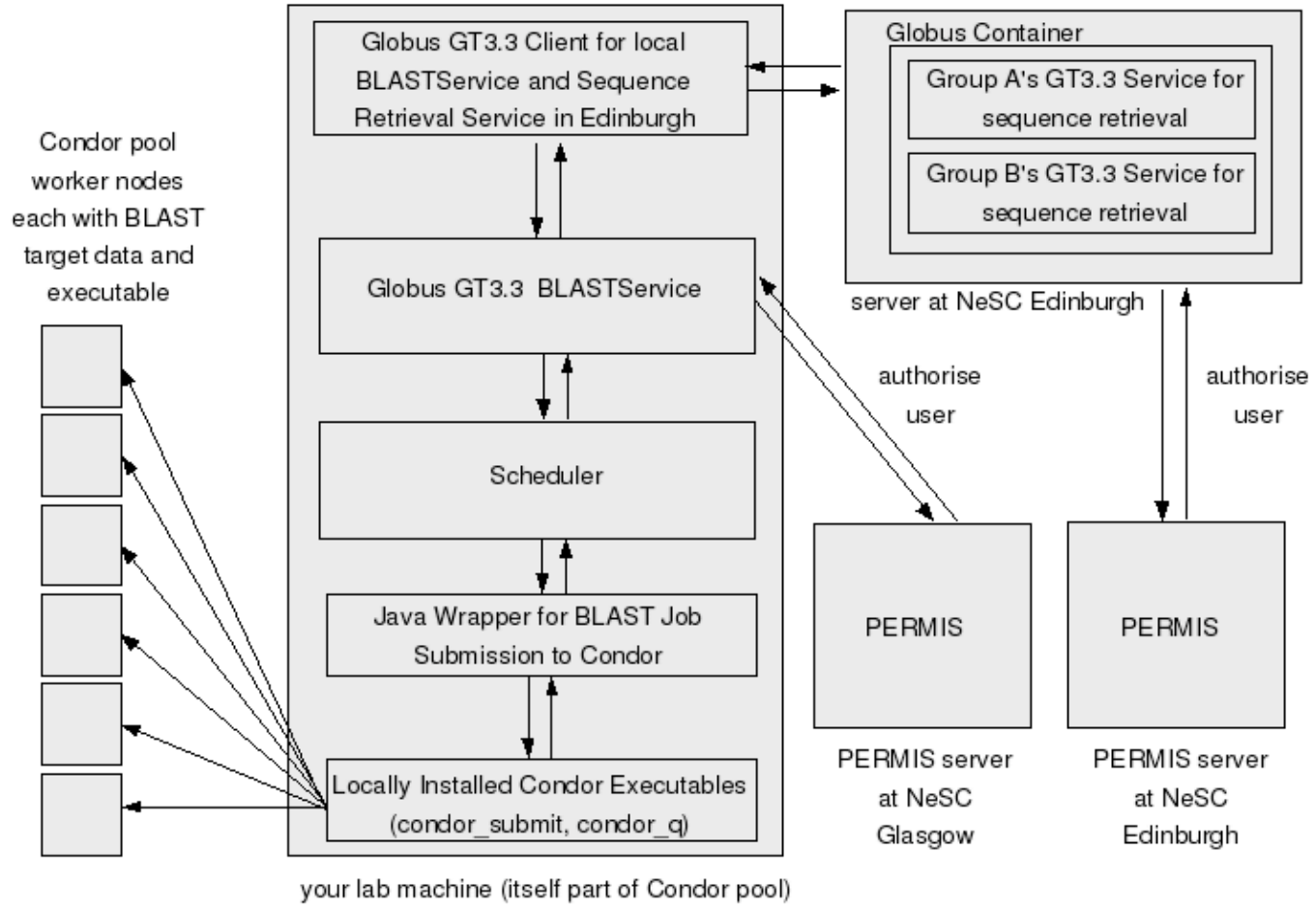
Grid Computing (M):
Programming Assignment 2006

Background

Bioinformatics is rapidly becoming a major application area of Grid computing due to the exponentially growing amounts of biological data available and the associated need for processing power. In this exercise you will be using Grid technology to parallelise and make accessible a real-world bioinformatics application: Basic Local Alignment Search Tool (BLAST) (see http://www.ncbi.nlm.nih.gov/BLAST/). BLAST is a heuristic matching algorithm used to compare one or more biological sequences against a set of target sequences, with the objective of finding the closest matches in the target sequence set. This is used for example in the discovery of genes causing diseases.

The figure shown here describes the system architecture and the components needed for this to function. Some of these are provided, and some you will have to implement. Specifically, you will be building this on top of an existing infrastructure at NeSC (Glasgow and Edinburgh) where we have a Condor pool with BLAST data and executables set up for this purpose, Globus preinstalled on your workstations, and Grid data services implemented at Edinburgh for you to use.



Instructions

The following is a recipe for what you are expected to implement. We strongly suggest that you follow the plan described here. We also strongly recommend that you refer to the Globus (GT3) tutorial that is available at http://gdp.globus.org/gt3-tutorial/singlehtml/progtutorial_0.4.3.html. An appendix to this handout for instructions on using PERMIS has also been provided for your benefit.

In this assignment you have been split into two groups:

- Group A (nucleotide team):
  - John Findlay,
  - Clemens Rodegast,
  - Konstantinos Topoglidis,
  - Andrew Ong,
  - Jonathan Matthey

- Group B (protein team):
  - Christopher Miller,
  - Ioannis Psarras,
  - David Rankin,
  - Gordon Stewart,
  - Craig Wilson

These groupings are to be used to establish and enforce authorisation aspects associated with the Grid services you will implement. The Distinguished Names (DNs) associated with your Grid certificates which you have been provided with are of the form:

```
DN = /c=gb/o=glasgow/ou=dcs/cn=student name
```

Implementation Steps Needed

1. Grid Service Client
Write a GT3.3 client class that calls a service located at NeSC in Edinburgh and returns a String with biological sequences to be used as input for the components described below. For this purpose you have been allocated to one of two groups of students, group A or group B as noted above. You should call the method on the service appropriate to your group. The URL of the Grid service to call is:

```
http://129.215.30.155:8080/ogsa/services/nesc/BlastData
```

The Java signatures of the methods are:

```
GROUP A: public String getNucleotideData();
GROUP B: public String getProteinData();
```

The method for group A will return gene sequences (nucleotides), the method for group B will return protein sequences. The methods are access controlled with respect to your identity, and only students from group A can access method A and ditto for method B/group B. The purpose of this exercise is to demonstrate PERMIS authorisation applied to a grid service, although in this instance the server side PERMIS infrastructure has been provided already. You will need to use client side GSI security, and have a valid proxy on your local system when you run the client. The group A sequence data will be of the form ATCCTGTGTCTGA and the group B sequence data will be of the form QHGHGSPJKHFGGRTSD (see http://www.ncbi.nlm.nih.gov/blast/html/search.html).

2. Running Blast
We have preinstalled target data and BLAST executables on all machines in the Condor Pool. Your local machine is part of the Condor pool, so you can test BLAST locally to get a feel for how it works. You should be able to run the following command locally on your machine (if you can't, please contact one of the NeSC team members):

```
/data/blastall -p blastn -i /data/testdata/test.txt -o test.out -d /data/ecoli.nt
```

This should produce an output file called test.out in your current directory, which is exactly 10942 bytes in size. The executable called is /data/blastall, the -p flag is needed to indicate the flavour of BLAST program you need for this (blastn for nucleotide data, blastp for protein data), -i specifies the full path of the input file, -o the output file and -d the full path to the target data file.

BLAST works with both gene (nucleotide) sequences and protein (amino acid) sequences. Since you have been divided into groups for this, and each group will be returning different data from the service in Edinburgh, you need to adapt the BLAST command accordingly. For nucleotide sequences (group A) the command will look like the above example (you will need to modify path and file names for the input and

out files accordingly but you have to use the target data used there, i.e. /data/ecoli.nt). For protein sequences (group B), the syntax will look like this:

```
/data/blastall -p blastp -i <input_file> -o <output_file> -d /data/NC_003304.faa
```

where again you need to use the target data specified here (/data/NC_003304.faa).

3. Running a BLAST job through Condor
Write a Condor execution script for the above BLAST nucleotide job (first example in section 2) and submit this using the condor_submit command. This is not a deliverable but will help you see what you have to do next. You will have to stage in your input file (use /data/testdata/test.txt and rename it). Condor documentation is available at http://www.cs.wisc.edu/condor/. The target data you used here are the ones you should also be using for your actual BLAST job implementation (/data/ecoli.nt).

4. Build the Java Condor Wrapper
Wrap the Condor BLAST job submission in a Java class which writes the submit script to file and submits that by calling the locally installed condor_submit executable as a native process.

- Tip 1: There are utility classes in the java.lang package that allow you to do this.
- Tip 2: To find out when a job has finished you can monitor the job's status with the condor_q command or monitor the local output file on your machine – this gets produced by Condor instantly at the beginning of the job but doesn't get populated with data until the job has finished.

5. Build the Scheduler
Write a scheduler class that splits your overall input String (obtained from the grid service in Edinburgh) into a number of parts. You will need to keep track of the ordering of the subsequences you have created. These subsequences should be submitted to the Condor pool through the Java wrapper class as several independent jobs. You will probably have to use separate threads here. For group A the input file format for BLAST looks like:

```
>seq1
AGCTTTTCATTCTGACTGCAACGG
>seq2
TTCTGAACTGGTTACCT
```

i.e. a text file with sequence names on a separate line, preceded by ">", and then the sequence on another separate line.

This component should return the combined result from all the subjobs, with individual results being concatenated into a single String in the correct order, i.e. the order of the results should mirror the original order of the input sequences in the overall input String. One should be able to specify how many subjobs the input gets split into but your code should limit this to a maximum of 20.

6. Build the BLAST Service
Wrap the scheduler with a GT3.3 service deployed locally on your allocated lab machine so that a call to the service will run the parallelised BLAST on Condor and return the combined result. The service must be called "***BLASTService***" and must contain ***one method*** with one of the following method signatures as appropriate to your group membership:

Group A: `public String runBLASTN(String inputData, int numberOfSubjobs);`
Group B: `public String runBLASTP(String inputData, int numberOfSubjobs);`

where `runBLASTN` will run a nucleotide (gene sequence) BLAST job with the nucleotide sequences retrieved from the Edinburgh server as the input, and `runBLASTP` runs a protein BLAST job with the protein sequences retrieved from Edinburgh (using the appropriate BLAST programs blastn/blastp as discussed above in section 2).

Your service needs to be access protected by PERMIS so that only members of your own team can call the method you implemented. Instructions for how to do this are provided in the appendix.

Deploy this service into a Globus container on the lab machine you have been allocated.

7. Build Additional Functionality to the Grid Service Client to Call the BLAST Service
Write some code that calls your BLAST service and retrieves the overall result. It should take as a command line argument the number of subjobs that you want your input to be split into. Include this code in your Grid Service Client component (section 1). This code will also need to use client side GSI security so that PERMIS can extract user credentials and make decisions about which methods can be invoked (see appendix).

Assessment

- It is assumed that you will build these components in the order they are numbered in the instructions. This will allow you to choose the amount of time you invest into the exercise, by breaking it down into components.
- Each of the components will earn you a certain amount of marks:

Software Description	***Marks (total =85)***
Data Grid Service Client	10
Java Condor Wrapper	15
Scheduler	25
BLAST Service	25
Client code for BLAST service	10

- In addition to the possible 85 marks available for implementing the software components, a further possible 15 marks will be available for the system design and discussion of the issues in using the Grid technologies.

- You must submit a full report including your designs, a printout of all your source code and a summary of how far you have got (i.e. which components you have built) and what difficulties, if any, you encountered. This report must also include instructions on how to test your components (see note below). These reports should be stapled together, along with a "pink plagiarism sheet" (available from the Teaching Office) and submitted in the appropriately labelled locked box on the ground floor of house 16 in Lilybank Gardens.

- **IMPORTANT: Your report must include clear instructions on how to run the various components for assessment (including names of classes and full paths to compiled executables on your machine). You will be expected to demonstrate the execution of your solutions at a date to be determined.**

- **Submission Deadline: Friday 10th March 2006 17:00 hrs**

Appendix: Guide to Using PERMIS in this Assignment

This guide is designed to help with setting up a grid service secured by PERMIS. The instructions detail points of information that would not be otherwise available but they are not comprehensive. Errors and exceptions can and will occur - it is up to you to use your initiative in solving these problems.

Setting up the PERMIS environment

The following configuration must be done before PERMIS can be used to secure grid services. These changes must be made to the Globus 3.3 installation on your machine.

1. Go to the PERMIS website (http://sec.isi.salford.ac.uk/permis/) and go to the "GT3 Cookbook". Scroll down the page and download the files "permisAuthz.gar" and “authz-sample.gar”.
2. Deploy this service into the Globus container, which should result in three versions of the PermisAuthorizationService being rendered in the container: decider, public and secured.
3. Create your own grid service, deploy it and run it unsecured to check that it works as it should
4. Add the following parameters to server-config.wsdd under the tag **service="your service-name"**:
    - **<parameter name="authorization" value="custom"/>**
    - **<parameter name="authzService" value="http://localhost:8080/ogsa/services/decider/PermisAuthorizationService"/>**
    - **<parameter name="authzClass" value="org.globus.ogsa.impl.security.authorization.SAMLAuthorizationCallout"/>**
5. Add the following parameters to server-config.wsdd under the tag **service="decider/permisAuthorizationService"**:
    - **<parameter name="OID" value="1.0.0.2"/>**
    - **<parameter name="SOA" value="cn=soa, o=glasgow, c=gb"/>**
    - **<parameter name="LDAP" value="ldap://titania.nesc.gla.ac.uk:389"/>**
    - **<parameter name="LDAP_AC_attribute" value="attributeCertificateAttribute;binary"/>**
    - NOTE: This last one may or may not be needed depending on whether the bug has been fixed that made its requirement necessary. Play around - remove it, or remove the ";binary" part - go with whatever works
    - Remove any existing parameters that refer to "authorization", "authzClass" or "authzService"
    - Remove any parameters referring to **ROOT_CA**
6. At the top of server-config.wsdd there should be three parameter references that involve GRIM authentication. Search for the word "grim" and comment out any lines that have this (there should only be three).
7. Unset the following environment variables: **CP** and **X509_USER_PROXY**
8. Set the **LD_LIBRARY_PATH** variable to **$GLOBUS_LOCATION/lib**
9. Get the files **fcec9e22.0** and **fcec9e22.signing_policy** and place these into ***<your_username>*/.globus/certificates** directory
10. Copy your own user p12 file from **~/.globus/permis/** to the **~/.globus/** directory itself and run the following commands to extract your user certificate and key (your password will be the same as the one originally handed out at the beginning of the course and make sure you have renamed your previous user certificates):
    - **openssl pkcs12 -in *filename.p12* -clcerts -nokeys -out usercert.pem**
    - **openssl pkcs12 -in *filename.p12* -nocerts -out userkey.pem**
11. Now save server-config.wsdd and restart the container. Make sure when you start the container that you do so from **$GLOBUS_LOCATION**.

Running a service through PERMIS

Follow these commands to run a grid service through PERMIS

1. Run your grid service as before, but create a proxy certificate for yourself and 'globus' using your own user certificate
2. The client class must have the following three lines in it somewhere so that GSI security methods can be invoked:
    - **((Stub)company)._setProperty(Constants.GSI_SEC_CONV, Constants.SIGNATURE);**
    - **((Stub)company)._setProperty(GSIConstants.GSI_MODE, GSIConstants.GSI_MODE_NO_DELEG);**
    - **((Stub)company)._setProperty(Constants.AUTHORIZATION, NoAuthorization.getInstance());**
3. Note that if an action is unauthorised an AuthorisationException will be thrown. The function of PERMIS is simply to alert the user to authorisation violations by this method, it is up to you to catch and deal with this exception as you see fit.