Performance Characterization of BLAST for the Grid

Enis Afgan, Purushotham Bangalore
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294-1170
{afgane, puri}@cis.uab.edu

Abstract

BLAST is a commonly used bioinformatics application for performing query searches and analysis of biological data. As the amount of search data increases so do job search times. As means of reducing job turnaround times, scientists are resorting to new technologies such as grid computing to obtain needed computational and storage resources. Inherent with advent of new technologies, are additional complexities that arise, forcing scientists to deal with them. Grid computing exemplifies dynamic and transient state of heterogeneous resources which become a major obstacle in realizing user desired levels of service. Many users do not realize that techniques applied in more traditional cluster environments do not simply transition into grid environment. This paper analyzes resource and application dependencies for BLAST in terms of job parameters that result in performance tradeoffs. We present a set of examples showing performance variability and point out a set of guidelines, which lead to establishing job performance tradeoffs.

1. Introduction

Basic Local Alignment Search Tool (BLAST) is a well known 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 [1, 2]. The BLAST application is used to help identify the function of a new sequence by performing the similarity search against an already known and categorized gene. In recent years, high throughput DNA sequencing techniques have been creating large sequence databases resulting in exponential growth of target databases [3], causing user searches to take a significant amount of time. Parallel computing techniques have helped BLAST to gain speedup on searches by distributing searching jobs over a cluster of computers. There are two main methodologies for parallelizing BLAST searches, namely database segmentation and query segmentation. Database segmentation methodology (employed by mpiBLAST [4] and TurboBLAST [5]) distributes a portion of the sequence database to each cluster node. Thus, each cluster node only needs to search a given query set against its portion of the sequence database. On the other side, in query segmentation (employed by [6, 7]), a subset of user submitted queries, instead of the database, is distributed to each cluster node. Each compute node has access to the whole database but performs a search faster because of the smaller number of search queries. As for as the end-user of the BLAST application is concerned, the final outcome and turnaround time is of interest and thus, a typical user does not really care which of the above techniques is used, as long as the performance is satisfactory.

Although the above mentioned techniques result in shorter application search times, they do so at a cost of increased requirements for computational power. As a way to accommodate for these requirements users are aiming at using grid computing technologies [8-10]. Grid computing [11] is a technology enabling virtualization and aggregation of distributed resources into a seamless less resource pool. Although advances in grid computing have brought unsurpassed computational capabilities to user’s fingertips and, as such, offer an excellent foundation for speeding up execution of BLAST jobs, heterogeneity of available resources introduces confusion and unsatisfactory performance as observed by the end-users.

As shown throughout the rest of this paper, there are many reasons for such performance variation and most of them can be summarized by the word parameterization. We define parameterization as an understanding and selection of parameters that can be fed to a job submission engine as requirements (or generated by such a tool on user’s behalf) that are
algorithm, input data, and resource dependent. Understanding how application and resource parameters, as well as input data, affect performance of a job can greatly influence productivity on a single resource as well as across heterogeneous resource pool. Authors of [12] elude at this idea as well, but do not elaborate on it. The purpose of this paper is to analyze possible parameters and performance variations exhibited when executing BLAST jobs using different algorithms and different resources. The goal is to answer the following question: If one is to have a choice of either executing on or purchasing a system to be primarily used for BLAST searches, which of the following would be the best choice: high clock speed, large number of CPUs, large number of cores, large main memory, or fast interconnect? In addition, we address the question of how to properly parameterize a given job for current resource availability. Through a set of examples and benchmarks, we derive at observations as to which resource components, and parameters are most influential in terms of execution time and associated resource cost. These experiments lead us to BLAST parameter analysis and principles that offer execution time tradeoffs one can follow when submitting BLAST jobs or when developing an automated BLAST submission tool. Through our earlier experiences, we have observed such performance variations and were of the opinion that many other scientists must have as well (especially non-computer savvy ones are interested more in the results than understanding of the process). In order to shine some light on the topic and help in removing some ambiguities involved with resource selection and job submission, we present this paper.

The rest of the paper is organized as follows. Section 2 introduces and explains our environment and testing procedure. Section 3 presents a list and discussion of possible causes for performance variation of BLAST jobs on resources. Section 4 provides examples and analyzes reasons for these variations. In Section 5, we provide BLAST parameter analysis and guidelines while Section 6 summarizes our findings.

2. Related work and experimental setup

In our attempt to get an understanding of BLAST parameterization, we have performed detailed collection and analysis of BLAST performance data across a wide range of resources and resource configurations. Even though various implementations of sequence alignment exist (e.g., FASTA [13], SSEARCH [14] and SCANPS [15]), we have focused our experimentation on NCBI BLAST [16], the most widely used implementation, and thus obtained results will have the widest audience benefit. In addition, because the goal of our work is applicability of results in grid environment, we have focused on the query segmentation method for parallelizing BLAST. Query segmentation is generally implemented as an embarrassingly parallel type application where a master process divides user supplied query input file into appropriate set of fragments and then distributes those to available compute nodes where a standard BLAST job is submitted. This type of parallelization model fits very well into the realm of grid computing because individual job can be submitted to independent site throughout a virtual organization. Implementation of this parallelization type is based on our earlier work [8], but here we are only focusing on query fragmentation and job submission to a set of local machines and clusters rather than using grid middleware and a true grid environment.

While we recognize database segmentation as an important approach to parallelizing BLAST searches, due to increased difficulty of adopting such tightly coupled parallel algorithms to execute on grid resources, we did not perform performance comparison and analysis of this type of parallelization. Because execution of this type of applications is inherently limited to single resources, within or not, their performance can be transitioned to grid resources by considering performance on dedicated systems and incorporating middleware overhead. For more details on implementation and performance analysis, we refer interested reader to [4, 17, 18].

During out tests, we used nr database to perform the searches. nr database is a non-redundant protein database with entries from GenPept, Swissprot, PIR, PDF, PDB, and RefSeq. Used version was 1.6 GB in size and available from NCBI. The selection of the database was based on merit that the resulting analysis would provide to local, and hopefully, global users. The input query files used originated from 1024 protein queries/sequences in FASTA format that were selected from Viral Bioinformatics Resource Center (VBRC)\(^1\) database. The VBRC database is created and maintained by Dr. Elliot Lefkowitz group at the University of Alabama at Birmingham (UAB) and contains genomic information of viruses, which are considered a bio-threat by the Center for Disease Control/NIH\(^2\). In majority of our tests, we used the entire 1024 queries as input for BLAST jobs, but in some of the more specialized test we performed, a

\(^1\) http://www.biovirus.org/

\(^2\) http://www.cdc.gov/
subset of all the available queries was selected to better meet resource capabilities.

Resources used during the experimentation were selected from a pool of available resources on UABgrid [19]. We aimed at selecting a range of resources where architectural differences were greatly obvious on one hand, and also minimized on the other hand. This allowed us to observe global relationship of BLAST on overall hardware architecture (e.g., NUMA vs. SMP) as well as understand how individual hardware and software differences affect performance (e.g., equivalent architecture with higher CPU speed or different versions of BLAST).

All the testing was done on resources described in Table 1 and each of the tests was performed several times. The timings were obtained using Linux ‘time’ command and results presented are the average of all the collected runs. While this paper presents only a subset of all collected data, the complete set of experiments can be obtained at the following address: http://www.cis.uab.edu/ccl/index.php/BLASTanalysis.

Submission of the jobs was automated through a Perl script that encompassed all the required tasks (i.e., file fragmentation, Local Resource Manager (LRM) script generation, and job submission) and is able to adapt to any selected resource. Query fragmentation module divides user input file into as equal-sized fragments (based on file size). The number of fragments created is a parameter supplied at the time of invocation and it maps to equivalent number of job submissions. If a LRM was available on the submission resource, jobs were submitted to currently available nodes and each task was assigned to a single node. If multiple CPUs were available on a single node, only one task was submitted per entire node. This was performed in order to avoid CPU and memory contention between other processes. If no LRM was available, jobs were submitted directly to the operating system while other processes on the resource were limited to the ones required by the operating system. All the resources were accessed individually using SSH rather than grid middleware because the goal was to benchmark and capture performance of BLAST jobs alone rather than combining it with middleware overhead. As such, presented results can be applied in both grid and cluster environments.

### 3. Parameter considerations

This section discusses possible BLAST job submission parameters perceived from both hardware and software sides. The discussion serves as a general listing and overview of possible causes for performance variation of BLAST jobs in heterogeneous environment. While there may be additional hardware and software considerations (e.g., low-level interrupt calls, caching, hardware details, compilers used), the goal of this discussion is to present user controllable and easily understood parameters where researchers can adopt lessons learned in their everyday routines.

From the hardware architectural features of execution resource, following parameters and selection can offer performance variation of a given BLAST job: Hardware architecture – difference between basic architectural blocks such as NUMA or SMP architectures offer variable application performance. Depending on BLAST algorithm and input data set, memory paging may be a frequent operation, which, if not handled correctly by the operating system for the given architecture, will result in drastic variation in execution speed. Thus, depending on recognized architectural features and support, selection of execution resource architecture provides basic insight and controls of execution performance.
**Number of processors/cores** – depending on number of available processors/cores on a system, correct algorithm invocation and parameter options need to be recognized. For multi-process systems, as well as systems offering partial simultaneous multi-processing (e.g., Hyper-threading), user needs to be aware of node load optimality where processors do not start competing with each other to complete the same task.

**Processor clock speed** – the trend over recent years of measuring processor performance has been toward multi-core systems where the more traditional concept of higher CPU clock speed offers higher performance has been, more or less, abandoned. This concept works well in multi-processing environments, but number crunching applications such as BLAST do not necessarily fall into this category and, depending on invocation parameters, may benefit more from a single, high clock CPU than multiple ones at lower clock rate.

**Amount of main memory** – if the size of the search database exceeds size of main memory on a resource, BLAST application is required to page to disk, which reduces performance drastically [20]. Use of the database segmentation BLAST parallelization model in this case reduces, and occasionally eliminates, the requirement to page to the disk, thus improving overall application performance. Reports using this parallelization method show super-linear speedup over hundreds of nodes [4]. Correct selection of algorithm for available architecture is another user goal.

**Cluster interconnect** – the interconnect on a cluster offers significant benefit in terms of job performance, but only to the applications that make use of it. Depending on problem type, one parallelization method may offer significantly better performance than the other, and this point can only be exaggerated by the speed at which individual nodes communicate (e.g., as shown in [12]). If the algorithm employed does not make use of interconnect for frequent communication, the user will not observe significant, if any, benefit.

**Data staging** – realizing that a particular resource offers better performance than the one locally available may be a tempting reason for transferring current jobs to the new machine. Although this may present itself as a better option, the user needs to consider the entire job lifecycle, which includes input data staging, database availability and formatting, as well as output data retrieval. Long term planning may need to be considered to realize which route should be taken.

**Cost** – whether purchasing a system or selecting from a pool of available systems, a user is inherently faced with cost associated with their job execution time, either in real currency or allocation time. Depending on the current needs of a user, they may not require minimum execution time but rather minimum cost. Through correct parameterization of their job, they are able to increase efficiency of their jobs and thus maximize the cost objective.

**Execution resource software stack** – performance of a system can drastically be influenced by availability of appropriate operating system (e.g., Linux kernel), compilers and libraries. Recognizing dependencies between job execution time and such parameters can guide both, resource owner and resource consumer to improve utilization of available compute power.

Beyond looking at the hardware of execution resource alone, BLAST application features offer a choice of parameters and values that can result in significant performance variation. Although some of these may appear to be hardware dependent, understanding performance of these parameters may guide the initial selection of hardware to execute on.

**Algorithm choice** – starting with the basic differentiation of parallelization methods down to implementation details and considering data recollection and aggregation, the user’s primary choice is which algorithm to employ. Algorithms are primarily input data dependent and certainly resource dependent. Observations need to be made to be able to adjust to the changing job and resource availability alternatives.

**Thread count** – execution of a BLAST application supports use of more than one thread for any single job invocation (using ‘–a’ option). Using this option, user queries are automatically distributed between the threads and depending on the number or processors/cores on a system, performance may be increased by orders of magnitude. Recognizing dependencies between resources and available options is a key feature in combating both, performance and cost for user jobs.

**Input data statistics** – size of the database searched against is the primary base for making decisions on performance optimization. Other data related options include size of the input file, number of submitted queries, and average size of submitted queries. Even though these may often be outside of user control, understanding performance tradeoffs may result in more appropriate formatting of the data, which results in increased performance.

Lastly, although performance of jobs with different job parameters, such as blastp vs. blastn may be affected, these tests were not performed and were not subject of this paper. These parameters are input data dependent and are imposed by the problem at hand, rather than parameter options related to job performance only.
4. BLAST performance experiments

Previous section outlined possible causes for performance variability of equivalent BLAST jobs, while this section focuses on showing possible performance range as a cause of such variations. Due to page limitation, we only show a subset of all the possible causes: performance variability caused by executing hardware and software configurations, influence of variable number of threads on job runtime and resource utilization, and influence of query input file statistics on job execution time.

![Figure 1](image1.png)

**Figure 1.** Architectural comparison of BLAST execution time across many architectures and configurations.

Figure 1 is showing runtime of a set of BLAST jobs using *nr* database, *blastp* program, and a 32-query input file across hardware architectures. Each of the used resources uses two processors, some of which are dual core while others are two single core processors (full resource details are given in Table 1). There are two very interesting points captured in this figure. The first one is the observation that the number of processors or cores available for execution offers the largest benefit in terms of runtime. This is observed within a machine as well as across machines and architectures, where each of the above machines offers nearly linear speedup. Above data was obtained by using BLAST built-in feature for multiple threads and in order to compare this functionality’s performance to the alternative option of submitting multiple, single threaded jobs with smaller input files to the same node, we have performed these additional test using Coosa machine. As seen from Figure 1, the runtime results did not show a benefit of either case.

The second point derived from data shown in Figure 1 focuses on impact of software available on a given machine rather than hardware alone. Olympus and Coosa are very similar machines in terms of hardware where the major differences, as pertinent to our test cases, are the size of main memory (4 GB vs. 2 GB, respectively) and speed of the processor (3.2 GHz vs. 3.6 GHz). Yet, the performance differences were vast where Coosa, with much less main memory and slightly faster processor, was outperforming Olympus by almost an order of magnitude. In the process of testing, we upgraded software packages available on Olympus to include new Linux kernel (2.6.9) and installed new version of BLAST (v2.2.16). The new test results were in line with those obtained on Coosa leading to conclusion that installed software package version can influence BLAST performance nearly as much as the underlying hardware. In addition, we installed an older version of BLAST (v2.2.14) on Olympus and observed a slight degradation of performance, as compared to the newest version.

![Figure 2](image2.png)

**Figure 2.** Limited Scalability of Dual Opteron system using BLAST built-in parameter option.

As stated earlier, data shown in Figure 1 was obtained using built-in BLAST parameterization values. Using this method to scale the number of processors beyond two on given machine provided additional speedup only on Dual Opteron machine, which is equipped with two dual-core processors. This data is shown in Figure 2 where jobs were executed using shown number of threads and queries. Given machine exhibited linear speedup for values of four threads, but beyond that it stagnated. From this data, it can be concluded that this approach only scales to the number of available processors on a single node.

In order to gain additional performance benefits, we conformed to query segmentation parallelization model and divided the work across many resources. Results shown in Figure 3 depict runtime of BLAST jobs executing two threads per node using 1024-query input file against *nr* database, but this time on large clusters consisting of Xeon and Opteron processors. The aim of this figure is to show the benefit and runtime relationship of dividing the input file into these smaller fragments beyond that supported through BLAST parameterization alone. The data collected
shows, again, a nearly linear speedup for multiple fragments irrespective of architecture.

As discussed earlier and shown in Figure 3, larger number of fragments results in overall shorter runtime and apparently better performance. The next question to consider is: how many queries could the input data be divided into to provide optimal performance? Could we continuously break the input file into additional fragments and lower our execution time or would the performance start to decrease due to involved overhead of creating fragments, copying the input data, starting processes, and collecting output data?

Figure 3. Comparison of runtime and number of fragments, each using two threads, across large clusters of varying architecture and CPU speed.

Figure 4 provides architecture dependent runtime data for varying number of fragments. The data represent cumulative time that took all individual fragments to complete the overall job and in turn provides insight into efficiency of fragmentation. Intel based machine (Coosa) shows continuous decrease in efficiency as the number of fragments increases leading to conclusion that even though multiple fragments provide significant benefit in terms of overall runtime, this configuration does not provide most efficient use of resource. This observation is important when trying to minimize cost for a job where a user is not concerned with execution time. On the other hand, AMD based machine (Everest) provides optimal resource utilization for values of four and eight fragments, even though the overall runtime is much longer than that of Coosa.

The final analysis component presented deals with query input file statistics and runtime dependency. Beyond optimal fragment parameterization analysis, as derived from Figure 4, information shown in Figure 5 can be used to analyze and determine influence of number of queries in an input file to job runtime. As described in the section on our testing procedure, we implemented a Perl module to divide the input file into a specified number of fragments. Given module works based on file size and it tries to distribute queries across fragments as evenly as possible with the end goal of having as many files with as little size variation as possible. We have adopted this model because jobs are obviously input data dependent. Working with available code, we found that for standard distribution of queries in the input file it works very well. Throughout our testing procedure, we recorded an average standard deviation of fragments to be only 564 bytes for 638 Kb input file (less than 1%). Based on analysis of input (fragment) file size and runtime, we observed that only significant difference in file sizes (larger by at least 10%) affect search runtime. Small variations resulting from fragment creation by the Perl script did not show a direct correlation between file size and runtime.

In order to test whether file size or number of queries has most effect on execution time, we performed additional tests where we kept the file size as even as possible but varied the number of queries in the input files. The resulting data is shown in Figure 5, where two query input files were used. We created a file with only 19 long queries with average length of 3946 characters and a file with 310 short queries of average length 92 characters, each approximately 77 Kb in size.

Figure 4. Total execution time (in seconds) on AMD Opteron and Intel Xeon based cluster across varying number of fragments, each using a single thread.

Figure 5. Comparison of runtime and number of queries/sequences in input file across threads.
The job runtime using a few long queries was very similar to that of comparable jobs resulting from automated input file generation by Perl script. Surprisingly though, job runtime of the input file consisting of many short queries resulted in 81-82% increase in performance across architectures! We contribute this to the requirement of the search algorithm to keep reiterating over the long span of characters for long queries, which is not needed with short queries.

5. Parameterization analysis

Throughout this paper, we have presented a number of BLAST parameter combinations resulting in minor as well as major influence on overall job performance. In this section, we summarize our findings and outline a set of guidelines that should be considered when submitting BLAST jobs, especially in heterogeneous environments such as the grid.

From the presented data, we can divide our observations in four categories, namely: architecture, speed and number of CPUs, input data statistics, and resource software stack. In terms of architecture, we observed a trend where the speed of the processor in SMP (i.e., Intel) based machines compared to NUMA (i.e., AMD) based machines needed to be much higher to provide comparable performance. Overall, speed of the CPU seemed to provide proportional increase in performance when compared across CPUs within a given architecture. This observation is influenced by probably the most surprising conclusion in this paper and that is versioning of resource software stack. Based on our analysis, we arrived at an observation that even small variations in versions of both, operating system and BLAST provide significant influence on job performance. Simultaneously to the speed of CPUs comes the number of CPUs or cores. Overall, this factor has the most influence on overall turnaround time of a job. Increasing number of CPUs employed provides continuously a nearly linear job speedup. Even though job efficiency degrades as the size of problem instance (i.e., file size) is reduced, if the cost is not a major constraint, adding more processors to the job will result in minimization of turnaround time. In addition, using number of threads available as BLAST parameter directly (corresponding to the number of processing cores available on the system) results in limited but efficient speedup. Lastly, if input data permits, splitting the input queries based on size of queries rather than file size, significant performance benefits can be observed. If the data was distributed using this method, resource selection could be performed to meet capabilities of individual resources aiming at achieving appropriate load-balancing.

In terms of guidelines that should be considered during job submission, we present them as a set of brief statements and questions. Job parameterization should be based on available resources and associated resource cost. Multiple resources should be analyzed rather than trying to optimize any single resource. When using multiple resources, the entire parameterization process should be completed beforehand for all available resources. User objectives should be considered (cost vs. time, resource owner vs. resource user) rather than assuming everyone’s goal is turnaround time minimization. If cost is primary objective, look at resource utilization rather than turnaround time alone. Analyze proximity of data, length and frequency of computation. Analyze input data and observe possibilities of data fragmentation – remember benefits of executing a large number of short queries vs. long queries. Based on number of available CPUs on a node, parameterize given task independently for each compute node. Some examples have shown very slight increase in performance for up to twice the number of threads as there are available processors on a given node. For large jobs, this may be worth exploring further. Consider age of software stack on submission resource – what may appear as a slower machine, with new versions of relative libraries and programs, performance may be better than expected.

In conclusion, we point at two more general observations. With the advent of grid computing, utilization of small workstations available in student labs during night hours could be considered for use with a continuous influx of jobs. Regarding resource availability and load – smaller and older resources often exhibit lighter loads and as such, even though execution time may be longer, queue wait times may be minimized or non-existent and cost is probably lower.

6. Summary and conclusions

BLAST application is one of the most widely used bioinformatics applications of today, where scientists from various disciplines employ it in their daily routines. With the ever-growing size of search databases and thus longer search times, scientists are exploring new technologies such as grid computing to accommodate for increased requirements in resource capabilities. Resources available in such environment, as well as individual resources directly available to users, are subject to performance variation due to job parameter selection. With expected increase in use of grid resources, such variabilities are exemplified due to
the heterogeneous environment. Through selection of inappropriate parameters and parameter values, resources are being underutilized and users are overpaying for received service. In order to allow users to have an understanding of performance dependencies between BLAST application and underlying hardware, in this paper, we outlined possible reasons for variability of BLAST search performance within a resource as well as across resources and architectures. This allows users to maximize utilization of resources and minimize their search times. We focus our efforts on several of the most prominent and influential parameters and, through examples, show performance penalties and tradeoffs for various architectures and algorithms. By analyzing our results, we derived at a set of principles and observations, which can help BLAST users during job submission to maximize their utility. As understanding of such parameter optimizations become more prominent and recognized, tools will emerge that will automate this process and incorporate all the components (i.e., resource selection, application dependencies for given resource, and user utility) into a single job submission interface. This can lead to automatic enablement and presentation of cost vs. time tradeoffs to the users for their jobs.

7. Acknowledgements

We thank our colleagues, namely Pavithran Sathyanarayana, on development of the Perl script for job submissions and Shankar Changayil for providing us with the search sequences and appropriate explanations and descriptions. This work was made possible in part by a grant of high performance computing resources from the Department of Computer and Information Sciences at the University of Alabama at Birmingham, and the National Science Foundation Award CNS-0420614.

8. References