BLAST Parallelization on Partitioned Databases with Primary Fragments *

Daniel Xavier de Sousa[1], Sergio Lifschitz[1] and Patrick Valduriez[2]

[1] PUC-Rio Dep Informatica, Rio de Janeiro - Brazil
{dsousa,sergio}@inf.puc-rio.br
[2] INRIA and LINA, Nantes - France
patrick.valduriez@inria.fr

Abstract. As a result of recent advances in sequencing methods, genomic databases are getting larger and larger, thus raising performance issues for bio-sequence analysis tools. In this paper, we consider BLAST, one of the most popular such tools. To increase performance in large databases, much work has considered the evaluation of BLAST in distributed and parallel environments like clusters and Grids. We propose a new parallelization approach to execute BLAST in distributed and parallel environments. We use a replicated allocation of the (sequences) database, where each copy is physically fragmented. We investigate two dynamic load balancing methods that exploit our database allocation. Our preliminary experimental results based on a cluster indicate that our approach achieves both very good speedup and good load balancing.

1 Introduction

In this paper, we consider one of the most popular tools in bioinformatics, namely BLAST - Basic Local Alignment Search Tool - evaluation [2]. BLAST provides a popular family of algorithms for (bio)sequences comparison and alignment operations. These operations are widely used in laboratories that make Genome sequencing and analysis.

Except for single input queries and small sequence databases, BLAST processing is very time consuming. But as genomic databases are getting larger and larger, performance becomes a key issue for BLAST. To increase the performance of BLAST in large databases, many distributed and parallel strategies using clusters and Grids have been proposed (e.g. [8, 1]).

From a database point of view, there are two basic approaches: replication and fragmentation. With replication, the sequence database is fully replicated at all processing nodes (single nodes or clusters) and the query is partitioned into subqueries, each running at a different node. With fragmentation, the database is split into disjoint fragments and the complete query is executed at all sites [4]. While the replication approach is straightforward (inter-query parallelism), the

* Work partially funded by CAPES-COFECUB (DAAD project) and CNPq-INRIA (GriData project)

fragmentation approach is more difficult: BLAST execution on smaller fragments may not generate the correct (sequential) results if runtime statistical parameters (e.g. Z for WU-BLAST [12] and Y for NCBI-BLAST [9]) are not well defined [7]. Furthermore, as for other parallel computation problems, an uneven workload may yield poor load balancing, thus reducing the benefits of parallelism [5].

In this paper, we propose a new parallelization approach to execute BLAST in distributed and parallel environments, considering load balancing. We use a replicated allocation of the (sequences) database, where each copy is physically fragmented. We investigate two dynamic load balancing methods that exploit our database allocation. Our preliminary experimental results based on a cluster environment indicate that our approach achieves both very good speedup and good load balancing.

The rest of this paper is organized as follows. We discuss BLAST parallelization, in particular load balancing, in Section 2. In Section 3, we present our approach for database allocation and BLAST parallelization. Section 4 gives our preliminary experimental results and Section 5 concludes.

2 Motivations

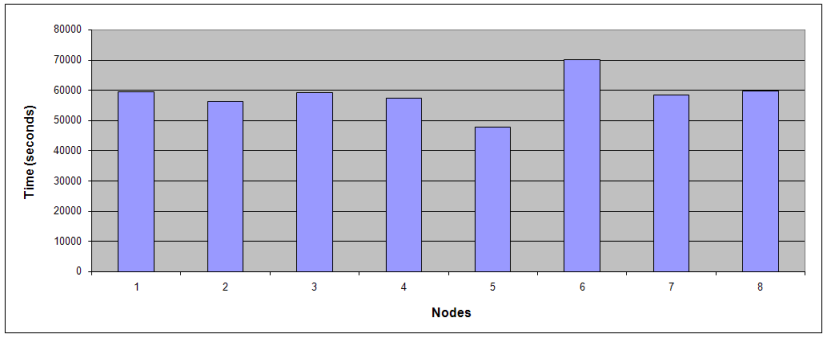

Fig. 1. Uneven Workload for Equal DB sizes

The work in [4] presents a detailed discussion and implementation results regarding database distribution in order to execute BLAST in a cluster of workstations. We have compared methods that run on both partitioned and replicated databases. Both input sequences (query) allocation and database distribution strategies are shown to be important in order to improve the parallel execution of BLAST processes.

We have also evaluated a few different load balancing methods that may be used when running BLAST in parallel. These include strategies that consider the total number of sequences allocated to each node, methods that estimate the total execution time at each node and also on-demand approaches, that distribute tasks (queries and data) to nodes whenever these become idle [4].

Due to the sensibility of each method with respect to multiple parameters, different techniques may be applied in order to deal with data skew. In particular, dynamic issues like *similarity skew*, which cannot be detected before the actual execution [5], must be taken into account.

We show in Figure 1 an example of BLAST parallel execution on a 8-node homogeneous cluster, with a *pure* fragmented database configuration - distinct fragments have approximately the same size at each node. Some of the input sequences were randomly taken from the same database. One could expect a "perfect" load balancing when considering only database sizes. However, query decomposition assigns distinct input sequences to each node. As some of the sequences are more similar to the database sequences than others, the alignment process takes longer to finish. This similarity (or alignment) skew generates a clear uneven distribution of tasks.

There have been many works that focus on BLAST parallelization. MPI-BLAST [8] is widely accepted as the standard parallel BLAST tool. It copies all query sequences to each node, while the database is segmented for a demand driven delivery. For each idle node, a new database fragment is sent. MPI-BLAST developers argue that the process should use multiple fragments in order to balance the workload. However, it is not a trivial task to determine the optimal granularity for good performances. Moreover, concurrency is an important problem when nodes get fragments from the same master node.

Some other works discuss other issues. Just to mention a few, the authors in [1] propose a strategy called Dynamic BLAST. They show that one of the main problems for running BLAST in a distributed environment is related to assigning fragments with distinct sizes to each processing node. However, we have already shown that the similarity degree among is also a fundamental factor. The same question appears in [13], where the authors look forward to extend MPIBLAST for grid environments. The basic idea is that most biology research labs cannot afford to maintain efficient cluster environments. Nevertheless, the MPIBLAST tool is not that straightforward to use: it needs frequent database re-formatting when either the database is updated, or the number of nodes changes. Another solution is proposed in [11] and is not only dedicated to BLAST. There are fault-tolerant methods that could be considered as load balancing strategies. However, the goal is to guarantee completeness, not necessarily with best performances.

There have been many other proposals. Our work here brings a somewhat distinct point of view, focusing on a database-approach (database distribution design, I/O parallelism and query execution) to improve BLAST evaluation in clusters or Grids. In the next section we present our strategy for BLAST parallelization, which considers dynamic load balancing to obtain good overall performances.

3 BLAST parallelization

We consider a cluster with a number of similar nodes that store data and a master node that receives queries (and may also store data). In our approach to execute

BLAST in distributed and parallel environments, the database is fully replicated at each node of a cluster. Each copy is then physically fragmented at each node, where distinct sets of fragments, called primary copies, are mainly responsible for local BLAST processing. Figure 2 illustrates our database allocation.

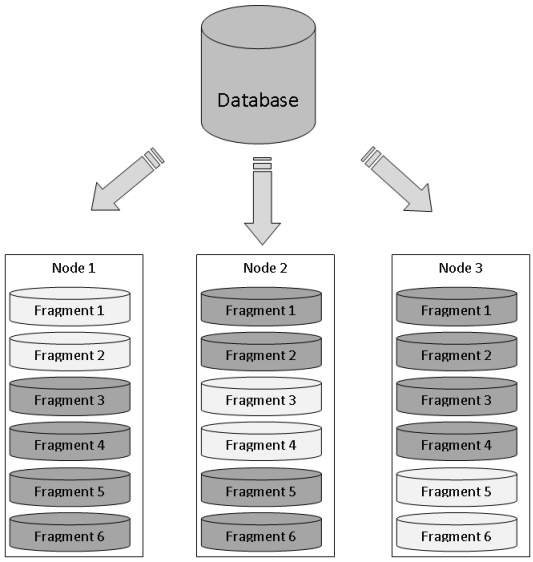

Fig. 2. Database replicated on 3 nodes, each with their own primary fragments

The main idea underlying such database assignment to processing nodes is twofold: on one hand, fragmentation enables distributed execution and is very effective when there is an even workload. When load unbalancing is detected, due to similarity skew or any other reason (e.g. a broken connection to one node), all other non-primary fragments already available at each node are then considered to achieve the complete execution. On the other hand, as the database is also replicated, it enables a correct BLAST execution regarding statistical issues and parametrization.

At least two dynamic load balancing techniques may be applied for running BLAST in a cluster. A *demand-driven* approach, where processing nodes ask for new tasks (input sequences subset) when they become idle and a *task stealing* strategy - where a node may execute part of a job originally assigned to another node. In this paper, we argue that with these rather simple yet effective strategies, we make better use of the available resources in distributed environments. Both strategies are briefly explained in Figures 3 and 4.

Basically, we strive to enjoy the advantages of both fragmented and replicated strategies. Therefore, our strategies split a database into different fragments and copy them all to each node, as illustrated in Figure 2. Each node has different fragments, and only some of them, called primary fragments, are used for BLAST

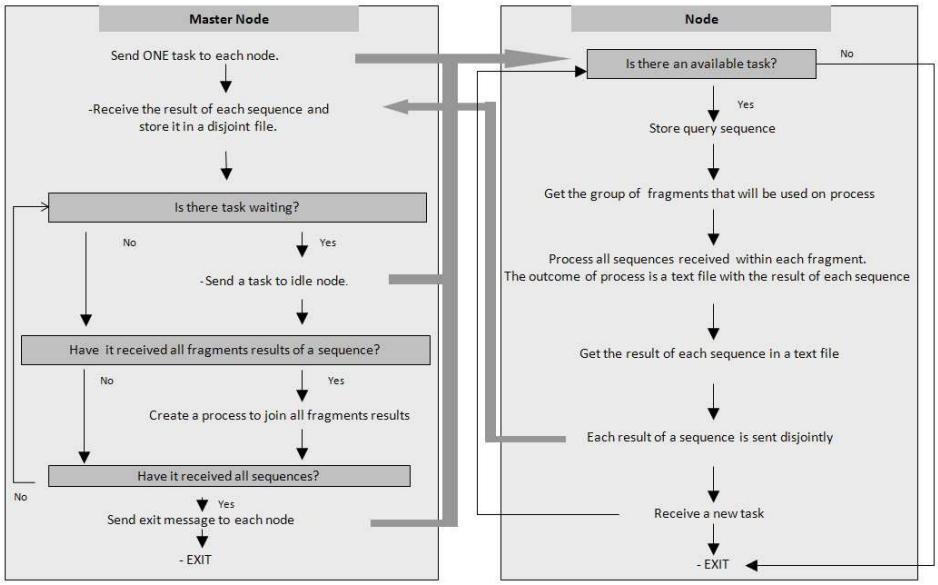

Fig. 3. Demand-driven load balancing strategy for BLAST

processing. All other non-primary fragments at a given node are only used when load balancing methods are triggered. The physical fragmentation method may vary. We have mostly used a fragmentation process that aims at obtaining fragments with approximately the same size. As we have already shown that equal database sizes are not sufficient for an even distribution of parallel tasks, we could have considered as well a simple round-robin database distribution [6].

4 Preliminary Experimental Results

We have compared our strategies to several others, like MPI-BLAST [8] and a serial execution, to assess speedup. Figure 5) that both approaches (demand-driven and task-stealing) yield much better performance when compared to MPI-BLAST, the most popular parallel BLAST tool. Note that Figure 5 shows a comparison with MPI-BLAST (version 1.4) using 96 database fragments. Even though we have pushed all fragments into each node before actual execution, in order to reduce communication costs between the nodes and the master node, our strategies still outperform MPI-BLAST.

Our workload balancing strategies, *demand driven* and *task stealing*, initially split both the database and query sequences in order to process the entire database. Each strategy has some parameters that the user can initially set. These parameters are very important as they allow the strategy to apply in different environments such as grids. For these particular tests shown in Figure 5,

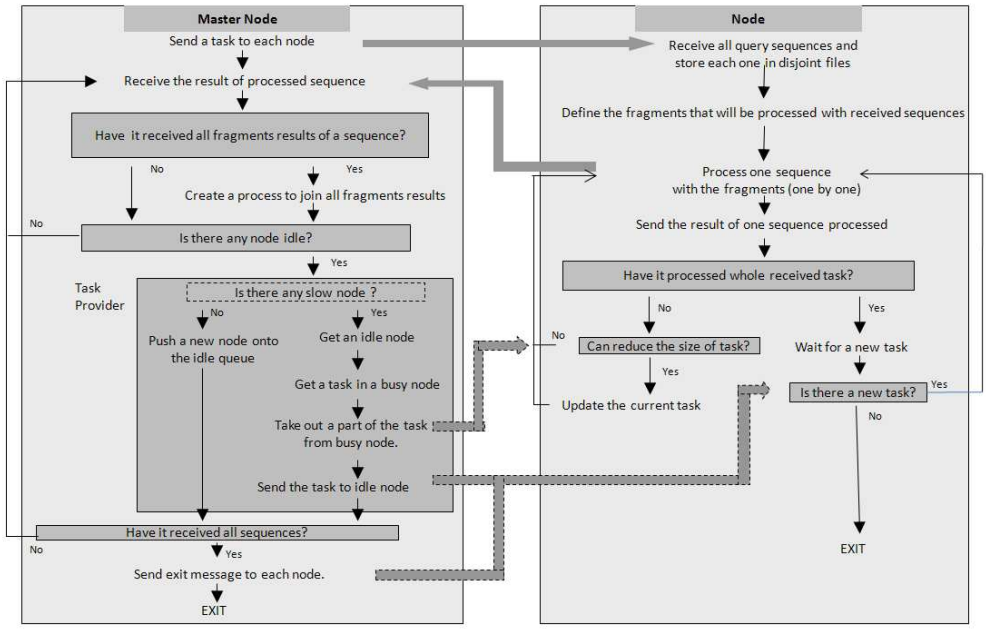

Fig. 4. Task stealing load balancing strategy for BLAST

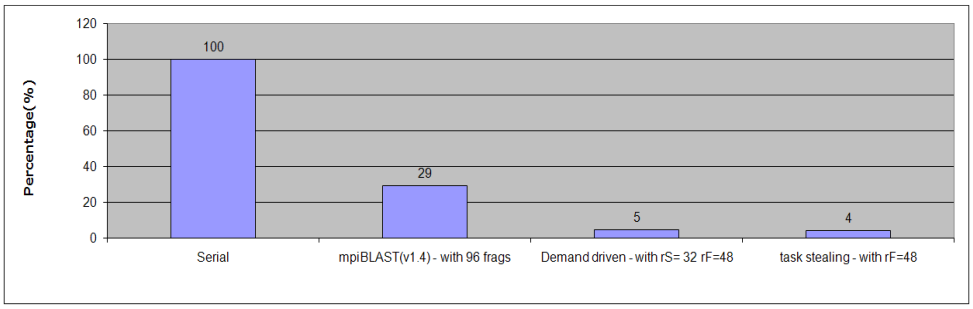

Fig. 5. Comparison among strategies

the demand driven strategy sends runs 32 query sequences each time and the execution uses a fragmented database with 48 parts. Many other experimental results, mostly for cluster-based environments, can be found in [6].

Figure 6 and 7 show the effectiveness of our approaches for load balancing. Our performance results show that our dynamic strategies to balance the workload do not become an overhead for executing BLAST in parallel.

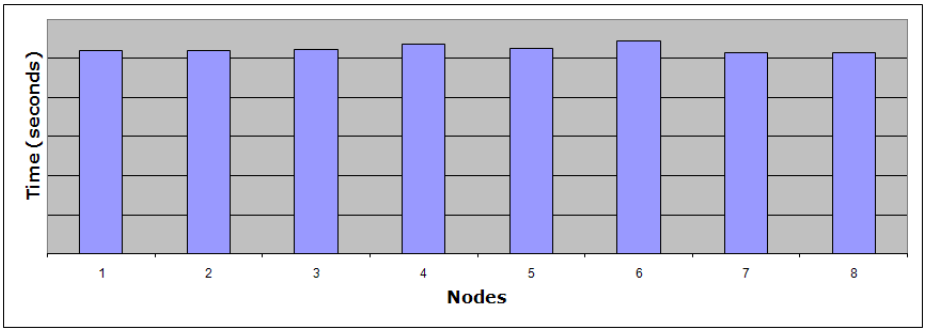

Fig. 6. Demand-driven Strategy and Load Balancing

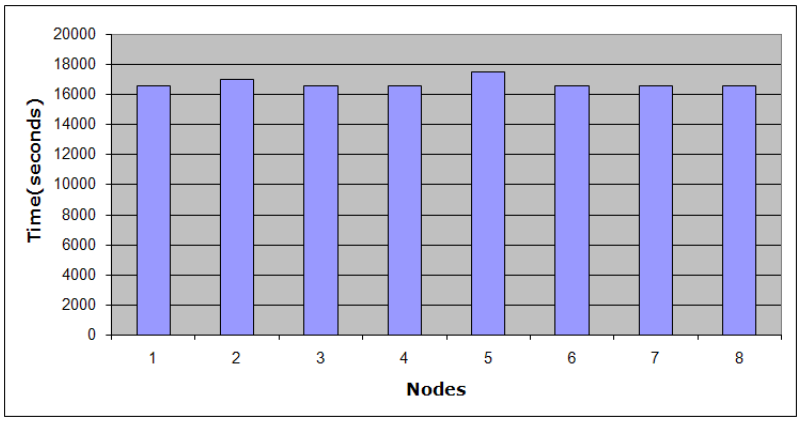

Fig. 7. Task-stealing Strategy and Load Balancing

5 Conclusion

In this paper, we propose a new parallelization approach to execute BLAST in distributed and parallel environments in order to increase performance. We use a replicated allocation of the (sequences) database, where each copy is physically fragmented. The main idea underlying replicated databases with primary fragments is that the workload may be initially distributed and, in case of load unbalancing, e.g. due to unavailable nodes, two load balancing methods should be considered. Our preliminary experimental results based on a cluster indicate that our approach achieves both very good speedup and good load balancing.

Our approach can be extended to deal in different distributed environments. In environments like the grid, cooperation for parallel execution needs consistency (of program versions and data), otherwise the results would not be reliable. A replicated database is, then, more than natural, since genomic databases are usually kept at unique repositories on the web and users may download them at every new release. However, a parallel execution could be further exploited if at

each node the BLAST program runs on only a fraction of the database - as we have done here.

An additional observation is related to the fact that we adopt a non-intrusive approach, that is, we make no changes to the specific (and usual) BLAST program used at each node. Indeed, we adopt a database point of view, investigating query processing and database allocation issues. For Grid environments, methods that modify particular source codes should be avoided. These and other data intensive challenges within a Grid are well discussed in [10].

References

1. E. Afgan, P. Sathyanarayana and P. Bangalore, "Dynamic Task Distribution in the Grid for BLAST", Procs IEEE Intl Conference on Granular Computing, pp 554-557, 2006.
2. S.F. Altschul, W. Gish, W. Miller, E.W. Myers and D.J. Lipman: "A Basic Local Alignment Search Tool", Journal of Molecular Biology 215, pp 403-410, 1990.
3. S-N. Chen, J.J.P. Tsai, C-W. Huang, R-M. Chen and R.C.K Lin, "Using Distributed Computing Platform to Solve High Computing and Huge Data Processing Problems in Bioinformatics", Procs IEEE Intl Symposium on Bioinformatics and Bioengineering (BIBE), pp 142-148, 2004.
4. R.L.dC. Costa and S. Lifschitz, "Database Allocation Strategies for Parallel BLAST Evaluation on Clusters", Distributed and Parallel Databases 13(1), pp 99-127, 2003.
5. R.L.dC. Costa and S. Lifschitz, "Skew Handling for Parallel BLAST Processing", II Brazilian Workshop on Bioinformatics, pp 173-176, 2003.
6. D.X. de Sousa, "Workload Balancing Strategies for BLAST Parallel Evaluation on Replicated Databases and Primary Fragments", MSc Dissertation, PUC-Rio Departamento de Informatica, 85p., ftp://ftp.inf.puc-rio.br/pub/docs/theses/07_MSc_sousa.zip, 2007.
7. D.X. de Sousa and S. Lifschitz, "E-value Evaluation for BLAST Parallel Execution on Fragmented Databases", Tecnical Report MCC 17/07, PUC-Rio Departamento de Informatica, 16p., ftp://ftp.inf.pucrio.br/pub/docs/techreports/07_17_sousa.pdf, 2007.
8. mpiBLAST, http://www.mpiblast.org/
9. NCBI-BLAST, http://www.ncbi.nlm.nih.gov/BLAST
10. E. Pacitti, P. Valduriez and M. Mattoso, "Grid Data Management: Open Problems and New Issues", Journal of Grid Computing 5, pp 273-281, 2007.
11. Y. Sun, S. Zhao, H. Yu, G. Gao and J. Luo, "ABCGrid: Application for Bioinformatics Computing Grid", Bioinformatics (Applications Note) 23(9), pp 1175-1177, 2007.
12. WU-BLAST, http://blast.wustledu/
13. C-T. Yang, T-F. Han and H-C. Kan, "G-BLAST: a Grid-Based Solution for mpiBLAST on Computational Grids", Procs IEEE TENCON 2007, pp 1-5, 2007.