BGBlast: A BLAST Grid Implementation with Database Self-Updating and Adaptive Replication

Gabriele A. TROMBETTI [a,b], Ivan MERELLI [a], Alessandro ORRO [a,b,1] and Luciano MILANESI [a]

[a] *Institute for Biomedical Technologies – National Research Council, via Fratelli Cervi 93, 20090 Segrate (MI), Italy*

[b] *Consorzio Interuniversitario Lombardo per L'Elaborazione Automatica, via R. Sanzio 4, 20090 Segrate (MI), Italy*

Abstract. BLAST is probably the most used application in bioinformatics teams. BLAST complexity tends to be a concern when the query sequence sets and reference databases are large. Here we present BGBlast: an approach for handling the computational complexity of large BLAST executions by porting BLAST to the Grid platform, leveraging the power of the thousands of CPUs which compose the EGEE infrastructure. BGBlast provides innovative features for efficiently managing BLAST databases in the distributed Grid environment. The system (1) keeps the databases constantly up to date while still allowing the user to regress to earlier versions, (2) stores the older versions of databases on the Grid with a time and space efficient delta encoding and (3) manages the number of replicas for each database over the Grid with an adaptive algorithm, dynamically balancing between execution parallelism and storage costs.

Keywords. Bioinformatics, adaptive database replication.

Introduction

BLAST [1], [2] is a well known and widely used bioinformatics application for comparing (usually unknown) “query” biological sequences, either genomic or amino-acidic, against a set of known “reference” sequences (“Blast Reference Database” or BRD in this paper). BLAST is a variation and approximation of the exhaustive dynamic-programming Smith-Waterman [3] algorithm for local sequence-alignment, resulting in a speed increase of 10-100x, at the expense of some sensitivity [4].

While BLAST sensitivity is generally regarded as still adequate for most circumstances, the speed of BLAST can still be scarce for certain massive computations, which are in fact performed rather commonly by many bioinformatics research groups.

The problem of BLAST speed can be addressed in various ways, the solutions usually belonging to the following groups (a) faster alternatives to BLAST, (b) BLAST execution on clusters and (c) BLAST execution in Grid. Pros and cons for (a) and (b)

[1] Corresponding Author: Alessandro Orro, Institute for Biomedical Technologies – National Research Council, via Fratelli Cervi 93, 20090 Segrate (MI), Italy; E-mail: alessandro.orro@itb.cnr.it

will be mentioned in the next chapter. The solution we present in this paper belongs to (c). As far as (c) is concerned, the main problems usually arising from BLAST execution in Grid are:

1. Defining and enforcing a policy for replication of the BRDs over the Grid. BRDs are large files needed during BLAST execution over the Grid. Due to their size they require allocation on Grid Storage Elements (SEs). Rising the number of replicas for a BRD reduces the Grid queue times for BGBlast runs using that BRD, but also rises the associated storage costs (see chapter 2.2.1 below). Due to the significant size, it is not reasonable to replicate every BRD on a large number of SEs; compromises have to be made.
2. Keeping the replicated BRDs up-to-date.
3. Optionally it might be profitable to store older versions of the BRDs so that BLAST users can reproduce and verify results obtained in the past. The problem in providing this feature is that keeping older versions of BRDs available normally has a very high storage cost.

In the Methods section more details are given regarding the above issues and on how we were able to address them.

1. Related Work

As mentioned in the Introduction, a number of solutions have been developed to address the problem of BLAST speed for large BLAST runs. In this chapter we will report about the main approaches.

1.1. Faster alternatives

Various alternatives to BLAST which are faster and similar in scope are available such as MegaBLAST [5], [6], BLAT [7] and PatternHunter [8]. These alternatives usually are different enough to be not suitable for exactly the same situations as BLAST is, or sometimes can have different drawbacks. As far as the examples are concerned, MegaBLAST and BLAT, albeit much faster, have a lower sensitivity than BLAST. PatternHunter on the other hand claims a similar sensitivity but is a commercial closed source product, and the algorithm is not known exactly. Such drawbacks might or might not be acceptable for the research, depending on the specific circumstances. In addition, researchers aiming at publishing their results might want to use specifically BLAST simply because its reliability is well established and cannot be object of discussion.

1.2. Cluster execution

Various solutions [9], [10], [11] have been developed to parallelize the BLAST algorithm for execution on computing clusters and supercomputers. These solutions have been used for quite some time now and are regarded as reliable. The main drawback of cluster execution for BLAST is the initial cost for purchasing the dedicated cluster, which is high, and might be unreasonably high -relatively speaking- in case the cluster is not going to be used full-time (uneven workloads).

1.3. Grid execution

A number of implementations of BLAST for the Grid environments already exist [12], [13], [14] but in general suffer from the problems already mentioned in the introduction. In this paper we present BGBlast, another Grid implementation for BLAST which we developed evolving the earlier GridBlast project carried out by Merelli and Milanesi [14]. In BGBlast we successfully addressed the issues mentioned in the introduction.

2. Methods

BGBlast (BioinfoGridBlast) has some unique advantages over the existing solutions. BGBlast is an innovative porting of BLAST onto the Grid providing the following capabilities (1) automatic update of the biological databases handled by BGBlast (2) adaptive replication of databases on the Storage Element Grid nodes (3) version regression for the biological databases. BGBlast is the evolution of the earlier project GridBlast [14] on top of which the features (1), (2) and (3) have been added:

1. Automatic Database Updater (ADU): ensures the users always work with the latest version of every Blast Reference Database (BRD), and this without needing human staff to manually monitor the release of newer versions of BRDs or manually performing database updates over the Grid.
2. Adaptive Replication (AR) for the BLAST Reference Databases: ensures that the most used BLAST databases are replicated more times than less used databases. The optimal number of replicas for each BRD is calculated dynamically based on the relative usage of the specific database in recent times. This keeps a constant optimization of Grid queue times vs Grid storage costs.
3. Version Regression for BLAST Reference Databases: allows the user of BGBlast to specify an older version of a certain BRD to be used for the computation. This allows the user to reproduce exactly computations obtained in the past, something that might be needed to confirm results that were obtained. The storage of older version of BRDs is performed with a delta-encoding efficient in both space (storage costs) and time (a short download time and a short time to patch a BRD for regressing it to an earlier version).

2.1. GridBlast core

BGBlast is composed of the following three functional parts: GridBlast core, Database Version Regression (DVR) and Automatic Database Updater (ADU). Here follows a more detailed description. GridBlast [14] is still the core for BGBlast, providing the following capabilities:

1. Factor J parallelization of large BLAST executions. This is done by splitting the user input into J even subset, each taking 1/J of the original time to execute. This is followed by the submission of J smaller BLAST jobs (1/J of query sequences against the target BRD) on the EGEE [17] Grid platform. J is chosen so that jobs of reasonable length are created: neither too small (Grid overhead would be comparatively large) nor too big (insufficient parallelization).

4. A rate limiting feature triggered on very large BLAST executions. This limits the rate at which the jobs are submitted to the Grid so to avoid a sudden massive Grid exploit.
5. Monitoring of every launched launched job and automatic resubmission in case of failure. This is still important nowadays, as the Grid platform is still new and reliability is not excellent.
6. Fetching of the results back after the completion of the Grid jobs. Merging of such results into a single BLAST results file.
7. A recent improvement of the core provides measurements of the queue times and CPU hours consumed by the J Grid jobs for each run of BGBlast. These measurements are passed to the Adaptive Replication Manager and are essential for the correct functioning of the AR functionality (see).

On top of the GridBlast core, the following functionalities have been implemented:

2.2. Adaptive Replication Manager (ARM)

The Adaptive Replication of BLAST Reference Databases is a BGBlast feature for optimizing the number of replicas for each BLAST database dynamically and adaptively.

2.2.1. Motivation for ARM

BLAST Reference Databases (BRDs) are large files, usually in the range 500MB-5GB, and are needed during the run of BLAST on the Grid CEs for each of the J BGBlast-generated jobs. Due to their size, it is not reasonable to download a BRD from a remote location. It is hence necessary to constrain the J jobs to execute on CEs having a replica of the user-requested BRD on a near (local network) SE.

Due to this constraint, the number of CEs to choose from for the BGBlast generated Grid jobs is limited. This impacts the queue times negatively and this is particularly true if the replicas of the requested BRD are few. A massive replication of every BRD on all the SEs of the Grid is not feasible either, because of their size which would make the storage costs unbearable.

Clearly, it is more useful to have additional replicas for BRDs used often, so that the queue times are reasonably small for the most common BGBlast runs, while it is better to have fewer or possibly only one replica for the BRDs used less frequently, in order to reduce the Grid storage costs.

Since the amount of usage of for each of the BRDs cannot be known in advance, we have implemented a dynamic, adaptive replication mechanism to balance between queue times and storage costs.

2.2.2. Methods for ARM

The ARM performs a D days moving average (usually D=10) of the CPU hours and queue times used for each reference database. This statistical measurement is used to compute the optimal number of replicas for each of the BLAST reference databases. This algorithm balances between the additional storage costs incurred in increasing the number of replicas and the benefit of the reduced queue times.

Additionally, when evaluating the addition of a replica the ARM engine also evaluates which of the SEs would be the most advantageous for a replica addition. Similarly, when evaluating the benefit of removing a replica, the ARM engine also evaluates the least advantageous of the currently existing replicas, that is, best for removal. See below for further details on the algorithm.

The measurements of used CPU-hours and queue times experienced for each BGBlast run, and implicitly for each BLAST database, are provided to the ARM by the GridBlast core (see). The dynamic variation of the number of replicas is evaluated, and possibly performed, at each BGBlast run and at the end of each day.

2.2.3. Algorithm details

BGBlast's ARM optimizes the number of replicas for each BRD separately, by minimizing the sum of the storage cost and user wait time cost. The algorithm is an iterative algorithm which converges on the optimal number of replicas and the optimal location for them, simultaneously.

The ARM optimization algorithm at each cycle evaluates the benefit of the *addition* of one replica and the benefit of the *removal* of one replica.

During the evaluation of the *addition* of one replica, the ARM takes into account the specificity of each Grid location suitable for replication (i.e. every SE not yet holding a replica), hence finding the best location for an added replica. The ARM then evaluates whether the addition of a replica in that specific place is profitable or not, using the costs formula.

The best location for adding a replica is ideally a SE having a large amount of free disk space (so as to cause proportionally little impact when adding the replica) near a CE being large in the number of nodes (a larger computing power means that the job queue is generally consumed more quickly).

The costs formula for evaluating variations in replicas numbers considers the Grid queue times to be inversely proportional to the number of nodes useable by BGBlast, i.e. those having a replica nearby (see chapter 2.2.1). The correctness of this assumption can in fact be demonstrated under some simplifying assumptions. The cost of a minute of user wait time is to be specified in the BGBlast configuration file.

The cost of Grid storage is the other cost to be specified in the BGBlast configuration file. The cost of storage is to be expressed in terms of cost per percent of free storage space occupied per day on a SE. This approach was chosen for reflecting the intuitively higher impact on other Grid users that a GB-sized file has when uploaded on a small or already full SE compared to the impact it has when uploaded on a SE with plenty of free space.

The ARM engine hence works by minimizing the sum of the storage cost and user wait time cost, for each BRD separately.

The process for evaluating the benefit of the removal of one replica is analogous. The worst existing replica is chosen using the same kind of analysis as described above. The cost formula is then recomputed while simulating the removal of the “worst” replica, and the result obtained in this way is compared to the cost associated to the current situation. If the cost after the removal of the replica appears lower, the replica is removed.

This algorithm converges quickly.

2.3. Automatic Database Updater (ADU)

BGBlast's ADU engine constantly monitors FTP sites for newer versions of the BRDs registered to be handled by BGBlast. If a newer version of a BRD is detected, the ADU automatically updates all the replicas of such BRD over the Grid. This is not the only action performed by the ADU: the ADU also computes an xdelta patch for regressing the newer version of the BRD to the earlier version of the BRD now being replaced, and uploads the said xdelta patch on a predefined SE. The xdelta patch computed by the ADU, together with the xdelta patches computed during previous database updates, is needed for the DVR functionality (see).

Such xdelta patches are many times smaller than any version of the BRD they refer to, and this makes the storage costs reasonable. In order to further reduce the storage costs, we decided to keep only one replica for the xdelta patches. Also see chapter 2.4 on this topic.

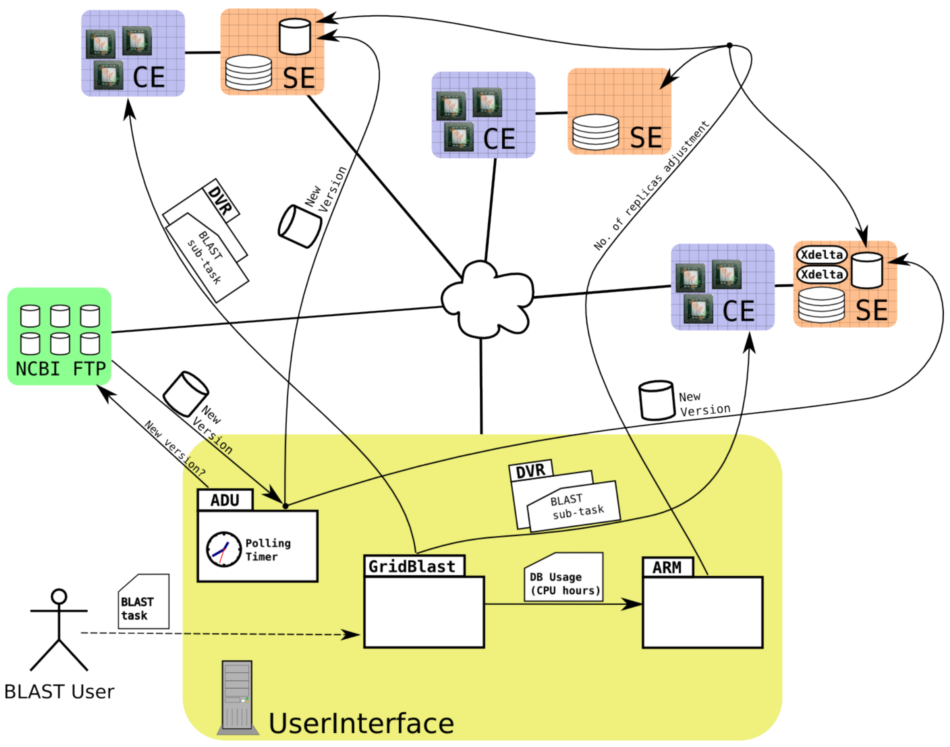

Figure 1. *Interactions between parts of BGBlast and Grid elements.* The user interacts with the *GridBlast* core for launching a BLAST task on the Grid. The BLAST task is split by GridBlast in jobs of equal size (“*sub-tasks*”) and sent to Computing Elements (*CEs*) for execution. The code for performing the *DVR* is sent to the CEs together with the Grid jobs. The xdelta patches are stored on a single SE (not replicated) as shown, however, the interaction between the DVR code and the xdelta patches is not shown in the figure. The *ADU* uses a timer-based polling to detect and fetch new versions of a database from the *FTP* site of origin, then it updates the databases located on the Storage Elements (SEs). The *ARM* receives notification of CPU time spent on a database by the GridBlast core, then it adjusts the number of replicas of the database on the SEs.

2.4. Database Version Regression (DVR)

BGBlast provides an option for specifying a *version* (in terms of *date*) of the BRD to be used for the BLAST computation, along with the *name* of the BRD. The requested version of the BRD is obtained from the latest version of the BRD by applying the ADU-generated xdelta patches in sequence, from the newest to the oldest. Each xdelta patch regresses the BRD by one version, and this action is performed until the requested version is reached.

The version regression operation is performed on the Computing Element (CE) after the download of the BRD from the near (local network) SE and prior of starting the computation.

The download of the xdelta patches is generally remote, as the patches are only replicated once on the Grid (see chapter 2.3), and this is in contrast with the download of the BRD (latest version) which is over a local network (see chapter 2.2.1). However, due to the small size of the patches, the patches' download time rarely surpasses that of the BRD (latest version). Since the DVR is also a relatively uncommon request by users, we considered the patches download time an acceptable overhead.

3. Conclusion

We have shown how we were able to improve BLAST performances by distributing the BLAST execution on the EGEE Grid infrastructure, leveraging the power of thousands of CPUs. Additionally we have shown how we could further reduce the queue times while impacting the Grid storage costs as little as possible, by using adaptive replication for BLAST databases, and how we were able to provide version regression for such BLAST databases. Our work can shorten biologists' waiting times for their research, and also acts as a proof of concept showing what can be done to optimize Grid resources and Grid applications.

BGBlast will be available as a service for EGEE in the upcoming BioinfoGRID portal by CNR-ITB. CNR-ITB will be in charge of the maintenance for the Blast biological databases on the Grid.

Acknowledgment

This work was supported by the Italian FIRB-MIUR project “Italian Laboratory for Bioinformatics Technologies – LITBIO” [15] and by the European Specific Support Action BioinfoGRID [16] and EGEE [17] projects.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers and D.J. Lipman: **Basic Local Alignment Search Tool**, *J. Mol. Biol.* 1990;215(3):403-10

[2] BLAST - http://www.ncbi.nlm.nih.gov/BLAST/

[3] T.F. Smith and M.S. Waterman: **Identification of Common Molecular Subsequences**, *Journal of Molecular Biology* 1981;147:195-7

[4] http://searchlauncher.bcm.tmc.edu/help/AlignmentScore.html

[5] Z. Zhang, S. Schwartz, L. Wagner, and W. Miller (2000): **A greedy algorithm for aligning DNA sequences**, *J. Comput. Biol.* 2000;7:203–14

[6] http://www.ncbi.nlm.nih.gov/blast/megablast.html

[7] W.J. Kent: **BLAT – the BLAST-like alignment tool**, *Genome Res* 2002;12:656–64

[8] B. Ma, J. Tromp, and M. Li: **PatternHunter: faster and more sensitive homology search**, *Bioinformatics* 2002;18(3):440–5

[9] Y. Qi, F. Lin: **Parallelisation of the blast algorithm**, *Cell Mol Biol Lett.* 2005;10(2):281–5

[10] D.R. Mathog: **Parallel BLAST on split databases**, *Bioinformatics* 2003;19(14):1865–6

[11] A. Darling, L. Carey, and W. Feng: **The Design, Implementation, and Evaluation of mpiBLAST**, *4th International Conference on Linux Clusters* June 2003; San Jose, CA

[12] M. Bayer, A. Campbell, D. Virdee: **A GT3 based BLAST grid service for biomedical research**, *Proceedings of the UK e-Science All Hands Meeting* 2004

[13] F. Konishi, and Y. Shiroto, and R. Umetsu, and A. Konagaya: **Scalable BLAST Service in OBIGrid Environment**, *Genome Informatics* 2003;14:535–6

[14] I. Merelli, L. Milanesi: **High performance implementation of BLAST using GRID technology**, *Proceedings BITS* 2005: p.59

[15] LITBIO – Laboratory for Interdisciplinary Technologies in Bioinformatics – http://www.litbio.org

[16] BioinfoGRID – Bioinformatics Grid Applications for Life Science – http://www.itb.cnr.it/bioinfogrid

[17] EGEE – Enabling Grids for E-SciencE – http://public.eu-egee.org/