Parallel Genomic Sequence-Searching on an Ad-Hoc Grid: Experiences, Lessons Learned, and Implications

Mark K. Gardner[*] Virginia Tech
Wu-chun Feng[†] Virginia Tech
Jeremy Archuleta[‡] University of Utah
Heshan Lin[§] North Carolina State University
Xiaosong Ma[¶] North Carolina State University and Oak Ridge National Laboratory

Abstract

The Basic Local Alignment Search Tool (BLAST) allows bioinformaticists to characterize an unknown sequence by comparing it against a database of known sequences. The similarity between sequences enables biologists to detect evolutionary relationships and infer biological properties of the unknown sequence.

mpiBLAST, our parallel BLAST, decreases the search time of a 300 KB query on the current NT database from over two full days to under 10 minutes on a 128-processor cluster and allows larger query files to be compared. Consequently, we propose to compare the largest query available, the entire NT database, against the largest database available, the entire NT database. The result of this comparison will provide critical information to the biology community, including insightful evolutionary, structural, and functional relationships between every sequence and family in the NT database.

Preliminary projections indicated that to complete the above task in a reasonable length of time required more processors than were available to us at a single site. Hence, we assembled GreenGene, an ad-hoc grid that was constructed "on the fly" from donated computational, network, and storage resources during last year's SC|05. GreenGene consisted of 3048 processors from machines that were distributed across the United States. This paper presents a case study of mpiBLAST on GreenGene — specifically, a pre-run characterization of the computation, the hardware and software architectural design, experimental results, and future directions.

Keywords: bioinformatics, BLAST, sequence search, grid computing, cluster computing, optical networking, scheduling, fault tolerance, scalability, agile development, scripting

[*]e-mail: mkg@vt.edu
[†]e-mail: feng@cs.vt.edu
[‡]e-mail: jsarch@cs.utah.edu
[§]e-mail: hlin2@ncsu.edu
[¶]e-mail: ma@csc.ncsu.edu

1 Motivation

The vast majority of compute cycles consumed in bioinformatics are spent on the BLAST family of sequence database-search algorithms. These algorithms search for similarities between a query sequence and a large database of nucleotide (DNA) or amino acid sequences [Altschul et al. 1990; Altschul et al. 1997]. Newly discovered sequences are commonly searched against a database of known nucleotide or amino acid sequences. Similarities between the new sequence and a sequence of known function can help identify the function of the new sequence. Other uses of BLAST searches include phylogenetic profiling and bacterial genome annotation.

Traditional approaches to sequence homology searches using BLAST have proven to be too slow to keep up with the current rate of sequence acquisition [Kent 2002]. From 1982 to 2004, the number of sequences in the NCBI GenBank has grown by a factor of 67,000 [Genbank]. Because BLAST is both computationally intensive and parallelizes well, many approaches to parallelizing its algorithms have been investigated [Braun et al. 2001; Camp et al. 1998; Chi et al. 1997; Pedretti et al. 1999; Singh et al. 1996]. Our open-source parallelization of BLAST, mpiBLAST,[1] fragments and distributes a BLAST database among cluster nodes such that each node searches a unique portion of the database. Figure 1 and Table 1 show that the latest version of mpiBLAST exhibits super-linear speedup and scales to hundreds of nodes, greatly improving on the performance of a earlier version [Darling et al. 2003]. The super-linear speedup is due to the fact that the large sequence database can

[1]See `http://www.mpiblast.org/`.

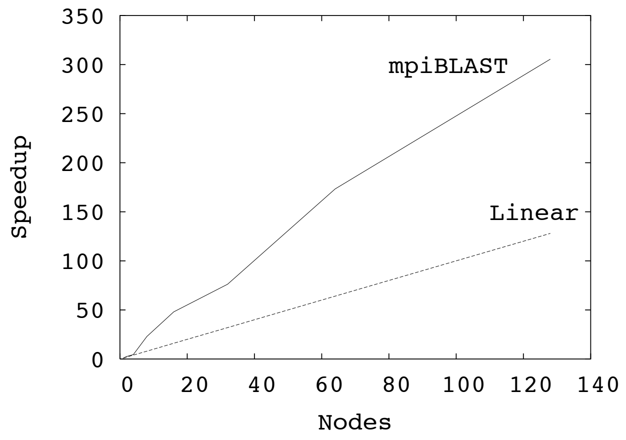

Figure 1: Speedup of mpiBLAST (Version 1.4)

	Execution	
Nodes	Time (sec)	Speedup
1	176,880	1.00
2	68,640	2.58
4	39,109	4.52
8	7,730	22.88
16	3,683	48.03
32	2,321	76.21
64	1,021	173.24
128	579	305.49

Table 1: Speedup of mpiBLAST 1.4 vs. BLAST

be segmented and fit into the aggregate memory of the nodes, therefore eliminating I/O which is orders of magnitude slower than memory accesses.

Currently, the largest database against which a sequence may be compared is the nucleotide (NT) database containing GenBank, EMB L, D, and PDB sequences. The database is an unordered ASCII flat file that is updated daily and has grown to more than 14 GB as of November 2005. When biologists need to catalog an unknown sequence, they use BLAST to search against the database. If the sequence is not found, the new sequence is added to the end of the database file, making the unordered file larger.

Because of the importance of the NT database, we propose searching each sequence in the NT database against the complete NT database itself as a service to the bioinformatics community and to stress-test mpiBLAST. By sequencing the NT database against itself, we hope to make the biological information contained in the NT database more useful to researchers by enabling a Google-like indexing structure. Such a structure could increase search-speed times by a factor of 100 while at the same time providing up to a 20-fold compression in the size of the database [Gans 2005]. Furthermore, it would also allow searches to be conducted in a nonlinear manner.

Preliminary projections indicate that to complete the above task within a reasonable amount of time requires more processors than are available to us at a single site. That is, assuming a dedicated cluster supercomputer with 128 processors that computes a result every 10 minutes on average (based upon a linear extrapolation of the execution time as a function of the number of nodes from Figure 1), matching all 3.5 million sequences in the NT database against the NT database would take over 408 days. Therefore, to make this problem more computationally tractable, we leveraged the fact that mpiBLAST is highly parallel and proposed running mpiBLAST over a grid with sufficient computational, network, and storage resources. But what kind of grid should we use?

One way of obtaining large quantities of resources for a highly parallel computation is to take the desktop grid approach. Projects, such as SETI@home [SETI@home] and Folding@home [Folding@home], have shown that desktop grids are a viable way to obtain high parallelism at low cost. Because people are willing to donate the spare cycles on their machines for good causes, the costs associated with desktop grids are often very low. However, we discounted this approach for several reasons.

First, it takes time and publicity to foster enough participation that significant resources become available. Second, the resource requirements to successfully compute some sequence matches are *much* greater than what can be expected of machines in a desktop grid. SETI@home clients (or workers) operate on fixed-size work units that are roughly 350 KB in size while mpiBLAST clients (or workers) search with variable-size work units (i.e., queries) that can range in size from a handful of bytes to as much as 80 MB and generate megabytes to gigabytes of output. In addition, a sizable chunk of the NT database must also be distributed to each mpiBLAST client before sequence search can even commence, resulting in a substantial data-movement problem that SETI@home does not have. Third, SETI@home is embarrassingly parallel; mpiBLAST is not. mpiBLAST achieves super-linear speedup by segmenting the database to fit in the aggregate memory of the nodes, thereby eliminating costly I/O accesses. However, database segmentation also increases the dependencies between parallel tasks in a query because the final results for each query must be obtained by consolidating intermediate results from many tasks. Finally, the highly skewed and unpredictable distribution in sequence-search times for queries, combined with the heterogeneity of distributed personal computers, would require high capacity in caching and processing intermediate result data, which is not easy to achieve efficiently in the @home model.

Consequently, our approach aggregates sufficient supercomputing resources (*a la* the TeraGrid) to solve the problem. Grid frameworks, most notably Globus [Foster and Kesselman 1997], have been developed to aggregate such resources into a computational grid. Indeed, a version of mpiBLAST for Globus, mpiBLAST-g2 [mpiBLAST-G2], already exists. However, we were unable to take advantage of mpiBLAST-g2 and existing grid frameworks at the time for a multitude of reasons: OS incompatibility, hierarchical scheduling, lack of a "grid"ified version of mpiBLAST-PIO (an extended version of mpiBLAST with parallel I/O capabilities), heterogeneous security across administrative domains, and time pressures. Each of these is elaborated upon below.

First, the aforementioned grid frameworks were neither available nor well-tested on Mac OS X and 64-bit Linux OS, respectively. And unfortunately, both of these operating systems made up a sizable majority of the supercomputing resources that we had available to us during the week of SC|05 in Seattle. Second, computing the NT-versus-NT alignments required a hierarchical scheduler. The bottom level of the scheduler, functionality that is available in grid toolkits, executed queries against the NT database. The top level scheduled queries across the bottom-level toolkits, collected the results, and re-scheduled queries to accommodate resource failures. None of the available grid toolkits provided this aspect of hierarchical scheduling. (However, this required capability appears to be under development at the Edinburgh Parallel Computing Centre (EPCC) [EPCC a; EPCC b] but does not appear to be released.) Third, mpiBLAST-PIO [Lin et al. 2005], our extended version of mpiBLAST with parallel I/O capabilities that dramatically improves scalability, has not yet been ported to the grid frameworks. Fourth, security and administrative concerns prevented the installation of the frameworks and the opening-up of holes in institutional firewalls. Finally, given the time pressures of building and running on an ad-hoc grid during the week of SC|05, coupled with the above issues, we decided to create our own "ad-hoc grid software" as all we required was a simple system for distributing queries, executing mpiBLAST (or more specifically, mpiBLAST-PIO), and gathering results.

Consequently, we assembled "GreenGene" — an ad-hoc grid that was effectively constructed on-the-fly from donated computational, network, and storage resources during last year's SC|05.[2] In the remainder of the paper, we will discuss the problem of sequence searching NT-against-NT in greater detail, the hardware and software architecture of GreenGene, and finally the results obtained from running the experiment during SC|05.

Group	Sequence Length (KB)	Count	Percentage
G1	0–5	3,305,170	95.66
G2	5–50	87,506	2.53
G3	50–150	25,960	0.75
G4	150–200	26,524	0.77
G5	200–500	9,592	0.28
G6	> 500	248	0.00007
Total		3,455,000	100.00

Table 2: Composition of the NT Database

2 Computing "NT-Complete"

Performing a sequence alignment search is a computationally intensive undertaking with resource usage in time and space that depend on both the input query and the database. Table 2 shows the composition of the NT database at the time we began the project in October 2005. There were approximately 3.5 million sequences in the database. Of those, 95.66% were 5 KB in size or less. Nearly all the remaining sequences were between 5 KB and 500 KB. Only a few sequences exceed 500 KB in size. Because the execution time, memory requirements, and output size are some function of the input sequence length and the database size, the project looks to be a straightforward problem. That is, given enough processors, it should be easy to divide the sequences and the database segments across the nodes and compute the results. Naturally, real life is somewhat more complicated.

The BLAST algorithm employs heuristics to decrease the computation time, as compared to the precise Smith-Waterman algorithm [Smith and Waterman 1981], but at the cost of decreased sensitivity. Yet even with the performance improvement obtained via the heuristics, along with our parallelization of BLAST, a comparison of a 300 KB query with 441 sequences (none of which are found in the NT database) to the NT database takes over an hour to complete on 16 desktop-equivalent nodes. The NT database is 50,000 times larger with 8,000 times as many sequences, and the comparison against itself will take even longer to execute because each sequence is found in the database and is guaranteed to have close matches which are expensive to compute. Furthermore, the heuristic nature

[2]GreenGene is a tongue-in-cheek reference to Green Destiny [Feng 2003], our previous energy-efficient supercomputer used to run mpiBLAST, and the system's capability for performing genomic searches.

	Sample Input		Sample Output				Complete NT-to-NT Output	
	Seq	Query		Time	Size	Time per CPU	Output	CPUs to finish
Group	Count	(KB)	CPUs	(secs)	(MB)	(sec)	(GB)	during SC|05
G1	909	1,107	32	4,496	342	133,144	1,139	1,110
G2	91	1,060	32	5,127	558	48,410	593	403
G3	2	254	64	28,154	2,163	5,450,612	23,555	45,422
G4	7	1,260	32	7,941	1,117	252,450	3,995	2,104
G5	4	1,012	64	>36,000	†	†	†	†
G6				§	§	§	§	§
Total						>5,884,615	>29,282	>49,038

† *Did not complete.* § *Unknown. Test cluster too small to run.*

Table 3: Estimating the CPU Count Needed to Finish During SC|05 and Resulting Output Size

of the BLAST algorithm also leads to a weaker correlation between the sequence length and the resources needed (e.g., CPU cycles, memory, and disk space), which makes resource prediction more difficult.

In order to *estimate* the resource requirements for the project, we randomly selected sequences from each group in Table 2 to form a query and perform the search. The results are shown in Table 3. The query in group G3 is limited to only two sequences that happen to be particularly difficult to complete given the amount of memory that they require to run to completion. Furthermore, both sequences are very similar to many sequences in the database, and hence, require a lot of computation to find all the matches. As a result, the query requires 64 processors rather than 32 and takes nearly 8 hours to complete. The sample selected for group G5 was also found to contain particularly hard sequences and did not complete within 10 hours (the maximum time of the PBS scheduler on a testbed cluster). Likewise, sufficient processors were not available until SC|05 to execute the sample runs from group G6.

The preliminary results suggest that some of the sequences in the NT database are particularly hard and will take a long time to complete. The troublesome sequences have two characteristics in common: they are larger than 5 KB, and they closely match many sequences in the database. This causes the nodes to buffer large quantities of intermediate results, much larger than the average output per node would suggest, causing a shortage of memory on the node.[3] One particularly hard sequence has a memory footprint that exceeds 1.7 GB on at least one node.

Another difficult sequence, which took 5.2 hours to complete when re-run on 90 nodes, generates 1.8 GB of output even though the input is only 122 KB in size. The RAM usage on at least one node exceeded 1.3 GB even though the average output per node is only 20.5 MB (1.8 GB / 90). Increasing the number of nodes from 64 to 90 increased the amount of buffering available for intermediate results and allowed the sample query to complete.

Why does BLAST buffer so many intermediate results, especially since it greatly increases memory requirements and prevents job completion? The buffering is necessary to sort the results according to their scores and select the N (typically 500) best matches in order to present the most useful results first. Besides consuming large amounts of memory, it is this final pass through the intermediate results that prevents BLAST from being embarrassingly parallel.

An alternative approach to estimating the execution time and output size is to exploit the fact that BLAST performs searches in two phases [Altschul et al. 1990]. In the first phase, BLAST compares the query sequence with database sequences at the word level to identify high scoring pairs (`hits`), which are word pairs of fixed length (normally 11 for DNA sequence search) with similar score beyond a certain threshold T. In the second phase, these hits are then extended to find the result matches (maximal segment pairs). Therefore, executing only the first phase can provide an estimate of the maximum number of hits without having to execute the more expensive second phase, thus making it an attractive potential predictor as it inherits some internal information from BLAST algorithm.

To test the hypothesis that the number of `hits` computed in the first phase generates better execution time and output size estimates, we collect sequence length and number of database hits as the X (or "input") variables and execution time and output file size as the Y (or "output") variables for 500 randomly selected sequences. We model these sample data with linear regression and present the resulting statistics in Table 4. The minor differences in mean square error (less than 5% for both output size and execution time prediction) suggests that using the number of database hits from

[3]BLAST already discards matches with scores below a threshold, otherwise *all* matches would need to be buffered.

	Correlation	Coefficients X Variable	Intercept	P-value	Mean Square Error
X=Length, Y=Output Size	0.628	0.713	-2.611	2.98E-56	34376702.468
X=Hits, Y=Output Size	0.643	5.72E-05	216.167	1.34E-59	33332554.787
X=Length, Y=Execution Time	0.671	0.004	2.020	8.74E-67	893.667
X=Hits, Y=Execution Time	0.691	3.29E-07	3.235	3.88E-72	850.609

Table 4: Sequence Length vs. Estimated Matches as Predictor

the first phase of the BLAST calculation does *not* produce a better predictor than does the sequence length.[4] Therefore, we choose to use the sequence length as the predictor rather than partially executing BLAST to obtain hit counts, especially since hits are not a significantly better predictor of execution time and output size.

Returning to the question of estimating the resources needed to complete the project, if all of the sequences in the database require as much effort as those in the sample from group G3, the complete NT-against-NT alignment would take around 1,561 years (3.9 hours per G3 query multiplied by 3.5 million queries), an intractable task even if the database were static. Fortunately, preliminary results indicated that many queries execute quickly and produce small amounts of output. This pattern was particularly true of queries in G1. Since the queries in G1 constituted the bulk of the NT database, there was hope that the project would complete by the end of SC|05.

3 Hardware Architecture

Because each query could be run independently of the others, low-latency communication was not required to achieve good performance. This was fortunate because the supercomputing hardware that was made available to us for this project was located, for the most part, on the west and east coasts of the United States, as shown in Figure 2.

The GreenGene ad-hoc grid has a hierarchical organization resembling a constellation architecture. Clusters at each of the four physical sites were connected to high-speed wide-area network (WAN) connections via National LambdaRail (NLR) or Internet2 via Abilene. The clusters and their sponsoring organizations are listed in Table 5. Below we provide more detail about each of the clusters and their wide-area connections to the rest of the GreenGene infrastructure. All these resources were temporarily integrated during the week of SC|05 into a functioning ad-hoc grid.

[4]Alternatively, we note that while the correlation between the X variables, i.e., sequence length and estimated number of matches is high (0.9942), and the correlation between the Y variables, i.e., execution time and output size, is also high (0.9583), the correlation between the X and Y variables ranges between 0.628 and 0.691.

Cluster	Organization	Location
System X	Virginia Tech	Blacksburg, VA
TunnelArch	Univ. of Utah	SLC, UT
LandscapeArch	Univ. of Utah	SLC, UT
DuPont	Intel	DuPont, WA
Jarrell	Intel	Seattle, WA†
BladeCenter	Intel	Seattle, WA†
Panta	Panta Systems	Seattle, WA†

† On exhibit at SC|05

Table 5: Clusters Making Up GreenGene

To facilitate the computation, Virginia Tech generously provided exclusive access to their System X supercomputer for the week of SC|05. System X is composed of 1,100 Apple Xserve G5 nodes, each with dual 2.3-GHz PowerPC 970FX CPUs, 4-GB ECC DDR400 RAM, one 80-GB SATA local hard disk, one Mellanox Cougar InfiniBand 4x host channel adapter, and Gigabit Ethernet (GigE) network interface card. Shared storage was via the traditional network file system (NFS). The ten head nodes of System X were directly connected to NLR, and hence, the bottleneck network capacity to SCinet was ten gigabits per second (10 Gb/s). The MPI implementation that we used to run mpiBLAST was MPICH-1.2.5 [Gropp et al. 1996] with modifications for InfiniBand [InfiniBand Trade Association 2000].

The University of Utah also provided dedicated access to their clusters: TunnelArch and LandscapeArch, which are both part of the Arches metacluster. The TunnelArch cluster consisted of 63 nodes containing dual AMD Opteron 240 CPUs, 4-GB ECC DDR266 RAM, and a 40-GB IDE hard disk connected with GigE. The LandscapeArch cluster consisted of 64 nodes containing dual AMD Opteron 244 CPUs, 2-GB ECC DDR333 RAM, and a 200-GB hard disk connected with GigE. In both cases, shared storage was via the Parallel Virtual File System (PVFS) [Carns et al. 2000]. The bottleneck network capacity to SCinet is one gigabit per second (1 Gb/s) . The MPI implementation on both clusters was MPICH2 [MPICH2].

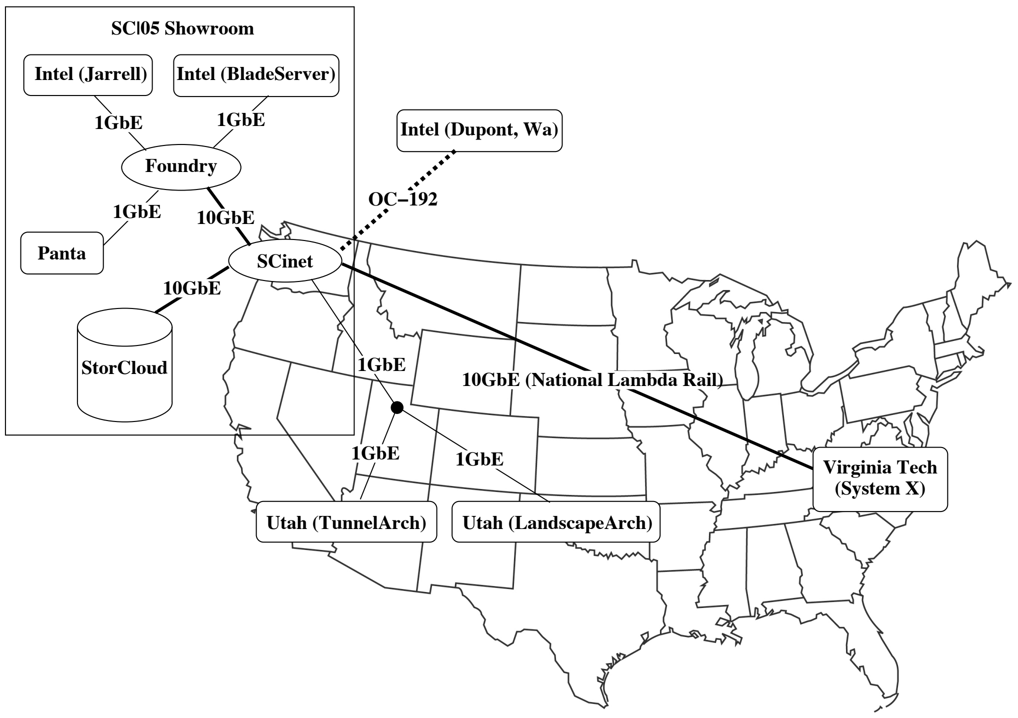

Figure 2: High-Level Architecture of the GreenGene Ad-Hoc Grid

Intel provided three clusters: the DuPont cluster, the Jarrell cluster, and the BladeCenter cluster. The DuPont cluster was a 128-node quad-core system connected with InfiniBand and Gigabit Ethernet (GigE). Only half of the system was available during the day.[5] During the night, the entire cluster was available. The Jarrell cluster contained 10 nodes each with dual Intel 3.4-GHz Pentium 4 CPUs, 2-GB RAM, 70-GB hard disk and dual GigE running RedHat Enterprise Linux 4U2 and Intel MPI v1.0. The BladeCenter cluster contained 14 BladeCenter nodes, nine with dual 3.06-GHz and five with dual 2.66-GHz single-core Intel Xeon CPUs. Each blade contained 2 GB of RAM, two 80-GB hard drives, and GigE.

The cluster provided by Panta Systems contained eight nodes, each with four AMD Opteron 246HE CPUs with 2-GB RAM interconnected with Infiniband and GigE running 64-bit SuSE Linux v9.3 with MVAPICH-0.9.6 [Liu et al. 2003a].

Global storage for the expected multi-terabyte output was provided as part of SC|05 StorCloud and consisted of 32 dual-3.2GHz Intel Xeons nodes driving the Lustre parallel file system [Luster] connected via GigE to each other and to the 10-Gb/s Ethernet SCinet backbone, and from there, to NLR. In addition, Intel provided 18.3 TB of raw RAID-5 storage in four disk arrays (two units with 16 400-GB drives and two with 12 250-GB drives), and we brought a single custom-built 1.75 TB of raw RAID-5 storage in a single disk array. The Intel storage was aggregated using Intel's storage manager product. Both storages systems were formatted with the Linux XFS file system. (As we will discuss later, this storage redundancy proved to be vitally important.)

Finally, Foundry Networks provided a SuperX 10GigE switch to connect to SCinet. They also provided two GigE switches for use where needed.

With the wide variation in hardware, operating systems, shared storage, and message-passing layer implementation, interoperability was a huge concern. The Virginia Tech machine, in particular, stands out for three reasons. First, its PowerPC 970FX CPU was big endian whereas the more common AMD and Intel CPUs were little endian. Second, System X was running Mac OS X 10.2.9 Server which is *not* a Linux variant like the other systems. Third, System X, along with the Panta cluster, used NFS as the shared file system. Since System X comprised more than 72% of the 3,048 processor cores in GreenGene, its homogeneity had a proportionately high impact on the architectural design. For GreenGene to complete the NT-complete computation, the infrastructure had to be independent of byte order, operating system (OS), and shared file system. In addition to hardware heterogeneity, software heterogeneity played a large role, particularly the OS, MPI implementation, and mpiBLAST version.

[5]The entire cluster was unavailable during Bill Gate's keynote speech at SC|05.

4 "Gluecode" for the GreenGene Ad-Hoc Grid

In this section, we discuss the architecture of the software and the challenges that needed to be overcome to build a working computational grid.

4.1 Software Architecture

As discussed in Section 2, the key factor in ensuring that mpiBLAST runs efficiently and completes a query was the amount of RAM on the compute nodes. There must be sufficient memory for each database segment to fit into the RAM of each compute node or performance suffers due to data being fetched from disk. This implies that clusters with large amounts of RAM per node will require fewer nodes to keep the database in memory. There also must be sufficient RAM to store intermediate results. Because of the lack of a good correlation between a predictor, such as query size, and output size, it is also extremely difficult to estimate *a priori* how much memory a query requires for intermediate results. Therefore, we took the pragmatic approach of classifying queries at run time as "hard" or "easy" depending on whether or not they ran to completion within a reasonable amount of time using the amount of memory available. If the query did not complete, we re-ran the query on a larger group of nodes.

All of the clusters in GreenGene had multiple processors or cores per node. As is common in running parallel processes on SMP machines, we treated each core as a "virtual node" in order to make maximum use of the available compute power. Thus, a physical node on the DuPont cluster was considered to be four virtual nodes and the amount of memory per virtual node was one fourth of the total. We called the set of virtual nodes that ran a mpiBLAST job a *group*. The minimum size of a group, which was dependent upon the amount of memory per virtual node, ranged from 8 to 16 physical nodes depending on the cluster. As a result, each of the clusters contained several to many groups, depending on the number of nodes. The smallest cluster contained four groups and the largest contained nearly 300 for a total of over 375 groups.

Each group in GreenGene was an independent worker capable of running queries. In order to accomplish the tasks of distributing queries and accumulating results, we used a hierarchical software architecture as shown in Figure 3. The *GroupMaster* was responsible for running mpiBLAST to execute a query. It was also responsible for fetching queries from the *SuperMaster* and returning the results as the queries complete. The SuperMaster's sole function was to assign queries to GroupMasters, as

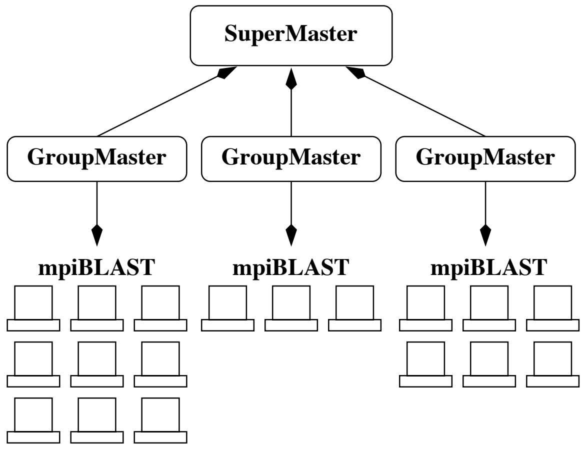

Figure 3: Software Architecture of the GreenGene Ad-Hoc Grid

appropriate, in order to balance load across the hardware running at varying speeds.

Searching hard queries against the NT database required substantial computational resources and generated a very large amount of output. mpiBLAST 1.4 (the most recent and widely used mpiBLAST version), despite its high efficiency in searching easy queries that produce a small volume of output, had limitations in dealing with hard queries. In mpiBLAST 1.4, a worker generated intermediate results after it searched a query sequence against a local database fragment. The intermediate results only contained the alignment metadata and needed to be converted to final output by combining with the corresponding sequence data. During the search, all intermediate results and their corresponding sequence data were sent to and buffered by the master. Once a query sequence was finished, the master sorted the intermediate results according to their similarity scores and converted them to produce the final output.

The above architecture resulted in two major issues. First, the *result processing* (converting intermediate results to final outputs) was serialized by the master. Second, the master needed to buffer all intermediate results and their corresponding sequence data, which can dramatically impact performance when the master's memory size is not large enough for buffering. These limitations made the master a potential performance and space bottleneck. As a result, mpiBLAST 1.4 performed slowly or hung when hard queries were sequence-searched.

To address the above problems, we created mpiBLAST-1.4-PIO, an enhanced version of mpiBLAST that combines mpiBLAST-1.4 with the parallel I/O capabilities of pioBLAST [Lin et al. 2005] to dramatically improve scalability and throughput when searching hard queries. In mpiBLAST-1.4-PIO (hereafter referred to as mpiBLAST-PIO), after searching a fragment, workers convert their intermediate results into the final output format and send final output *metadata* to the master. The master figures out the output offsets for each record in the global output file and sends the information to the workers. With output offsets, workers are then able to write local output records in parallel to the file system through the MPI-IO interface. By having workers processing results in parallel and locally buffering query output, mpiBLAST-PIO removes the performance and space bottlenecks that were present in the mpiBLAST 1.4 master.

With the aforementioned parallel-write solution, mpiBLAST-PIO delivers dramatic performance improvement on parallel file systems. However, it does not perform well over NFS. This posed a new challenge for us because more than 72% of our processor resources were located on machines using NFS as the shared file system. We tackled this challenge by designing a customized version of mpiBLAST-PIO with specific optimizations for better performance on NFS. The idea is to provide pseudo-parallel-write functions for output in substitution of parallel-write APIs from the MPI-IO library. When called by workers, the pseudo-parallel-write functions split output data into small messages and send them to a writer process through an asynchronous MPI communication interface. The writer process collects the output data and writes them out through the Posix I/O interface. With careful design of the asynchronous communication protocols for output data, the pseudo-parallel-write version of mpiBLAST-PIO is able to search hard queries efficiently on machines that do not have parallel file systems.

4.2 Addressing the Challenges

In addition to the inherent complexity of computing the complete NT-against-NT alignment, we must address an array of practical concerns in our GreenGene ad-hoc grid. First is the matter of heterogeneity. Heterogeneity naturally arises from the fact that GreenGene spans many different systems, each with their own hardware characteristics, system software, and administrative policies. Second is the issue of the resources residing in different administrative domains with different cyber-security policies. Third is scalability. Scalability problems arise from overheads in inter-node communications and from inefficiencies in hardware and software implementations. Because of the large amount of resources and number of software groups, the ability to scale well is a serious concern. Fourth is the issue of fault tolerance. This is a serious problem because the probability of failure increases with the number of components in the system. The final issue is data integrity. In the end, the data must be valid for it to be useful.

4.2.1 Heterogeneity and Accessibility

To solve the issue of heterogeneity and to eliminate most of the problems associated with different administrative domains, GreenGene only uses four existing, cross-platform tools – perl, ssh, rsync, and bash – to stand-up the ad-hoc grid that mpiBLAST runs on. Perl is used to write the "gluecode" scripts because it is already available on all of the clusters. As a scripting language, perl allows gluecode to be written quickly and efficiently. *Standing up our GreenGene ad-hoc grid required only five scripts, which totaled only 458 lines of commented Perl code.*

To handle communication between sites in order to coordinate efforts and to transfer data, a combination of ssh and rsync was employed for several reasons. First and foremost, they are readily available on all the clusters, and holes through the institutional firewalls for ssh already exist, thus greatly simplifying the deployment problem. By using ssh directly and as the underlying remote shell for rsync, we could start testing on the various clusters as quickly as accounts became available. Second, the use of ssh, a widely used secure login protocol, gives system administrators greater confidence that their systems will not be compromised. This too greatly eases the task of deployment.

Because mpiBLAST was run multiple times across each cluster, each invocation of mpiBLAST needed to be configured to not interfere with the others. The configuration was too tedious to be done by hand, especially with the very large number of mpiBLAST invocations that ran simultaneously on System X. For the purposes of configuration, GreenGene used custom bash login scripts to automatically configure the environment variables for each invocation. The perl scripts queried the shell environment variables to obtain the values needed to parameterize their execution. Using a bash login script eliminated the thankless task of configuring a separate perl script for every invocation individually and ensured that each machine had the same parameterized configuration. Furthermore, script installation was a simple matter of using scp to distribute the scripts.

4.2.2 Design for Scalability

In some ways, running mpiBLAST on GreenGene is a parallel-computing dream come true. At the query level, it is nearly embarrassingly parallel and allows large numbers of queries to be run at the same time. The limit is primarily the cost of distributing the queries in order to load balance and the cost of retrieving the results. As Figure 3 shows, we take a hierarchical approach to address the overhead issue. Not necessarily as apparent in the figure, is the need to invert the usual relationship that a master-slave organization, such as ours, implies. Usually, the SuperMaster would be the master and the GroupMasters would be the slaves. However, because groups reside behind firewalls, the SuperMaster is unable to initiate contact to GroupMasters. Instead the GroupMasters must assume the role of masters (communication originators) in order to allow communication through the firewall. The SuperMaster takes a more passive role.

The SuperMaster is a perl script that performs four functions: checks for new groups, assigns queries to groups, verifies the results returned, and retires the query. To hide the latency of wide-area network (WAN) connections to the clusters, the SuperMaster maintains a queue of outstanding queries for each group such that the group always has a new query available when completing the old one. (The desired length of the queue is related to the latency of the connection to the group. The higher the latency, the longer the queue required to prevent "bubbles in the pipeline.") Periodically, the GroupMasters fetch all the queries that the SuperMaster has assigned to them.

There is a single GroupMaster for each group of compute nodes. The GroupMaster is composed of three daemons: one to fetch and verify assigned queries from the SuperMaster, one to transfer the results back to the SuperMaster, and one to spawn a process that executes a mpiBLAST job. The mpiBLAST jobs execute on the compute nodes within a group. In order to determine at run time if a query is hard or not, a watchdog process that times out after 30 minutes is spawned prior to executing the query. (We chose the timeout value to be about 2–3 times the average execution time of queries from preliminary runs.) If the watchdog expires before the query completes, mpiBLAST is killed, and the query is reclassified as hard. Otherwise, the watchdog is killed when the query completes.

Another concern is data integrity. Based upon work done by Paxson [Paxson 1999] and the large amount of data expected to be transferred by the network, it is expected that a significant number of packets may be silently corrupted, i.e., a silent error every 500 GB . (By silent, we mean that TCP/IP checksums will not catch the error, thus compromising the integrity of the data.) To ensure that files are transferred successfully, both the GroupMasters and the SuperMaster check MD5 sums for the files. If any of the sums fail, the offending file is transferred again a finite number of times before setting the query aside for manual intervention and debugging. We also save the mis-transferred files in order to classify the types of errors that are seen.

One of the main advantages of the loosely-coupled, partially-inverted, hierarchical architecture is the flexibility of adding GroupMasters as new groups become available. There can be as many GroupMasters as needed; the number is not static, but fluctuates, as needed. The flexibility also allows the system to be started incrementally and for failed nodes to be restarted from where they left off. It also allows the queries from permanently failed groups to be reassigned easily. Furthermore, the flexibility makes it possible to consolidate two or more idle groups on a cluster to form a larger group in order to tackle the hard queries. Finally, the resulting topology also reduces the load on the SuperMaster as most of the work occurs at the GroupMaster level where it can scale as the number of groups grows.

The high rate of query execution on GreenGene presents another problem. Preliminary work indicates a rule of thumb of about 1 MB/s of mpiBLAST results are generated per CPU core. On the clusters with a GigE connection to the SuperMaster, this data rate will easily saturate the bottleneck link with as few as 125 cores (or about 15 groups). Even for System X with a 10-Gb/s connection to the SuperMaster, only about 1250 of the 2200 CPUs could be kept busy. Fortunately, rsync, which is used to transfer query files and results, has the capability to do on-the-fly compression. Because the query results are highly redundant, the compression ratio is between 5:1 and 7:1. This is more than enough to remove the network bandwidth from being the bottleneck. Furthermore, results are cached until several can be sent at once in order to amortize the cost of initiating a TCP/IP connection and to allow the connection to reach streaming state.

4.2.3 Fault Tolerance

Fault-tolerance has been identified as one of the most serious issues inhibiting the availability and usage of large-scale computing resources. Clusters with thousands of processors generally exhibit a mean time between failure of less than 10 hours [Reed 2004]. Because there are more than 3,000 processors in GreenGene, the majority of which are not in the control of the authors, we assume that the number of faults will be high. To complicate matters further, GreenGene is connected over wide-area networks (WANs) spanning the continental United States.

Traditional high-performance computing (HPC) applications, e.g., iterative numerical simulations, execute for long periods of time and build up a complex state in

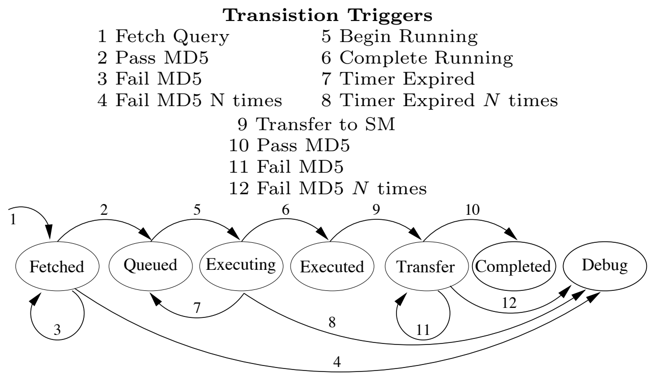

Figure 4: Query FSM on the GroupMaster

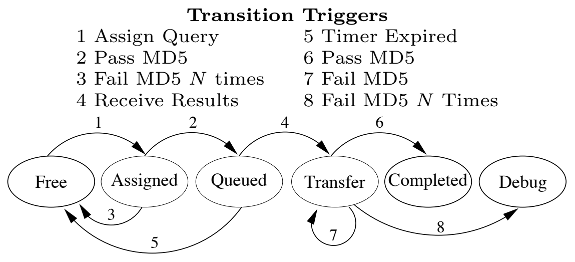

Figure 5: Query FSM on the SuperMaster

memory. Because the ratio of computation to memory state is very high, an efficient method for achieving fault tolerance is to periodically save the state of the computation to disk. Recovering from a fault consists of reloading the last checkpoint rather than re-executing the computation.

In contrast, Table 3 indicates that more than 95% of the sequences in the NT database, namely those in group G1, are likely to execute in less than 5 sec. Therefore, a checkpoint and restart approach is much less efficient than re-executing the query for most queries. Following the Google model [Barroso et al. 2003; Ghemawat et al. 2003], we have designed the system to have as many interchangeable parts (groups) as possible. Because of the potentially large cost of migrating a query to a new group, a failed computation is first re-executed on the same group. The primary issue with re-execution is the management of query status. Has the query been successfully fetched from the SuperMaster, but failed during execution? Or did the query complete successfully, but fail while transferring the results back to the SuperMaster?

The GreenGene grid software accomplishes this task by maintaining all query status on disk through a standard file system. The state for a GroupMaster is contained within a directory whose path is specified in a shell variable. Subdirectories represent the status of queries executing on the group. The file containing a query in the "queued" state would be in the `queued` directory, while a completed query (along with its output) would be in the `completed` directory. By being careful about when a query is advanced to the next state, e.g., not moving a query and its output from the the directory denoting the transfer state to the directory denoting the completed state until after the SuperMaster has indicated that the results have been received and validated via the MD5 sum, we can guarantee that no queries will be lost and that the group can resume execution after software failure. Figure 4 provides the finite state machine for queries on the GroupMaster, realized as directories within the group directory. Likewise, the finite state machine for queries on the SuperMaster is given in Figure 5.

A detailed description of the state machines is beyond the scope of this paper. However, a few observations are warranted. First, the validity of files transferred over the network is always verified via a MD5 sum. Failed checks are re-tried a finite number of times in case the failure is due to a transient error in the network. Persistent network errors cause queries to transition to the "debug" state indicating the need for manual intervention. Second is the ease with which hard queries can be identified and handled. Since it is not possible to know the resource requirements of the queries *a priori* in order to schedule the queries on nodes of sufficient resources, it is not possible to schedule queries statically. When a query is too complex for a group, it cannot complete in time and is re-scheduled. On the positive side, because groups are not the same, a query that is too complex for one group may not be too complex for another. For example, a query that fails to complete before the timer on Group A due to a lack of memory ends up in the debug state and the SuperMaster re-schedules it for another group, say Group B. Because group B has twice as much memory, the query is now able to complete in the allotted time. In this manner, hard queries are identified and eventually re-assigned to higher-performing groups. If a query does not complete when re-executed on the highest-performing group, it fails and is moved to the debug directory on the SuperMaster. After groups become idle, they are manually combined to form a new group and the hard queries are moved from the debug directory on the SuperMaster to the "free" (unassigned) directory where they will be re-assigned to the more powerful groups.

There are several advantages that come from using the file system to hold the state. First, the state is automatically persistent across software (and many types of hardware) failures or hiccups. Second, we are guaranteed that the tools are already installed because file system commands are necessary for any use of the machine. Third, using file system commands makes it eas-

```
>gb|AE008723.1| Salmonella typhimurium LT2, section 31
of 220 of the complete genome
Length = 22294

Score = 46.1 bits (23), Expect = 0.054
Identities = 32/35 (91%)
Strand = Plus / Plus

Query: 157   ctgctgcatggcggcgcatcggtagcgctggcgga 191
             ||||| || ||||||||||||| ||||||||||||
Sbjct: 21920 ctgctacacggcggcgcatcggcagcgctggcgga 21954
```

Figure 6: Representative Query Match

ier to develop the software because the functionality of yet-to-be-created scripts can be emulated by simple file system commands on the command line. Finally, the very loosely coupled approach implied by this implementation technique makes it very easy to pre-fetch sufficient queries and cache sufficient results to accommodate transient network failures.

Clearly, there are two major concerns with using the file system to save the system state. First is the cost of performing file system operations, such as `mv`. As long as directories do not get too large, the cost of using the `mv` command is minimal. The only directories that have the potential to get too large are the directory containing the queries initially and the final directories containing the results. For the initial directory, we use a set of hierarchical directories to get the number of files per directory below 1,000. We do nothing special for the final directories. Second, there is a potential for the state to be lost if the disk fails. In the case of disk failure, we are no worse off than if we had chosen a different approach. Either way the state is lost. If instead we use an in-memory database to hold the state, the state is not necessarily persistent. If we use an on-disk database, the state will still be destroyed by disk failure.

5 Results and Experience

At the beginning of SC|05, the NT database contained 3,563,759 sequences. We accumulated several sequences into each query in order to amortize the setup overhead of mpiBLAST. While accumulating as many queries as possible makes the overhead of setting up a mpiBLAST computation negligible, large queries also increase the amount of work that has to be re-done if the computation failed while searching a sequence in the query. Large queries containing many sequences also reduce opportunities for load balancing across groups. Therefore, just as we did in our pre-run characterization of the computation in Table 3, sequences here are coalesced into queries targeting an average execution time of 30 minutes. As a result, each query file averages 56.8 KB

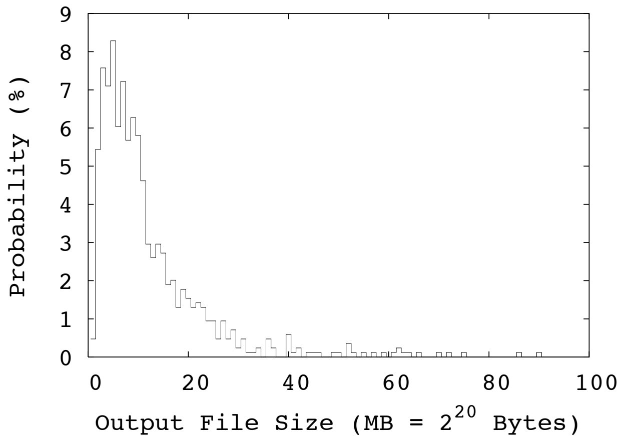

Figure 7: PDF of Output File Sizes

in size, containing 52.6 sequences. Executing the query creates an output file for each sequence in the query containing the 500 best matches from the NT database.

Figure 6 presents a representative match from a query against the NT database. This match is the third best match in the output for this query against the whole database. Of particular interest is the “Expect” score, also called the E-value. The last three lines give query and subject strings with a line detailing the match in between. The numbers to the left and right of the sequences give the offset of the left and right edges within the complete sequence. The vertical bars between the query and subject show areas of perfect alignment. The figure shows that out of the 34 base pairs, only 3 are mismatched.

The sizes of the output files range from 603.4 KB to over 7 GB. The mean output file size is 14.6 MB and the median size is 8.28 MB. Figure 7 shows the probability density function (PDF) of the size of the output files. The average output size per sequence is 284.2 KB.

The probability density function of execution times, shown in Figure 8, has a similar shape to the output file-size distribution. The average execution time for a query is just shy of 9.0 minutes, with a range from 6 seconds to 1.6 hours.

The biggest issue hindering a complete sequence-searching the entire NT database against itself was global storage. When we arrived at SC|05, we discovered that setup was not scheduled until after the show started, thus preventing us from testing the system. Furthermore, only 64% of the 100 TB we asked for was available. While the StorCloud volunteers did all they could to get the file system up and stable as quickly as possible, StorCloud continued to have problems throughout the show. Fortunately, the authors and Intel both brought RAID arrays so we were able to start

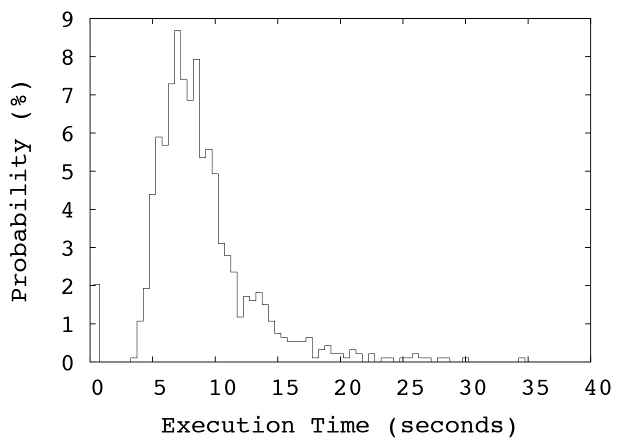

Figure 8: PDF of Execution Time

running earlier than we would have otherwise using the RAID arrays as global storage while we waited for our portion of StorCloud to come on-line (which it finally did on the last day of SC|05).

In spite of all the difficulties with StorCloud, we successfully sequence-searched approximately 10,000 queries containing about 526,000 sequences or 1/7 of the NT database.

6 Lessons Learned and Implications

When we initially envisioned an NT-versus-NT computation in early 2005, the goal was not to create a software infrastructure to support an ad-hoc grid, but to stress-test mpiBLAST while performing computations of benefit to the bioinformatics community. Although we were forced to develop our own infrastructure for the reasons stated earlier, doing so has helped us to understand some of the problems inherent in standing-up a large-scale ad-hoc grid "on the fly."

Many of our design choices proved to be good ones. For example, the choice to use perl scripts to glue existing tools together was crucial to having GreenGene operational during SC|05. If we had attempted to implement the needed functionality directly in C or C++, development time would have exceeded the time available. In exchange for faster development, however, we had to tolerate higher utilization on the head nodes.

Another good choice was to use the file system to implement the state machine that records the status of queries in the system. Once again, very little code had to be written to keep track of queries compared with what would have been required had we used a database. Using the file system allowed us to gain restart capability for very little cost.

Additionally, there are several characteristics that we intend to preserve and enhance in the next version of the GreenGene grid software. The first is portability across many platforms. Currently, the software works on Linux and Mac OS X. It should also work on any other Unix-based platforms. In order to execute on Windows, we would need to find semantically equivalent ways of manipulating the file system. The second is to retain the simplicity and lightweight nature of the implementation to support agile development and facilitate rapid deployment. The third is to further flesh out the implementation of hierarchical scheduling and to ensure that all fault-tolerance issues have been handled. Finally, we need to make the scripts more generic so that computations other than BLAST can be run. It is likely that making the scripts more generic will also require modifications to how the scripts operate since not all computations are as decoupled as sequence search.

7 Conclusion

The goal of this endeavor was to perform an all-to-all comparison of the sequences in the NT database to provide critical information to the biology community. Using resources donated for the week of SC|05, we assembled the GreenGene ad-hoc grid whose software architecture was based on five scripts totaling 458 lines of commented perl code. While a series of storage infrastructure problems prevented us from completely achieving our goal, we ran mpiBLAST on the GreenGene ad-hoc grid and successfully computed alignments for over half a million sequences.

In assembling the resources for GreenGene, we were unable to take advantage of existing grid frameworks, primarily because they were not available on nearly 75% of our resources. Instead, we developed an "ad-hoc grid framework" using commonly available Unix commands. Beside contributing a prototype of a framework for a computational grid spanning the continental United States, our experience serves as a use-case and provides insight on the issues involved in standing up large heterogeneous grids spanning many administrative domains. Finally, this work has resulted in the identification of particularly hard BLAST queries that likely have some special significance to biologists since long execution times and large outputs imply that the target sequence is highly similar to a large portion of the database. The hard queries can also be used to further the development of high-performance sequence-search software.

While the resources that formed GreenGene have returned to the donating organizations, the goal of sequencing the NT database against itself remains. As future work, an improved characterization of the resource requirements of queries is needed in order to increase predictability of resource usage in parallel BLAST and improve the successful scheduling of queries on computational resources. In the longer term, we hope to generalize the framework so that it can be more automated and reusable.

8 Acknowledgments

This effort would not have been possible without the time, effort, and equipment support of many people and institutions. First, we thank the following people who facilitated this endeavor: Erv Blythe, Jeff Crowder, Terry Herdman, Dennis Kafura, Judy Lilly, and Srinidhi Varadarajan (Virginia Tech); Julio Facelli (University of Utah); Mike Gustus, Tung Nguyen, and Patti Yamakido (Panta Systems); Rick Crowell (Foundry Networks); and the SC|05 StorCloud volunteers and vendors. Second, we recognize the invaluable technical contributions of Clark Gaylord, Peter Haggerty, Kevin Shinpaugh, Bill Sydor, and Geoff Zelenka (Virginia Tech); Guy Adams, Wayne Bradford, Martin Cuma, Julia Harrison, Brian Haymore, and Sam Liston (University of Utah); Ted Barragy and Dancil Strickland (Intel); Rusty Benson, Jason Chang, and Steve Sporzynski (Panta Systems); and Nagiza Samatova (ORNL).

This work is supported in part by a DOE ECPI Award (DEFG02-05ER25685), and Xiaosong Ma's joint appointment between NCSU and ORNL.

References

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. 1990. Basic Local Alignment Search Tool. *Journal of Molecular Biology 215*, 403–410.

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. 1997. Gapped BLAST and PSIBLAST: A New Generation of Protein Database Search Programs. *Nucleic Acids Research 25*, 3389–3402.

Barroso, L., Dean, J., and Hölzle, U. 2003. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro 23*, 2.

Bjornson, R., Sherman, A., Weston, S., Willard, N., and Wing, J. 2002. TurboBLAST(r): A parallel implementation of BLAST built on the TurboHub. In *International Parallel and Distributed Processing Symposium.*

The NCBI Handbook (BLAST Output: 1. The Traditional Report). `http://www.ncbi.nlm.nih.gov/books/bv.fcgi?rid=handbook.section.615`.

Braun, R., Pedretti, K., Casavant, T., Scheetz, T., Birkett, C., and Roberts, C. 2001. Parallelization of Local BLAST Service on Workstation Clusters. *Future Generation Computer Systems 17*, 6 (Apr), 745–754.

Camp, N., Cofer, H., and Gomperts, R. 1998. High-throughput BLAST. Tech. rep., SGI, Sep.

Carns, P., Ligon III, W., Ross, R., and Thakur, R. 2000. PVFS: A Parallel File System For Linux Clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference.*

Chi, E., Shoop, E., Carlis, J., Retzel, E., and Riedl, J. 1997. Efficiency of Shared-Memory Multiprocessors for a Genetic Sequence Similarity Search Algorithm . Tech. rep., University of Minnesota.

Darling, A., Carey, L., and Feng, W. 2003. The Design, Implementation, and Evaluation of mpiBLAST. In *ClusterWorld Conference & Expo and the 4th International Conference on Linux Cluster: The HPC Revolution 2003.*

Dumontier, M., and Hogue, C. W. 2002. NBLAST: A Cluster Variant of BLAST for NxN Comparisons. *BMC Bioinformatics 3*, 13 (May), 271–282.

EPCC Sun Data and Compute Grids Project Status Update. `http://www.sun.com/products-n-solutions/edu/events/archive/hpc/2003presentations/heidelberg/GRID05_Ratna_Abrol.pdf`.

EPCC Using Sun Grid Engine and Globus to Schedule Jobs Across a Combination of Local and Remote Machines. `http://www.epcc.ed.ac.uk/sungrid/DISSEMINATION/Regensburg-2002-04.ppt`.

Feng, W. 2003. Green Destiny + mpiBLAST = Bioinfomagic. In *10th International Conference on Parallel Computing (ParCo).*

Folding@home. `http://folding.stanford.edu/`.

Foster, I., and Kesselman, C. 1997. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications 11*, 2, 115–128.

Gans, J., 2005. Personal Communication.

Growth of GenBank: 1982–2004. `http://www.ncbi.nim.hih.gov/Genbank/genbankstats.html`.

Ghemawat, S., Gobioff, H., and Leung, S. 2003. The Google File System. In *Proceedings of the 19th Symposium on Operating Systems Principles*.

Gropp, W., Lusk, E., Doss, N., and Skjellum, A. 1996. High-performance, portable implementation of the MPI Message Passing Interface Standard. *Parallel Computing 22*, 6, 789–828.

InfiniBand Trade Association, 2000. InfiniBand Architecture Specification, Release 1.0, Oct.

Kent, W. J. 2002. Blat – The BLAST-Like Alignment Tool. *Genome Research 12* (Apr), 656–664.

Lin, H., Ma, X., Chandramohan, P., Geist, A., and Samatova, N. 2005. Efficient Data Access for Parallel BLAST. In *International Parallel and Distributed Processing Symposium*.

Liu, J., Wu, J., Kini, S., Wyckoff, P., and Panda, D., 2003. High Performance RDMA-Based MPI Implementation over InfiniBand.

Liu, J., Wu, J., Kini, S. P., Wyckoff, P., and Panda, D. K. 2003. High Performance RDMA-Based MPI Implementation over InfiniBand. In *17th Annual ACM International Conference on Supercomputing*.

The Lustre File System. `http://www.lustre.org/`.

mpiBLAST-G2. `http://www.twgrid.org/News_Event/News/mpiblast`.

MPICH2. `http://www-unix.mcs.anl.gov/mpi/mpich2/`.

Paxson, V. 1999. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking 7*, 3 (June), 277–292.

Pedretti, K., Casavant, T., Braun, R., Scheetz, T., Birkett, C., and Roberts, C. 1999. Three Complementary Approaches to Parallelization of Local BLAST Service on Workstation Clusters. *Lecture Notes in Computer Science 1662*, 271–282.

Rapier, C., and Stevens, M. Application Layer Network Window Management in the SSH Protocol. Tech. rep., Pittsburgh Supercomputing Center.

Reed, D. A., 2004. High-End Computing: The Challenge of Scale. Director's Colloquium, Los Alamos National Laboratory, May.

Schmuck, F., and Haskin, R. 2002. GPFS: A Shared-Disk File System for Large Computing Clusters. In *Proceedings of the First Conference on File and Storage Technologies*.

SETI@home. `http://setiathome.ssl.berkeley.edu/`.

Singh, R. K., Dettloff, W. D., Chi, V. L., Hoffman, D. L., Tell, S. G., White, C. T., Altschul, S. F., and Erickson, B. W., 1996. BioSCAN: A Dynamically Reconfigurable Systolic Array for Biosequence Analysis. `http://www.dlhoffman.com/~hoffman/papers/nsf96.ps.gz`.

Smith, T. F., and Waterman, M. S. 1981. Identification of Common Molecular Subsequences. *Journal of Molecular Biology 147*, 195–197.

StorCloud. `http://sc05.supercomputing.org/initiatives/storcloud.php`.