BioGAT: a Grid Toolkit for Bioinformatics Sequence Alignment

Giovanni Aloisio, Massimo Cafaro, Italo Epicoco, Sandro Fiore and Maria Mirto
CACT/ISUFI and NNL/INFM and CNR - University of Lecce, via per Arnesano - Lecce, Italy
{giovanni.aloisio, massimo.cafaro, italo.epicoco, sandro.fiore, maria.mirto}@unile.it

Abstract

Even though there are many useful tools for sequence alignment such as Blast by NCBI, Fasta by the University of Virginia and Smith-Waterman, by NIST, one of the key issues is that sequence databases are exploding in size, growing at an exponential rate. Grid and parallel computing are crucial techniques to maintain and improve the effectiveness of sequence comparison tools, whereas the Web Services approach may guarantee the interoperability among large collections of programs and data. This work describes BioGAT (Bioinformatics Grid Alignment Tools), a toolkit that offers optimized brokering and a data management system to exploit various bioinformatics alignment tools wrapped as Web Services in a Grid architecture.

Keywords: Bioinformatics, Sequence Alignment Tools, Data Management, Grid Computing, Computational Grid, Web Services.

1. Introduction

One of the key issues in sequence analysis is the determination of similarity between biological sequences, that is the percentage of sequence matches among nucleotide or protein sequences. The hypothesis is that similarity relates to functionality: if two sequences are similar, they will have related functionalities. Only similarity matching makes sense when searching among sequences.

The similarity among sequences is measured through the determination of the optimal alignment of (parts of) sequences, that is the attribution of a score: the higher the score, the higher the similarity between the sequences (*pairwise alignment*).

The sequence alignment is the basis of more complex analyses such as the search for sequence similarities in databases. In this case, there are several applications such as BLAST (Basic Local Alignment Search Tool) [1], FASTA [2] and Smith-Waterman [3], that offer flexible searching through boatloads of data. However, sequence databases are exploding in size, growing at an exponential rate, currently doubling in about 14 months. Grid computing [4] together with parallel alignment tools, wrapped as Web Services [5], are crucial techniques to maintain and improve the effectiveness of sequence comparison.

The opportunity to divide the load as different jobs for each simulation is a good choice for the alignment applied to a large dataset because the data input, composed by a set of sequences (generally in Fasta format), are compared with biological databases that are made of a set of sequences in various formats (flat file and Fasta). So it is possible to split the input dataset and/or the database, send each part of data to a grid node and merge the results.

The solution proposed in this work is based on the possibility to manage the untapped processing power of desktop PCs within an enterprise grid network to process computationally intensive jobs for scientific applications and in particular for the Bioinformatics domain. It is worth nothing here that there are other works in the bioinformatics area, that use this approach such as the SETI@Home [6] and related projects, and Condor Project [7], which in the Grid Computing field pioneered the idea of using the idle time of organizational workstations to do high-throughput computing. However, these are not optimized for sequence alignment using common and legacy biological tools, because tasks in our case are not completely independent: a "simple" merge of the results obtained by parallel tasks is not enough. Indeed, several parameters must be modified on the basis of the dimension of the database used to search for the sequences to be aligned and on the basis of the input sequence. Moreover, the tools are run on biological data banks that must be indexed with opportune software before starting the computation, otherwise these tools do not work. So, customized components such as merging, indexing tools and format translations are needed to support pairwise alignment.

In this work we describe the BioGAT (Bioinformatics Grid Alignment Tool) toolkit which aims at providing a Grid framework to support parallel alignment tools execution. In particular we have chosen the Blast, Fasta and Smith-Waterman tools because these represent a "de facto" standard for sequence alignment.

Our solution exploits a Grid Platform based on the Globus Toolkit 4 [8] and in which each module is deployed as a Web Service (WS). In particular the main modules include:

- Broker, that accepts and schedules user requests on WS Data Access Services, choosing them according to some parameters retrieved by a Grid Information System [9]. It merges all of the results.
- Data Access Service to split the database in chunks and store the results coming from one or more Computation Engines (CEs), that carried out the computation on grid nodes.
- CEs that implements two FIFO queues (in and out) to schedule a single run and, according to the request, dynamically selecting the opportune tool for alignment.

The proposed system allows splitting the computation on a biological dataset among several computational nodes. An important advantage is that this approach (fine grained) is different from others, such as the "simple job farming" (coarse grained) currently used by several grid projects.

BioGAT has been developed as part of the ProGenGrid (Proteomics and Genomics Grid) project [10] jointly with GRelC (Grid Relation Catalog) project [11] at the University of Lecce.

The remainder of the paper is organized as follows. Section 2 describes the Pairwise Alignment Tools that have been embedded in our system, whereas Section 3 presents the approach to optimize the execution of concurrent alignment tools; Section 4 discusses the architecture of BioGAT. Section 5 describes a case study related to BLAST whereas Section 6 concludes the paper highlighting future work.

2. Pairwise Alignment Tools

BLAST, or Basic Local Alignment Search Tool [1], is an alignment tool that uses a measure of local similarity to score sequence alignments to identify regions of good local alignment.

The basic BLAST algorithm can be implemented in DNA and protein sequence database searches, motif searches, gene identification searches, and in the analysis of multiple regions of similarity in long DNA sequences. There are 5 BLAST options on the NCBI web site [12]. BLASTP compares any amino acid query sequence against a protein sequence database. Similarly, BLASTN compares a nucleotide sequence against a nucleotide sequence database. BLASTX takes a nucleotide query sequence and translates it in all reading frames for comparison against a protein sequence database. TBLASTN compares a protein sequence against a nucleotide sequence database, translating the nucleotide database sequences in all possible reading frames. TBLASTX compares translations of a nucleotide query sequence against translations of a nucleotide sequence database. Sequences must be input in one of three formats; FASTA sequence format, NCBI Accession numbers, or GIs (GenBank Identifiers).

The FASTA bioinformatics tool [2], developed by W.R. Pearson at the University of Virginia, provides a quick search and local alignment of sequences contained within a specified database. Results of matched sequences are compared with the best optimum individual alignments between the query sequence and each of the database sequences based on a scoring matrix. The output of FASTA includes a list of aligned sequences from best to worst as well as a histogram showing the distribution of matches relative to a theoretical distribution, statistical scoring of each alignment (Expectation, or E-score value), a percentage related to the overlapping alignment. FASTA offers many of the functionalities provided by BLAST. Although BLAST tools are faster, FASTA provides more accurate sequence alignments.

The Smith–Waterman algorithm [3], on which many applications are based such as JAligner [13], generates an amino acid sequence alignment for sequence similarity. This algorithm is used to search databases for sequences similar to a query sequence. It employs dynamic programming to determine an optimal alignment, obtained by determining what transformations the query sequence would need to undergo to match the database sequence. Transformations include substitution, insertion, and deletion of elements. Scores are assigned for comparisons, using positive numbers for exact matches and negative numbers for some transformations. Scores are obtained from statistically derived scoring matrices. The combination of transformations that results in the highest score is used to generate an alignment between the query sequence and database sequence.

3. Concurrent execution of alignment tools

There are two approaches for the execution of concurrent alignment applications:

- **Database Segmentation.** In this approach, a large sized database is split into several nearly equal-sized databases. Each node stores part of the database. The independent segments of the database are searched on each node, and results are collated into a single output file. Database segmentation has the big advantage that it can eliminate the high overhead of disk I/O. The size of bioinformatics databases are now larger than core memory on most computers, forcing alignment tool searches to page to disk. Database segmentation permits each node to search a smaller portion of the database, thus

reducing (or even eliminating) extraneous disk I/O, and hence, vastly improving tool performance. With sequence databases roughly doubling in size each year, the problem of extraneous disk I/O is expected to persist. The adverse effects of disk I/O are so significant that tool searches using database segmentation can exhibit super-linear speedup versus searches on a single node [14].

- **Query Segmentation.** Query segmentation splits up a set of query sequences such that each node in a cluster or CPU on an SMP system searches a fraction of the query sequences. By doing so, several alignment tool searches can execute in parallel on different queries. Alignment tool searches using query segmentation on a cluster typically replicate the entire database on each node local storage system. If the database is larger than core memory, query-segmented searches suffer the same adverse effects of disk I/O as the traditional alignment tool. When the database fits in core memory, however, query segmentation can achieve quasi-linear scalability for all alignment tool search types [14].

Load-balancing and fault-tolerance are clearly other issues to take into account.

For the BioGAT system we have chosen the database segmentation approach because in our tests we considered a few sequences in input but in the future we would like to exploit an hybrid approach, using both Database and Query Segmentation, depending on the problem, the alignment tool, and the available resources.

4. BioGAT Architecture

As shown in Figure 1, the BioGAT architecture includes:

- Computational Engines (CEs) wrapping the alignment tools as Web Services;
- Split&Merge, that fragments the database according to the policy decided by the Broker, sends produced fragments to the CEs and receives incoming results by the CEs;
- Broker, that manages the requests of clients, decides the database fragmentation policy, formulates the right query to the Split&Merge module and returns the results of the computation to the Client;
- Client, that submits the request to the Broker and receives the results.

This architecture has been designed taking into account the need for extensible modules in order to

- guarantee fast access to the data;
- choose the opportune split policy, as indicated by the Broker that establishes how many sequences must be contained in a single chunk;
- provide plug-ins, dynamically downloaded, for the translation of data in some formats and use the opportune merge algorithm.

Moreover, the data are exchanged securely, using GSI protocol and ACL.

4.1 Scenario

The clients need to perform a sequence alignment using a specified alignment algorithm. Initially, the client (user/application indicated with A, B, etc. in Figure 1) calls the Broker service to submit a given execution request, and supply the parameters needed for the operations (for instance the use of a given alignment algorithm) and the input data. Then the Broker service returns an identifier (ID) that identifies uniquely the processing request.

Selection of alignment algorithms and fragmentation policy are made by the Broker component; it produces a query, in SQL format, and indicates the policy of the fragmentation to be submitted to the Split&Merge Component. The Broker, can be easily extended through a plug-in mechanism, so adding other possible choices is possible at run-time and without updating the system.

Hence, the client can query the Broker service to obtain the results providing the proper ID, or to verify the current status of the operations. The Broker, through the ID, checks the status of the request and if it has been satisfied then applies a merge mechanism, on the basis of the used alignment tool.

Both the Broker and client have plug-in modules allowing the selection of the opportune merge method and the choice of a given web service method in order to set the application parameters. These are stored in a customized data structure and passed to the Broker service.

"Behind the wall" operations

The "behind the wall" operations allow submitting the query to the Split&Merge web service component, described in a previous work [15], that carries out the fragmentation and merges the results for a given request.

For the processing of the request there are two fundamental entities, the Split&Merge component, which aims at

a) fragmenting the data sources;
b) distributing the computational workload on the available resources in a Grid,

and the CEs that are installed on the machines and manage the processing on the fragments produced by the Split&Merge component.

In this context, the main features of the system are:

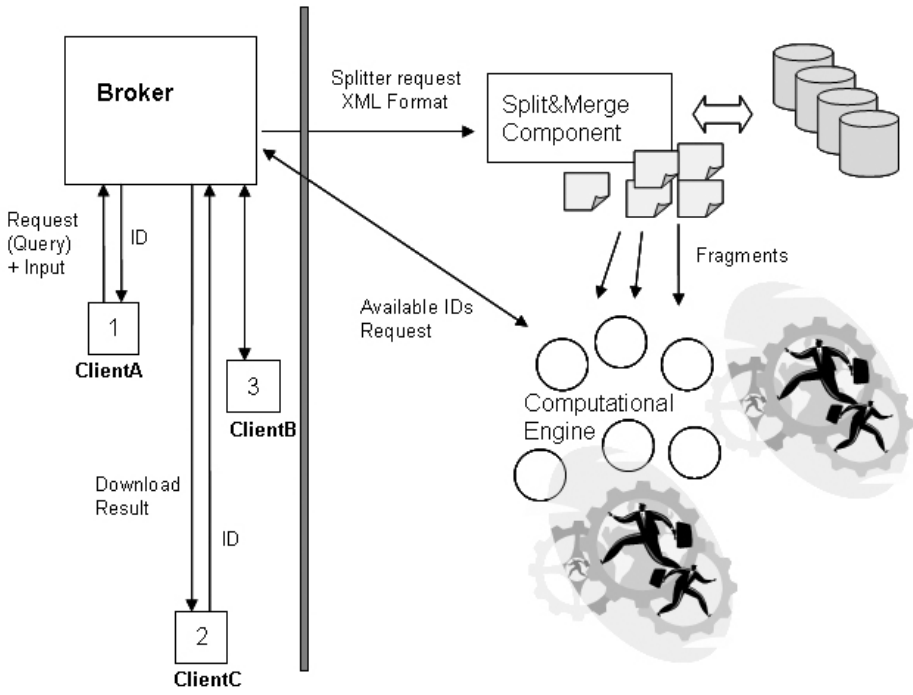

Figure 1. BioGAT Architecture.

- Security;
- Scheduling of the computations and distribution of the fragments to the CEs.

The goal of security and identification of the involved actors in the processing phase are obtained using the infrastructure provided by Globus [8], through the Grid Security Infrastructure (GSI) module.

The fragmentation of data sources is realized through the Split&Merge component, and takes into account the particular processing to carry out and the features of the particular data source, to obtain the maximal efficiency in the fragmentation of data.

Indeed, these features are available, through plug-ins, in order to specify the format of the output fragments used for a specific alignment tool or to specify how to distribute the load to various CEs.

The distribution of the fragments to the Computational Engines occurs on request and is mediated by the Split&Merge that can manage different priority queues and opportune scheduling policies for the selection of the fragments that request priority processing, in order to optimize some performance metric. The module for fragments distribution must manage different protocols for the transfer of the fragments such as GridFTP, scp etc.

The independence of the particular request from the considered database is a fundamental point for each module of the system because it allows building a common infrastructure for a large set of services that have the same implementation requirements.

4.2 Components

The components of this system are wrapped as Web Services and in particular, we have developed them in C, exploiting the gSOAP toolkit [16] (with GSI support available as a gSOAP plug-in [17]). We describe them in the following.

Client: the client represents the interface between the system and final user, and has the following requirements:

- Calling the Broker through the HTTPG protocol (SOAP over GSI) and hence use of the Globus GSI module for authentication;
- Verifying the right formulation of the request in terms of parameters and user input;
- Calling the Broker for the submission of a request;
- Calling the Broker to retrieve the status of a request being processed;
- Calling the Broker to download the results of a request already processed.

Broker: the Broker manages the interactions between all of the components of the system. It is the most important component and its requirements are:

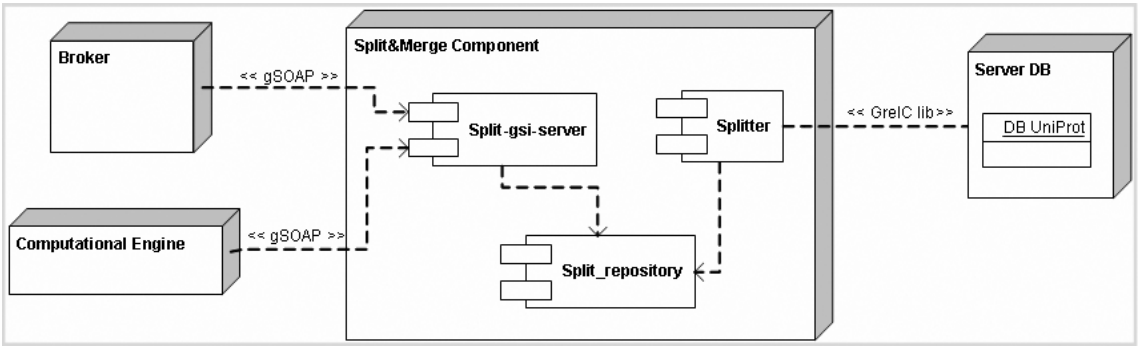

Figure 2. Split&Merge component.

- Verifying and accepting the client processing requests;
- Scheduling the requests with a FIFO policy;
- Calling the Split&Merge component to run a query on the data sources and fragmenting the result set;
- Storing the parameters and input of a request;
- Providing on request to the Computational Engines the ID of a processing on the basis of its priority and parameters, together with input and access url of the Split&Merge component to download the fragments;
- Monitoring periodically the progress of the request calling the Split&Merge component in order to provide the client with information related to the request;
- Collecting the produced results for a given query calling the Split&Merge component and applying the opportune merge algorithm on the basis of the specified processing.

Split&Merge Component: this component is called by the Broker and its main feature consists in splitting the data sources in fragments using an opportune policy for distributing/balancing the computational load to the Computational Engines. The requirements of the Split&Merge component are:

- Adopting an opportune fragmentation to divide the data source considering the processing that will use the fragments to obtain a balanced workload;
- Formatting the fragments considering the specific processing;
- Collecting the results of the processing carried out on fragments and sending them on request to the Broker;
- Handling a queue of fragments to satisfy (best effort) the requests of the Computational Engines.

The Split&Merge component is composed by two software modules and a repository (see Fig. 2).

Split-gsi-server: it is the module that is responsible for the interaction with the other components of the system. Regarding the interaction with the broker, this component handles three types of requests:

- Processing of a query and fragmentation of the result set;
- Information on the progress of a specific processing, for which there is the need to log of all of the events related to both the production and distribution of the fragments and information about the sending of results;
- Sending the produced results and storing them into a repository.

Regarding the interaction with the Computational Engines, the *Split-gsi-server* manages the requests for sending fragments and receives the produced results.

Splitter: it is the main module of the Split&Merge component, indeed it reads the requests, submits the query through the GRelC interface and produces the fragments on the basis of opportune policy and format.

Split_repository: it is the data storage of all of the data needed to execute the Split&Merge component, indeed it stores:

- Fragmentation requests that must be processed by the splitter;
- Fragments generate by the splitter that must be sent to the broker;
- Produced results that must be sent to the broker;
- Log files of the operations carried out for each request.

Currently it is implemented as GRelC Data Storage service.

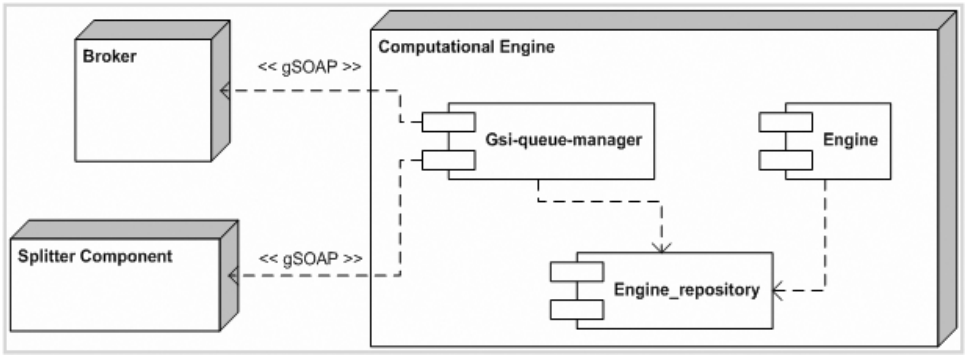

Figure 3. Computational Engine component.

Computational Engine: it applies the request algorithm on fragments, taking into account the parameters and input sent by the user. It is composed by two software modules and a repository (Fig. 3).

Gsi-queue-manager: it communicates with other modules of the system and manages the queue length of fragments available on the Computational Engine. The management policy of the queue is a fundamental aspect for the performance of the system. In particular we try to pursue two goals: there should always be fragments available in the queue, in order to avoid dead times between processing a fragment and the next one; on the other side, it is not possible to collect an excessive number of fragments because if a large set of fragments of a given query is assigned to a few computational engines, the time to fully complete the processing may become higher w.r.t. a distribution of the computational workload on a higher number of machines. In order to achieve these goals, the request of a new fragment is constrained both by the max number of fragments available into the queue and the workload of the CPU.
The responsibilities of the gsi-queue-manager are:

- Calling the broker to obtain the data of the requests awaiting to be processed;
- Downloading the fragments through a call to the Split&Merge component, on the basis of the queue policy;
- Verifying the availability of results and sending them to the Split&Merge component.

Engine: The engine is the module that executes computation, in particular it is responsible for:

- examining the pending requests and calling the opportune algorithm, fixing the parameters and input;
- getting from the queue the fragments related to the current request and starting the processing.

Engine repository: this component maintains all of the data needed by the Computational Engine and in particular:

- The information related to the requests: parameters for the processing, input and access url of the Split&Merge to download the data;
- The fragments of the database downloaded from the queue;
- The produced results to be sent to the Split&Merge component.

4.3. Interaction between Components

The interaction between different components of the system is carried out by following some steps starting with the processing request of the client. These steps are:

1. **Submission of a request:** the client calls the broker that carries out the operations to serve the request and builds a query for the Split&Merge component. In particular, the Broker has a request queue where each element is the logical name of a zipped directory containing:
   i) the input parameters in a XML file;
   ii) the input file in Fasta format;
   iii)contact.xml that contains some information about the access url of the split-gsi-server, and the ID of the query.
   When the computation engine asks if there are IDs available, the broker sends it the zipped file and enqueues the element in the queue only after checking (with a call to the split-gsi-server) that the request has been completely satisfied.
   When the split-gsi-server is called to fragment the data sources, it writes a request file in the Split_repository, the splitter will examine the file and will carry out the needed operations.
2. **Fragmentation**: the splitter checks the request files written by the split-gsi-server and starts the processing. Considering the information contained

in the examined request file, a query is submitted through the use of the GRelC libraries and an opportune algorithm is chosen in order to adopt an opportune fragmentation policy and formatting the obtained results.

3. **Transfer of fragments to the Computational Engines:** the transfer of the fragments is managed by the gsi-queue-manager, that interacts with both the broker to obtain the requests to be processed and with the split-gsi-server to download the fragments. The queue manager is responsible also for sending the results produced by the split-gsi server. In order to obtain optimal performances in the computational phase, the upload of results occurs only if the machine does not exhibit an high load, while to avoid queueing too many fragments, the download occurs only if the machine does not exhibit high load and the queue length is below a given threshold.
4. **Processing:** the processing is carried out by the engine, that does the spool of the repository, selects a directory related to the processing request and selects an opportune algorithm. Then it examines the queue of fragments that is filled by the gsi-queue-manager and processes the available fragments. When it finishes the current processing, it starts processing the next query.
5. **Results Merge:** the broker carries out the merging of results. It maintains the status of the processing of all of the queries that have been submitted at regular intervals, or on the basis of the workload condition, for each query in the list it queries the splitter component which has been assigned the fragmentation and updates the processing status. If results are available, or when the processing is finished, it gets the results querying the split-gsi-server and applying an opportune merge algorithm.

5. Case Study: Blast tool

In this section we describe the infrastructure used to run the Blast tool exploiting the BioGAT system.

We have downloaded the NCBI's Blast software [18] and put it in a directory of our installation package. Then, we have compiled all of the software (including NCBI software) on an HP XC6000 cluster with 128 Itanium 2 cpus.

The syntax for Blast Web Service client is the same of the Blast toolkit so that the user already knows the meaning of the parameters.

For our test we have chosen an alignment of amino acid sequences among a target protein, 1LYN, that is a fertilizant protein, with the data contained in the UniProt database [19].

We have put the sequence in a file in Fasta Format and passed it on the command line (-i option). Moreover, we have specified the database name and other parameters.

The data was processed by the broker and stored in a zipped directory that contains the following xml files and the input file:

1. contact.xml contains the ID and the access url of the split-gsi-server;
2. <id_query>_<timestamp>.xml contains all of the input parameters passed by the command line, and a timestamp in GMT format;
3. input file in fasta format.

Here is an example of the contents of these files:

Contact.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ProGenGrid_Utility>
    <id type="NOT STRING">
        100949Bz9518dnCOTBXSG</id>
    <Split_WS type="STRING">
        httpg://sigma1.unile.it:23000</Split_WS>
</ProGenGrid_Utility>
```

<id_query>_<timestamp>.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ProGenGrid_programs>
 <execution_cmd name="blastall">
  <arguments>
   <argument name="prog" option="-p">
       blastp</argument>
   <argument name="i_query" option="-i">
       input.blast</argument>
   <argument name="o_query" option="-o">
       output.blast</argument>
   <argument name="exp" option="-e">
       10.00</argument>
   <argument name="m_align" option="-m">
       0</argument>
   <argument name="filter" option="-F">
       T</argument>
   <argument name="gap" option="-G">
       0</argument>
   <argument name="ext" option="-E">
       0</argument>
   <argument name="x_drop" option="-X">
       0</argument>
    …
  </arguments>
 </execution_cmd>
</ProGenGrid_programs>
```

When the Gsi-queue-manager asks to run a new task, the broker sends it the zipped file, so that the access url information is used by the queue manager to contact the Split&Merge component while the ID is used to query and retrieve the new fragments.

The engine component uses the execution_command TAG to select the request algorithm while the other tags are cached to be used when a new fragment will be sent.

Both files sent by the broker and those containing the fragments or the results are in a text file or XML format so these are at first compressed with Lempel Ziv algorithm and then sent to the Split&Merge service where they will be decompressed. Moreover, we used an option of gSOAP Toolkit to further compress the files.

The engine carries out the spool of the engine_repository searching for queries to be processed, reads the parameters, selecting the opportune algorithm, and then searches the queue to retrieve related fragments.

The engine, having read the "execution_cmd" tag, searches the map of registered libraries to obtain the pointer to the construct of the ConcreteCommand object that implements the requested processing function.

Regarding to the Blast algorithm, for each fragment is run at a first the formatdb command, in order to index the data present into the fragment, and then the blastall command, using the execution parameters available in the request file. Blast algorithm can be applied if and only if the data are indexed with the formatdb command.

The results are sent to the split-gsi-server. The Split&Merge checks the result of the computation through a log file built for each request indicating how many fragments have been produced, how many fragments have been received by the gsi-queue-manager and the state of the fragmentation.

An example of the xml log file is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<SPLIT_QUERY_STATE>
   <ID_QUERY>100949Bz9518dnCOTBXSG</ID_QUERY>
   <TOTAL_FRAGMENTS_NUMBER>12
         </ TOTAL_FRAGMENTS_NUMBER>
   <FRAGMENTS_PRODUCES>8
         </FRAGMENTS_PRODUCES>
   <FRAGMENTS_SEND>5</FRAGMENTS_SEND>
   <FRAGMENTION_STATE>running
         </FRAGMENTION_STATE>
   <RESULTS_REACHED_NUMBER>3
         </RESULTS_REACHED_NUMBER >
</SPLIT_QUERY_STATE>
```

Finally the broker will carry out the merging of results. In this case the Blast algorithm requires not a simple merge among the partial output but an adjustment of e-value depending on the dimension of the database and on the length of the query.

6. Conclusions and future work

We have presented an infrastructure for running concurrent alignment applications in a distributed system. The main feature of the BioGAT platform is the decoupling between the client requests, the database partitioning and the alignment processing that allows exploiting the Grid resources balancing the workload. BioGAT is extensible, flexible and scalable at three levels: Broker, Split&Merge and Computational Engine. We plan to make tests, at first comparing this approach with others, such as mpiBlast [20], dBlast [21] etc. and then we will integrate this approach in the ProGenGrid platform and in particular in its Workflow to simulate complex experiments in Bioinformatics.

7. References

[1] S. F. Altschul, G. Warren, W. Miller, E. W. Myers, and D. J. Lipman. "Basic local alignment search tool". *J. Mol. Biol.*, 1990, 215:403-410.

[2] W.R. Pearson. "Flexible sequence similarity searching with the FASTA3 program package". *Methods Mol. Biol.*, 2000, 132:185-219. Site address: ftp://ftp.virginia.edu/pub/fasta/.

[3] T. F. Smith and M. S. Watermann. "Identification of common molecular subsequence". *Journal of Molecular Biology*, 1981, 147:196–197.

[4] I., Foster, C., Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[5] Kreger, H. "Web Services Conceptual Architecture". *WSCA* 1.0. IBM, 2001.

[6] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D.Werthimer. "SETI@home: An experiment in public resource computing". *Communications of the ACM*, Nov. 2002, Vol. 45 No. 11, pp. 56-61.

[7] M.J. Litzkow, M. Livny, M.W. Mutka. "Condor – A Hunter of Idle Workstations". *Proceedings of the 8th International Conference of Distributed Computing Systems*, June, 1988, pp. 104-111.

[8] I., Foster, C., Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", *Intl J. Supercomputer Applications*, Vol. 11, 1997, No. 2, pp. 115-128.

[9] G. Aloisio, M. Cafaro, I. Epicoco, S. Fiore, D. Lezzi, M. Mirto and S. Mocavero, "iGrid, a Novel Grid Information Service", *Proceedings of the First European Grid Conference (EGC) 2005*, LNCS 3470, Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 506—515, P.M.A. Sloot et al. (Eds.).

[10] G. Aloisio, M. Cafaro, S. Fiore, M. Mirto. "ProGenGrid: a Grid-enabled platform for Bioinformatics". *From Grid to HealthGrid Proceedings of HealthGrid 2005*, Tony Solomonides et al. (Eds), IOS Press, Volume 112, 2005, pp. 113-126, ISSN 0926-9630.

[11] G. Aloisio, M. Cafaro, S. Fiore, M. Mirto. "The GRelC Project: Towards GRID-DBMS". *Proceedings of Parallel and Distributed Computing and Networks* (PDCN) IASTED, Innsbruck (Austria) February 17-19 2004. Site address: http://gandalf.unile.it.

[12] National Center for Biotechnology Information. NCBI BLAST home page, 2003. Site address: http://www.ncbi.nlm.nih.gov/blast.
[13] A. Moustafa, "*JAligner: Open source Java implementation of Smith-Waterman"*. Site address: http://jaligner.sourceforge.net.
[14] GeB, a GRID-enabled parallel version of NCBI BLAST based on ProActive. Site address: http://www-sop.inria.fr/oasis/ProActive/.
[15] G. Aloisio, M. Cafaro, S. Fiore, M. Mirto. "A Split&Merge Data Management Architecture for a Grid Environment". To appear in *Proceedings of Computer-Based Medical System (CBMS) 2006*. 22-23 June, Salt Like City (USA).
[16] G. Aloisio, M. Cafaro, D. Lezzi, R. Van Engelen, "Secure Web Services with Globus GSI and gSOAP", *Proceedings of Euro-Par 2003*, 26th - 29th August 2003, Klagenfurt, Austria, Lecture Notes in Computer Science, Springer-Verlag, N. 2790, 2003, pp. 421-426.
[17] G. Aloisio, M. Cafaro, I. Epicoco, D. Lezzi, R. Van Engelen, "The GSI plug-in for gSOAP: Enhanced Security, Performance, and Reliability", *Proceedings of Information Technology Coding and Computing (ITCC 2005)*, IEEE Press, Volume I, 2005, pp. 304-309.
[18] National Center for Biotechnology Information. Site address: http://www.ncbi.nlm.nih.gov/BLAST/.
[19] UniProt - EBML- EBI European Bioinformatics Institute. Site address: http://www.ebi.ac.uk/uniprot/database/download.html.
[20] A. Darling, L. Carey, and W. Feng, "The Design, Implementation, and Evaluation of mpiBLAST". *ClusterWorld Conference & Expo* in conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003, San Jose, CA, June 2003. Site address: http://mpiblast.lanl.gov/.
[21] M. A. van Driel, M. L. Hekkelman, and R. Rodriguez, "dBlast. A wrapper to run NCBI BLAST parallel/distributed". Submitted. Site address: http://www.cmbi.kun.nl/software/dBlast/.