Dynamic BLAST – a Grid Enabled BLAST

Enis Afgan[†] and Purushotham Bangalore[†],

[†]University of Alabama at Birmingham, Dept. of Computer and Information Sciences,
1300 University Blvd., AL 35294, USA

Summary
Basic Local Alignment Search Tool (BLAST) is a heavily used bioinformatics application that has gotten significant attention from the high performance computing community. The authors have taken BLAST execution a step further and enabled it to execute on grid resources. Adapting BLAST to execute on the grid brings up concerns regarding grid resource heterogeneity, which inevitably cause difficulty with application availability, fault tolerance, interoperability, and variability in performance of individual segments that are being distributed across grid resources. This paper describes a BLAST-specific metascheduler, named Dynamic BLAST, a multithreaded, master-worker type application that handles all aspects of a BLAST job submission on the grid for the user. The main contribution realized with Dynamic BLAST is minimization of user job turnaround time through understanding and leveraging of resource heterogeneity found across grid computing environments. Experiments with Dynamic BLAST on UABgrid resources show reduction in total execution time of BLAST jobs up to 50% while improving resource utilization by approximately 40%.

Key words:
Grid computing, scheduling, load balancing, BLAST.

1. Introduction

Grid computing [1] can be described as being the culmination of distributed computing and high-performance computing where networking, communication, computation and information are integrated in order to provide a virtual platform for unlimited compute power and data management [2]. Realizing benefits enabled through grid computing (*e.g.,* shorter job runtime, increased resource utilization, improved and seamless collaboration, easier information and resource sharing) is typically done by joining a Virtual Organization (VO) [3] and increasing utilization of readily available resources through more continuous use. VOs represent aggregations of communities that may share national and international boundaries but have common objectives. Through these VOs, grid computing is capable of aggregating heterogeneous resources that belong to different administrative domains and thus create a unique and valuable resource for the collaborating community.

Once a VO exists, applications need to be deployed on available resources. Grid application deployment is a non-trivial task and it includes transfer of the entire application (*i.e.,* source code, dataset, scripts) to a remote site, compiling the application on a remote host, and making it available for execution. Even before an application is deployed and can realize desired goals, it must be grid enabled [4]. *Grid-enabling* is the process through which an application that is currently executing on a standalone computer (examples of standalone resource being any of the following: single CPU workstation, server, cluster, storage unit, task specific machine with a network interface, etc.) is reorganized and/or refactored so that the application can execute on any given set of resources that are dynamically discovered. Today however, most of the applications are developed for standalone resources and are thus not in a position to execute on the grid.

As one can imagine, grid-enabling an application often requires significant effort. The amount of effort required is greatly influenced by the application that is being grid-enabled. Some application categories are more suitable for grid-enablement than others, mainly because of the parallelization methods and modes employed by those applications. Based on application communication patterns and thus the parallelization model, applications can be divided into in the following general categories [5]:

1. Sequential applications
2. Parametric Sweep applications
3. Master-Worker applications
4. All-Worker applications
5. Loosely coupled parallel applications
6. Tightly coupled parallel applications
7. Workflow applications

Applications belonging to categories 1 through 4 are, in decreasing order, the easiest to grid-enable, while applications belonging to remaining application categories may require rewriting of the complete application before it can be adopted to execute on the grid (some efforts [6] are under way to remove some of the involved difficulties).

After gaining access to a grid environment through a VO and grid-enabling an application, users immediately expect an increase in performance of application jobs that is proportional to the number of grid resources available. However, vast but raw grid resource availability does not

Manuscript received April 5, 2009
Manuscript revised April 20, 2009

necessarily warrant increased application performance. An example for a specific application, highlighting the effects an approach and methodology of consuming available resources can have on performance of an application job, will be shown in this paper. This can be contributed to the high degree of variability individual applications impose on underlying resources in terms of their requirements making generalized or standardized approaches fall short of realizing set goals. Because of such variability, general-purpose metaschedulers (*e.g,* [7]) do no focus on understanding application-specific requirements and thus cannot realize application-specific job scheduling and job performance. In order to realize such behavior, application-specific schedulers are needed that focus on specific needs of an individual application [8] [9].

With such multiple factors contributing to the difficulty of not only executing applications across grid resources, but also executing an application efficiently, it can be concluded that a typical grid user will be hard-pressed to realize desired behavior for their jobs. In order to alleviate a user form these low-level, infrastructure details, help of specialized, higher-end tools that mange grid infrastructure details on user's behalf is needed.

To that extent, this paper presents design and implementation details of a widely spread bioinformatics application, namely Basic Local Alignment Search Tool (BLAST) [10]. BLAST is a sequence analysis tool that performs similarity searches between a short query sequence and a large database of infrequently changing information such as DNA and amino acid sequences. With the rapid development of sequencing technology of large genomes for several species, the sequence databases have been growing at exponential rates [11, 12]. Facing rapidly expanding target databases and more complex search queries, the BLAST programs take significant time to find a match. Initially, parallel computing techniques have helped BLAST to gain speedup on searches by distributing searching jobs over a cluster of computers. Several parallel BLAST search tools have been demonstrated to be effective at improving BLAST's performance. mpiBLAST [13] and TurboBLAST [14] use database segmentation to distribute a portion of the sequence database to each cluster node. In the database segmentation method, each cluster node only needs to search a query against its portion of the sequence database. Alternatively, query segmentation can be applied to alleviate the burden of searching jobs. In the query segmentation method, a subset of queries, instead of the database, is distributed to each cluster node, which has access to the whole database. Figure 1 points at the differences and the execution modes of the two BLAST parallelization models.

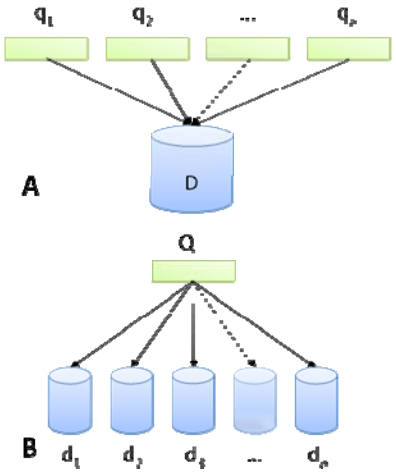

Figure 1. Two models for parallelizing BLAST: (A) query segmentation and (B) database segmentation

Because of the physical limitations encountered when parallelizing BLAST, using only traditional high performance computing technologies and techniques, a logical next step was to grid-enable BLAST. The BLAST application has a broad community whose grid-enablement was clearly going to bring significant benefit to the scientific community (in terms of resource availability as well as job turnaround time). This effort is described in this paper by providing details on developing Dynamic BLAST – a grid-enabled BLAST. As such, there are three key contributions of this work:

1. Enabling BLAST application to execute across grid environments through a simple user interface, thus hiding all complexities of underling infrastructure
2. Developing a BLAST execution model for heterogeneous grid environments, which analytically captures dependencies between an application and a resource
3. Incorporating a BLAST-specific metascheduler into the execution model that implements the BLAST execution model and is capable of understanding and leveraging heterogeneity of compute resources found across a grid to minimize BLAST job runtime

Other similar efforts of grid-enabling BLAST have been undertaken by various researchers [15] [16] but the one described in this paper is the only one that has made extensive use of available grid standards and other available tools to modularize the code as well as incorporate a BLAST specific scheduler directly into the job submission process, thus enabling the application to exploit the dependencies that exist between individual resources and the application to optimize job performance.

The remainder of the paper is organized as follows: Section 2 provides an overview of the architecture,

BLAST execution model and implementation details of Dynamic BLAST. Results of executing Dynamic BLAST and comparing it to basic query segmentation parallelization model are provided in Section 3. Section 4 suggests some future work and summarizes the paper.

2. Implementation Details

This section provides an overview of Dynamic BLAST architectural features ranging from high level technologies employed to the lower level, implementation details.

2.1 High Level Architecture

Implementation of Dynamic BLAST was done with maximum flexibility and high end-user applicability in mind. This was realized through adoption of standards as soon as those became available as well as modularization of the code to separate isolated functionality into easy-to-update modules. In particular, Dynamic BLAST was developed using Java on top of the Globus Toolkit [17] and has adopted the Distributed Resource Management Application API (DRMAA) [18] standard for all job invocation operations. Furthermore, Dynamic BLAST was built on top of GridWay [19], a grid job management system, for all job submission activities. A high level overview of Dynamic BLAST's interaction with grid components is given in Figure 2.

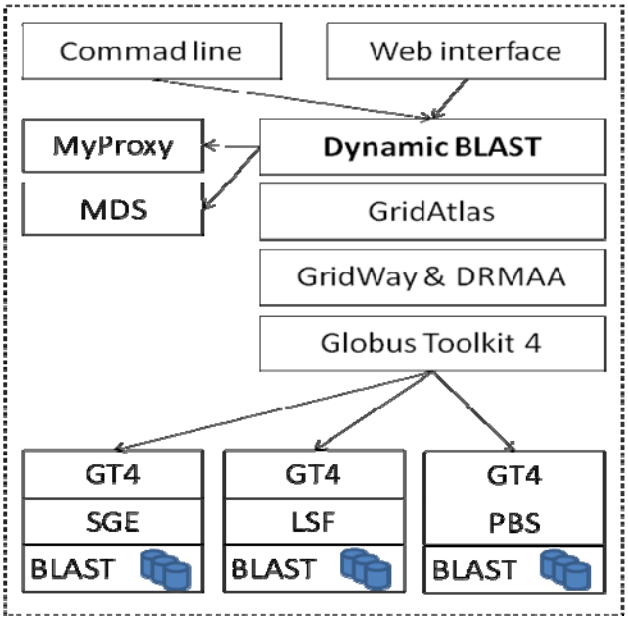

Figure 2. High level diagram of interactions between grid components and Dynamic BLAST.

As can be seen in the Figure 2, authentication and authorization are moved outside the Dynamic BLAST where the user is required to have valid X.509 GSI proxy credentials [20] before job invocation. Resources available for job submission are discovered dynamically through GIS/MDS [21]. GridAtlas [22] is a locally developed utility that monitors application related (in this case, BLAST related) parameters across various resources. This includes application installation and input data locations on various resources. Interaction with GridWay is performed through DRMAA API and is used for all job submission and monitoring activities. As discussed in the next section, Dynamic BLAST handles resource selection and data distribution but resource allocation, data transfer, and job monitoring are all delegated directly to GridWay. Adoption of GridWay in Dynamic BLAST development was a cornerstone with respect to Dynamic BLAST modularity and portability. GridWay provides a streamlined platform for high-level grid application development. Before adopting GridWay, the majority of development and maintenance effort within Dynamic BLAST was devoted to low level resource interactions and upkeep with ever-changing technologies (*e.g.,* pre-WS to WS). Adoption of DRMAA standard within GridWay has even alleviated direct dependencies of Dynamic BLAST to GridWay, thus even further increasing modularity of developed application.

2.2 Dynamic BLAST Architecture

Analysis of BLAST parallelization methods and grid resource characteristics have led Dynamic BLAST to be internally developed under the master-worker communication model, by embedding needed components into a hierarchical framework. The master-worker model allows a single process to control the resource selection, data distribution, job submission and parameterization, as well as job monitoring. Thus, selected application model maximizes execution flexibility, code modularity, and fault tolerance. Going hand in hand with the master-worker model is the choice regarding the parallelization model of BLAST jobs (*i.e.,* query segmentation or database segmentation). Suitability of one method over another is both data and resource dependent [13, 23, 24, 25]. As such, based on current resource availability, one method can often be found more appropriate than the other. In order to maximize flexibility of Dynamic BLAST in this aspect, as well and allow advanced scheduling techniques to be applied on user's behalf, the master-worker model has, once again, shown to provide the most flexibility. Selected model (*i.e.,* master-worker) allows different BLAST algorithms to be invoked on available resources even within a single job. This has the potential of increasing the suitability of available resources, maximizing resource utilization while minimizing job turnaround time.

Internal dataflow for Dynamic BLAST closely follows architectural components and consists of several key layers/steps. A diagram of the components and dataflow is provided in Figure 3. The main components of given architecture are:

- Data Analysis (analyzes input files)
- Create Job Plan (decide on resource selection, data distribution, algorithm, and job parameter selection)
- File Parsing and Fragmentation Module (based on job plan, splits the input query file)
- Thread creation (each resource and dataset is assigned a thread)
- Threads (manage all job submission related tasks for assigned resource)
- Post processing (wait on threads to complete, transfer data to local machine and join it into a single results file)

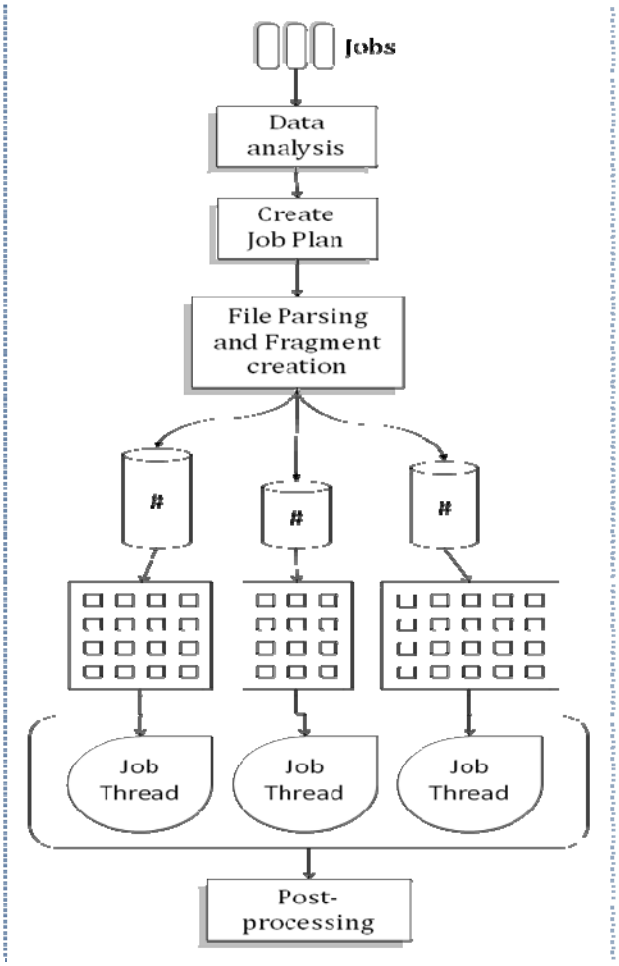

Figure 3. Internal dataflow for Dynamic BLAST.

Data Analysis performs statistical analysis of user input query file to extract some parameters needed in later steps of data processing. This information includes the number of queries, the average query length, the standard deviation, and similar. Described module also extracts current resource availability information and stores it in an internal, Dynamic BLAST specific format.

Create Job Plan module uses information retrieved by the Data Analysis module to perform on-line scheduling and job parameterization. This module is the very core of Dynamic BLAST and it implements a BLAST performance model for resource selection and data distribution. The model implemented aims at generating query fragments based on relative resource performance in order to minimize load imbalance. The model is realized by implementing Eq. (1) for resource selection and Eq. (4) for data distribution.

$$E^j \subseteq R^j \mid \begin{cases} R_i^j \in E_i^j, R_i^j \geq threshold \\ R_i^j \notin E_i^j, R_i^j < threshold \end{cases} \quad (1)$$

E^j from Eq. (1) represents the set of resources to be selected for execution of job j and it is further described through Eq. (2) and (3). In respective equations, R^j refers to the set of available resources capable of executing job j, while refers to the performance rate of individual resource from the perspective of BLAST job j. is calculated as a combination of the size of resource i, namely n_i representing the total number of processing elements (*i.e.*, cores), and the performance rate of individual processing element on resource i for the specific job j. Job performance rate is obtained from historical analysis of performance of BLAST application on a specific resource. Alternatively, a generic resource benchmark, such as SPEC [26] can be used to obtain performance value of a given resource or even the theoretical resource peak performance with a small adjustment [27].

$$R^j = \bigcup \{W_i^j\} \quad (2)$$

$$W_i^j = n_i * P_i^j \quad (3)$$

For Eq. (4), d_i represents the size of a task data chunk assigned to resource i; n_i represents the number of processing elements on resource i, D is the total size of user input and is the normalized weight or performance rate of the resource i. The result of executing the Job Plan module is a concrete plan for resource assignment and data distribution that is followed during the remainder of the job execution.

$$d_i = \frac{n_i}{\sum_{j=0}^{j<R} n_j} * D * W_i^j \quad (4)$$

File Parsing and Fragmentation module reads the generated Job Plan and proceeded in two steps. Initially, it

splits the original user query file into chunks, one for each resource, whose size is available from the Job Plan as derived from Eq. (4). Each data chunk d_i is is then further subdivided divided into fragments f_i. This is done according to equation (5), *i.e.,* proportionally to the number of processing elements (n_i) within any one resource:

$$f_i = \frac{d_i}{n_i} \quad (5)$$

Thread Creation takes place when the master thread creates worker threads; one thread is created for each resource. Individual **Threads** read their respective part of the Job Plan to parameterize given task. Because of such granular approach to job plan generation and execution, as stated earlier, different BLAST algorithms and parameters can be used for different resources. This provides needed customization and allows for maximization of resource utilization as well as very high level of user support and Quality of Service (QoS). Jobs are submitted directly by threads to individual resources through DRMAA and GridWay while the master thread waits on the threads to complete. Individual threads initiate output file transfer back to the initial job submission resource before completing their execution. If a resource fails or a task does not complete its execution as planned, the master thread could resubmit just the given task to another resource.

Post Processing module is part of the master thread and its primary task is combining all the output and result files into a single result file presented to the end user. Any cleanup and additional task, such as bookkeeping of performance results, are performed in this step as well.

3. Experimental Results

This section presents the experimental setup and performance results of Dynamic BLAST across UABgrid[1] resources. Performance comparison is performed as an iterative process starting with the basic query segmentation model for parallelizing BLAST and building the BLAST-specific metascheduler model described in the previous section to derive application-specific customizations that yield improved BLAST job performance.

3.1 Environment Setup

The experiments testing performance of Dynamic BLAST were conducted on three resources available on UABgrid. These resources are located across three independent departments, each locally administered with applicable policies and procedures in place. All of the resources had a version of BLAST application installed and required input data available for use. Technical resource details are provided in Table 1. We used the popular 1.6 GB *nr* database to search against. The *nr* database is a non-redundant protein database with entries from GenPept, Swissprot, PIR, PDF, PDB, and RefSeq. The version used was 1.6 GB in size and available from the National Center for Biotechnology Information (NCBI)[2]. The input file used consisted of 4096 search queries randomly selected from the Viral Bioinformatics Resource Center (VBRC)[3] database. The VBRC database contains the complete genomic sequences for all viral pathogens and related strains that are available for about half a dozen of virus families.

Table 1. Architectural details of resources used during experiments with AIS.

Machine	Cheaha	Ferrum	Olympus
Processor	Intel Xeon E5450	Intel Xeon E5345	Intel Xeon
Clock Frequency (GHz)	3	2.33	3.2
Instructions/Cycle	4	4	2
No. of Cores/Node	8	8	2
Memory per Node (GB)	16	12	4
Interconnect (Gbs)	10	10	1
No. of Nodes Available	12	24	64
Total No. of Cores Available	96	192	128

3.2 Performance Results and Analysis

Performance of Dynamic BLAST is compared to the performance of the plain query segmentation variant of BLAST job parallelization. The query segmentation variant operates under the model of dividing the total number of input queries across individual resources based on those resources' relative size. As such, resource weight factor, , from Eq. (4) is uniformly set to 1 and derived equation can thus be used to derive proportional amount of data that should be assigned to each resource. Given

[1] http://uabgrid.uab.edu/

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

[3] http://www.biovirus.org/

resource availability from Table 1 and using Eq. (4), data distribution shown in Figure 4 is obtained.

Having obtained job parameters for the basic query distribution parallelization model, we executed the job across available resources and obtained runtime data for the basic query segmentation model, as shown in Figure 5. Under the master-worker parallelization model employed, the overall job is considered complete only after the longest running task has completed. As such, the aim is to minimize load imbalance across individual resources; however, in obtained results, a considerable level of load imbalance across resources is exhibited.

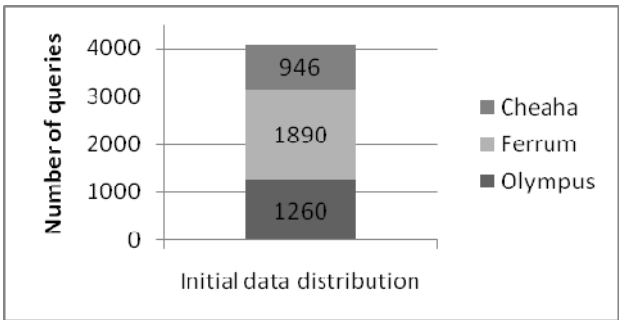

Figure 4. Initial input data distribution based on the size of resources alone

Exhibited performance and associated load imbalance can be explained by two properties. The first is a property of BLAST algorithm where, as shown in [28], runtime of BLAST algorithm is significantly affected by the length of the input query as opposed to the number of queries alone. By simply taking obtained input file provided and dividing it into a number of chunks at predetermined data points (*e.g.,* using UNIX *split* utility), the type of data that gets assigned to individual nodes within a resource can vary greatly and can thus result in the load imbalance problem. Figure 6 shows a profile view of query lengths that were used during performed experiments. As is evident from the (A) portion of the figure, lengths of individual queries vary greatly and are unevenly spread across the provided input file. When Eq. (4) is applied to this input data set, corresponding to the three available resources, three well defined data chunks are created; each of those data chunks is relative to the resource size. Furthermore, each of those data chunks is further divided among available nodes on a particular resource (according to Eq. (5)). As can be seen from the figure, a disproportionate type of queries (in terms of length) is assigned to individual nodes resulting in observed load imbalance.

Dynamic BLAST addresses this issue by reorganizing the input data so that a proportional number of short, medium and long queries are assigned to each individual node. This problem can be generalized into a bin-packing problem where the number and size of bins is predetermined (*i.e.,* number of chunks and number of queries assigned to each individual resource). A simple yet effective and efficient heuristic implementation for this problem is the first fit decreasing algorithm (complexity is $\Theta(n \log n)$ where n is the number of queries), which assigns data elements across individual bins in a decreasing order for as long as there is input [29]. Note that under the constraints of the heterogeneous and distributed environment where this algorithm is applied, a need for an optimal solution is minimal and, instead, focus should be put on efficiency. Therefore, the File Parsing module of Dynamic BLAST implements the first fit decreasing algorithm as a two-step progress: first, input file is divided into chunks of proportional type of data as dictated by the Job Plan, and secondly, each chunk is divided into a number of proportional fragments, as indicated in the Job Plan again. The result is that load balance among tasks will be achieved and thus resource comparison used to derive the job plan and the data distribution will actually hold during runtime (see Figure 7 under first-fit decreasing data re-distribution). By applying the first-fit decreasing algorithm to reorganize assignment of individual queries to corresponding nodes, data distribution shown in Figure 6 (B) can be obtained, showing a much more even distribution of comparative queries across individual compute nodes.

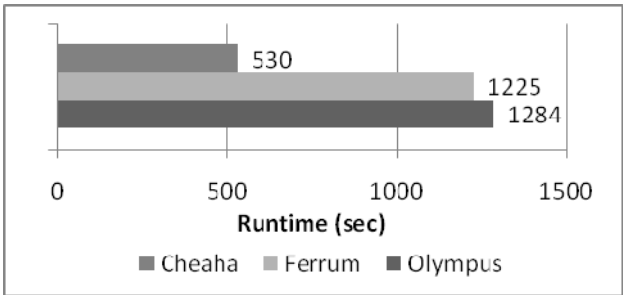

Figure 5. Runtime results across the three resources for the basic query segmentation parallelization model.

The final BLAST-specific contribution implemented as part of Dynamic BLAST, and also the second property affecting performance of BLAST jobs, is the resource benchmark for the BLAST application. This is implemented as the resource weight factor in Eq. (4) and it is realized by performing an application-specific benchmark on a given resource. The benchmark can be explicitly executed when a resource joins a largely static resource pool, alternatively, a short running benchmark can be executed prior to the submission of the real job, or information about historical runs of BLAST across resources can be kept in a local repository and then a resource-specific look can be performed prior to job submission. Current implementation of Dynamic BLAST

relies on availability of such application-specific information from Application Performance Database (AppDB) from the Application Information Services (AIS) [30] where runtime characteristics of previous application executions are stored and made available for extraction and analysis.

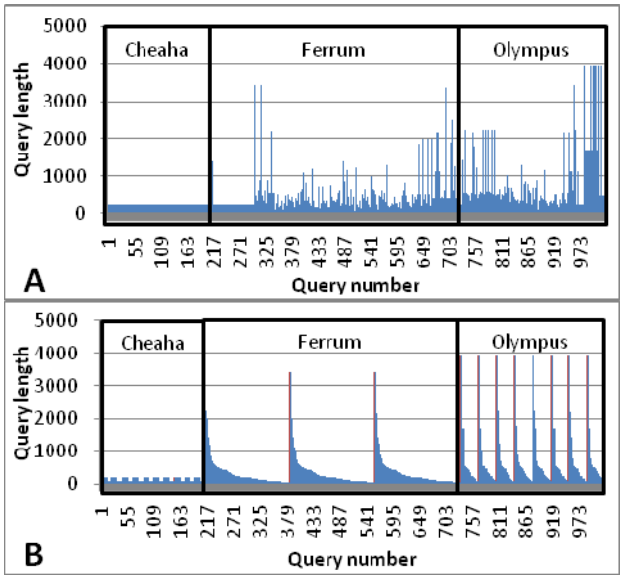

Figure 6. Difference in query distribution between (A) basic query segmentation model and (B) Dynamic BLAST data distribution model

For presented experiments, the benchmark data we used was the entire 4096 query input file otherwise divided across multiple resources. Benchmarked version was reordered by the first-fit decreasing algorithm prior to being used to provide even workload distribution to all processing elements within a resource. Although such a large benchmark is typically not necessary, we used it in presented scenario to show the relationship and impact of using any single resource versus a combination of several grid resources. Obtained runtime results, calculated normalized resource performance and newly derived data distribution based on Eq. (4) that incorporates resource weight factor is provided in Table 2.

Table 2 – Resource benchmark values, calculated resource weights and newly derived job data distribution

Resource	Cheaha	Ferrum	Olympus
Benchmark Runtime Data (sec)	1595	1025	2234
Normalized Resource Performance	0.643	1.00	0.458
Query Distribution	1255	1949	892

Final runtime results of BLAST-specific optimizations implemented as part of Dynamic BLAST are shown in Figure 7. Obtained performance results are shown along with the runtime performance characteristics of the basic query segmentation model and the data re-distribution model resulting from the application of the first-fit decreasing algorithm. As stated earlier, for the master-worker paradigm, the runtime of the overall job is determined by the longest running component (*i.e.*, Maximum). As such, results obtained in Figure 7 show a reduction in runtime of using the BLAST-specific data re-distribution model on the order of 50%.

Incorporating the BLAST-specific resource weight factor into the data distribution model, and thus implementing the Dynamic BLAST algorithm, results in additional 15% reduction of the overall job runtime. Furthermore, the overall load imbalance across multiple resources has been significantly reduced; standard deviation for the basic query segmentation model is 427 seconds while the standard deviation obtained by Dynamic BLAST is approximately 25 seconds showing the ability of devised BLAST-specific optimizations not only to cope with the heterogeneity of grid resources but also to leverage present heterogeneity.

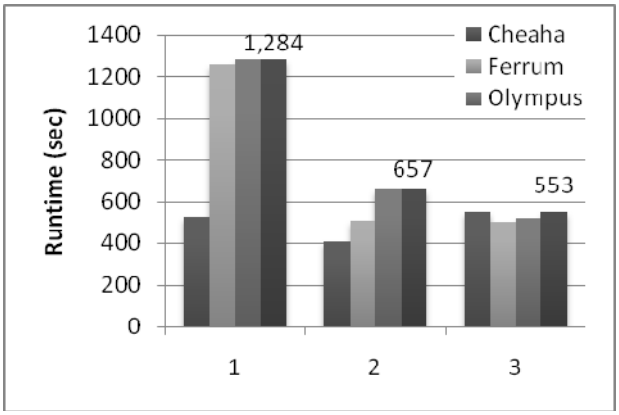

Figure 7 - Runtime characteristics of (1) basic query segmentation parallelization (this is the same data as what was shown in Figure 5), (2) first-fit decreasing data re-distribution, and (3) Dynamic BLAST (weighted first-fit decreasing). Maximum refers to the max runtime of any one resource and thus the overall runtime of a job.

3.3 Discussion

In this section, we present a brief discussion on the importance of the BLAST-specific, and more generally, application-specific optimization in the context of simultaneous distribution and submission of a job across multiple grid resources. As stated in Section 3.2, the selected input data for the BLAST benchmark unnecessarily consisted of the entire 4096 input queries.

This was done with the intention to highlight the impact and difficulty a typical grid user may experience when submitting a job to grid resources. Analyzing performance of individual resources that performed BLAST search on selected input file, as presented in Table 2, point at a great level of resource heterogeneity present across grid resources. If performance of the basic query distribution model shown in Figure 7 is compared to the performance of Ferrum resource presented in Table 2, it can be observed that the performance of the single resource outperforms the combination of three resources by approximately 20%. As such, it is easy to envision a typical user that is presented with an opportunity to execute their job across multiple resources to do just that: consume all of the available resources and at the same time realize subpar job performance. From the computer science perspective, it is therefore important to not only realize potential deficiencies in existing tools and infrastructures but also to enable users of available technologies to overcome potential hurdles without having to delve over low-level, accidental complexities introduced by the infrastructure. Dynamic BLAST provides a solution in that direction for one specific application and realizes a significantly improved experience for the end user without requiring manual user intervention.

4. Conclusions and Future Work

Development of end user applications is critical for success of grid computing. More so, those applications must execute efficiently and effectively. This paper presents the details regarding architecture, implementation and performance results of Dynamic BLAST, a true grid-enabled application. Dynamic BLAST is a powerful tool used to perform BLAST searches on widely available grid resources. The continuous goals of Dynamic BLAST are to extract potential capabilities from available grid resources as well as offer higher QoS to the users. Described application achieves set goals by exploiting BLAST-specific characteristics to better meet job requirements to resource capabilities, resulting in performance improvements exceeding 50%. At the same time, given approach enables efficient execution of BLAST jobs across general grid resources in a simple fashion resulting in significantly easier access to otherwise individual and heterogeneous grid resources that a user may have to deal with.

In conclusion, it can be stated that Dynamic BLAST represents an implementation of the query segmentation BLAST parallelization model that far supersedes performance of the basic query segmentation model. This achievement is possible due to the specific ties that have been made to understand and leverage BLAST execution characteristics across heterogeneous resources. Based on obtained experience and as part of future work, we plan on extending derived functionality into a general framework that would enable easier development of application-specific grid wrappers or applications where benefits observed in case of Dynamic BLAST can be easily realized.

Acknowledgments

This work was made possible in part by funding from the Department of Computer and Information Sciences at the University of Alabama at Birmingham, a grant of high performance computing resources from the CIS at UAB, the NSF Award CNS-0420614 and NIH/NIAID Contract No. HHSN266200400036C – Viral Bioinformatics Resource Center (VBRC).

References

[1] *The Grid: Blueprint for a New Computing Infrastructure*: Morgan Kaufmann Publishers, 1998.

[2] F. Berman, G. Fox, and T. Hey, "The Grid: past, present, future," in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds., Hoboken, NJ: John Wiley & Sons Inc., 2003, pp. 9-51.

[3] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid," *Lecture Notes in Computer Science,* 2150(2001, pp. 1-28.

[4] C. Mateos, A. Zunino, and M. Campo, "A survey on approaches to gridification," *Software—Practice & Experience,* 38(5), April 2008, pp. 523-556.

[5] E. Afgan, P. Bangalore, and J. Gray, "A Domain-Specific Language for Describing Grid Applications," in *Designing Software-Intensive Systems: Methods and Principles*, P. F. Tiako, Ed., 2007.

[6] M. Halappanavar, J.-P. Robinson, E. Afgan, M. F. Yafchak, and P. Bangalore, "A Common Application Platform for the SURAgrid (CAP)," in *Workshop on Grid-Enabling Applications, Mardi Gras Conference 2008*, New Orleans, LA, 2007.

[7] A. Kertesz and P. Kacsuk, " A Taxonomy of Grid Resource Brokers," in *Distributed and Parallel Systems from Cluster to Grid Computing*, 1 ed, P. Kacsuk, T. Fahringer, and Z. Németh, Eds.: Springer, 2007, pp. 201-210.

[8] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks," in *Supercomputing '96*, Pittsburgh, PA, 1996, p. 28.

[9] H. Dail, F. Berman, and H. Casanova, "A Decoupled Scheduling Approach for Grid Application Development Environments," *Journal of Parallel and Distributed Computing,* 63(5), May 2003, pp. 505-524.

[10] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Mol Biol,* 215(3), 1990, pp. 403-410.

[11] B. Bergeron, *Bioinformatics Computing* Upper Saddle River, New Jersey: Prentice Hall PTR, 2002.

[12] NCBI, "GenBank Statistics," February 3, 2009, Available at http://www.ncbi.nlm.nih.gov/Genbank/ genbankstats.html, Retrieved: March 17, 2009.
[13] A. E. Darling, L. Carey, and W.-c. Feng, "The Design, Implementation, and Evaluation of mpiBLAST," San Jose, CA, 2003.
[14] R. D. Bjomson, A. H. Sherman, S. B. Weston, N. Willard, and J. Wing, "TurboBLAST: A Parallel Implementation of BLAST Built on the TurboHub," Ft. Lauderdale, FL, 2002.
[15] A. Krishnan, "GridBLAST: a Globus-based high-throughput implementation of BLAST in a Grid computing framework," *Concurrency and Computation: Practice and Experience*, 17(13), November 2005, pp. 1607–1623.
[16] D. Sulakhe, A. Rodriguez, M. D'Souza, M. Wilde, V. Nefedova, I. Foster, and N. Maltsev, "GNARE: An Environment for Grid-Based High-Throughput Genome Analysis," Cardiff, UK, 2005 pp. 455 - 462.
[17] I. Foster and C. Kesselman, "The Globus toolkit," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds., San Francisco, California: Morgan Kaufmann, 1999, pp. 259--278.
[18] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardiner, A. Haas, B. Nitzberg, and J. Tollefsrud, "Distributed Resource Management Application API (DRMAA) Specification 1.0 GFD-R-P.022," Global Grid Forum (GGF) 2004.
[19] E. Huedo, R. S. Montero, and I. M. Llorente, "A Framework for Adaptive Execution on Grids," *Journal of Software - Practice and Experience,* 34(2004, pp. 631-651.
[20] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *5th ACM Conference on Computer and Communication Security Conference*, San Francisco, CA, 1998, pp. 83-92.
[21] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," in *10 th IEEE Symp. On High Performance Distributed Computing (HPDC)*, Los Alamitos, CA, 2001, pp. 181-195.
[22] E. Afgan, P. Bangalore, and S. Mukkai, "GridAtlas," December 18, 2008, Available at http://www.cis.uab.edu/ccl/index.php/GridAtlas, Retrieved: February 1, 2009.
[23] E. Afgan and P. Bangalore, "Performance Characterization of BLAST for the Grid," Boston, MA, 2007.
[24] C. Wang and E. J. Lefkowitz, "SS-Wrapper: a package of wrapper applications for similarity searches on Linux clusters," *BMC Bioinformatics,* 5(171), 2004,
[25] C. Dwan, "Bioinformatics Benchmarks on the Dual Core Intel Xeon Processor," The BioTeam, Inc., Cambridge, MA 2006.
[26] "Standard Performance Evaluation Corporation," March 10, 2009, Available at http://www.spec.org/, Retrieved: March 19, 2009.
[27] F. Sanchez, E. Salami, A. Ramirez, and M. Valero, "Performance Analysis of Sequence Alignment Applications," in *2006 IEEE International Symposium on Workload Characterization*, San Jose, CA, 2006, pp. 51-60.
[28] E. Afgan and P. Bangalore, "Performance Characterization of BLAST for the Grid," in *IEEE 7th International Symposium on Bioinformatics & Bioengineering (IEEE BIBE 2007)* Boston, MA, 2007, pp. 1394-1398.
[29] J. Y.-T. Leung, Ed. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 1st ed., vol. 1: CRC Press, 2004.
[30] E. Afgan and P. Bangalore, "Assisting Efficient Job Planning and Scheduling in the Grid," in *Handbook of Research on Grid Technologies and Utility Computing: Concepts for Managing Large-Scale Applications*, E. Udoh and F. Z. Wang, Eds.: IGI Global, 2009.

Enis Afgan is currently a Ph.D. candidate in the Department of Computer and Information Sciences at the University of Alabama at Birmingham, under the supervision of Dr. Purushotham Bangalore. His research interests focus around Grid Computing with the emphasis on user-level scheduling in heterogeneous environments. His other interests include distributed computing, optimization methods, and performance modeling. He received his B.S. degree in computer science from the University of Alabama at Birmingham in 2003.

Dr. Purushotham Bangalore is an Assistant Professor in the Department of Computer and Information Sciences at the University of Alabama at Birmingham (UAB) and also serves as the Director of Collaborative Computing Laboratory. He has a Ph.D. in Computational Engineering from Mississippi State University where he also worked as a Research Associate at the Engineering Research Center. His area of interest includes programming environments for parallel and grid computing, scientific computing, and bioinformatics. More information about his research activities can be found at: http://www.cis.uab.edu/puri.