BRIDGES project. http://www.brc.dcs.gla.ac.uk/projects/bridges/
2. Sinnott R, Atkinson M, Bayer M, Berry D, Dominiczak A, Ferrier M, Gilbert D, Hanlon N, Houghton D, Hunt E & White D. 2004. Grid Services Supporting the Usage of Secure Federated, Distributed Biomedical Data. Proceedings of the UK e-Science All Hands Meeting, Nottingham, UK, August 2004.
3. Altschul SF, Gish W, Miller W, Myers EW & Lipman DJ. 1990. Basic Local Alignment Search Tool. J. Mol. Biol. 215: 403-410.
4. EBI BLAST. http://www.ebi.ac.uk/blastall/index.html
5. NCBI BLAST website: http://www.ncbi.nlm.nih.gov/BLAST/
6. Pedretti KT, Casavant TL, Braun RC, Scheetz TE, Birkett CL, Roberts CA. 1999. Three Complementary Approaches to Parallelization of Local BLAST Service on Workstation Clusters (invited paper). Proceedings of the 5th International Conference on Parallel Computing Technologies, p.271-282, September 06-10.
7. Braun RC, Pedretti KT, Casavant TL, Scheetz TE, Birkett CL & Roberts CA. 2001. Parallelization of local BLAST service on workstation clusters. Future Generation Computer Systems 17: 745-754.
8. Clifford, R & Mackey AJ. 2000. Disperse: a simple and efficient approach to parallel database searching. Bioinformatics 16: 564-565.
9. Mathog DR. 2003. Parallel BLAST on split databases. Bioinformatics 19: 1865-1866.
10. Darling AE, Carey L & Feng W. 2003 The design, implementation and evaluation of mpiBLAST. Proceedings of ClusterWorld Conference & Expo and the 4th International Conference on Linux Clusters: The HPC Revolution.
11. Hokamp K, Shields DC, Wolfe KH & Caffrey DR. 2003. Wrapping up BLAST and other applications for use on Unix clusters. Bioinformatics 19: 441-442.
12. GridBlast at Keck BioCenter, University of Wisconsin: http://bioinf.ncsa.uiuc.edu/
13. GridBlast at A-Star Bioinformatics Institute Singapore: http://www.bii.astar.edu.sg/infoscience/dcg/gridblast/index.asp
14. North Carolina BioGrid. http://www.ncbiogrid.org/tech/apps.html
15. RIKEN GridBlast implementation. http://big.gsc.riken.jp/big/Members/fumikazu/Activity_Item.2004-02-02.0425
16. mpiBLAST. http://mpiblast.lanl.gov/
17. ScotGRID. http://www.scotgrid.ac.uk/
18. OpenPBS. http://www.openpbs.org/

19. Condor. http://www.cs.wisc.edu/condor/
20. Globus WSRF. http://www.globus.org/wsrf/default.asp
21.GenBank statistics. http://www.ncbi.nih.gov/Genbank/genbankstats.html
22.OGSA – Open Grid Services Architecture. http://www.gridforum.org/documents/GWD-I-E/GFD-I.030.pdf
23. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. I. Foster, C. Kesselman, S. Tuecke. International J. Supercomputer Applications, 15(3), 2001.
24. Globus toolkit version 2.4.3. http://www-fp.globus.org/gt2.4/
25. Java Cog kit version 1.1. http://www-unix.globus.org/cog/java/1.1/
26.UK National Grid Service. http://www.ngs.ac.uk/
27.PERMIS grid authorisation software. http://www.permis.org
28.The UK e-Science Programme. http://www.rcuk.ac.uk/escience/
29.Globus toolkit. http://www.globus.org/toolkit/