ELSEVIER

Future Generation Computer Systems 17 (2001) 745–754

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.nl/locate/future

Parallelization of local BLAST service on workstation clusters

R.C. Braun, K.T. Pedretti*, T.L. Casavant, T.E. Scheetz, C.L. Birkett, C.A. Roberts

Coordinated Laboratory for Computational Genomics, Department of Electrical and Computer Engineering, University of Iowa, Iowa City, IA 52242, USA

Abstract

This paper describes approaches to improve the performance of one of the most common and increasingly important aspects of the Human Genome Project (HGP) — large-volume, batch comparison of DNA sequence data. This basic comparison operation, usually carried out by the well-known BLAST program on one subject sequence against the internationally available databases of nearly five million target sequences, is already used hundreds of thousands of times each day by researchers around the world. At present, it is still used primarily in single query, or small batch query mode. As the entire sequence of the human genome nears completion, the area of functional genomics, and the use of micro-arrays of sets of genes, is coming to the fore. These developments will demand ever more efficient means of BLASTing sets of data that will make single processor implementation on powerful workstations infeasible. We describe the three primary parallel components to BLAST. The first is at the sequence-to-sequence comparison level. The second parallelizes a single query across a partitioned and distributed database. Finally, the set of queries themselves are partitioned across a set of servers with replicated or partitioned databases. The three methods may be employed alone or in concert. Our current implementation is described which parallelizes batch requests, and our plans for implementation of the other levels is also described. The results will ultimately be applied to hardware assistance for this soon-to-be primitive computer operation.

Keywords: Parallel cluster application; Genome project; BLAST; Heterogeneous parallelism; Network computing

1. Introduction

Modern genome sequencing, discovery and mapping research efforts *are* parallel/distributed processing systems. The processing involved is computationally demanding, but the basic nature of the entire process of gene discovery and understanding is highly parallel, heterogeneous, and distributed. To date, the dominant component of the *human genome project* (HGP) has been the discovery of the entire three billion base pairs of sequence of the human genome. This has been a worldwide parallel/cooperative effort in which the dimensions of parallelism have been drawn at the boundaries of organisms (e.g., human, mouse, rat, C. elegans, etc.) [1–3], and chromosomes of these organisms. However, close examination of this phase of the effort reveals that the parallelism being exploited is mostly at the job-level [4]. Partitioning of the sequencing effort along organism and chromosomal lines requires only very rudimentary parallel task structure, inter-process communication (IPC) and synchronization (e.g., often via human examination of WWW sites displaying similar regions of differing organisms).

As the entire sequence of the human genome nears completion (draft expected in 2000, and finished sequence expected in 2003), the frequency and intensity of inquiries against this data will expand exponentially. The area of *functional genomics* will require batch processing of large numbers of requests to identify homologous groups of sequences. The current

* Corresponding author.
E-mail addresses: pedretti@eng.uiowa.edu, genome@eng.uiowa.edu (K.T. Pedretti).

0167-739X/01/$ – see front matter
PII: S0167-739X(00)00057-1

	Fine Grained	Medium Grained	Coarse Grained
Subject(s)	1 sequence	1 sequence	N sequences (batch request)
Target(s)	1 sequence	M sequences (in database)	M sequences (in database)
Parallelism	Multiple alignments on single sequence pairs.	Partition Database / Multiple targets examined at once	Replicate Database / Partition Input Sets

Fig. 1. Three levels of parallelism exploitable in large batch BLAST processing.

mode used by 90% of researchers is to submit single queries for comparison of a segment of sequence data (a subject string of 300–600 characters) against one or more databases being served at a national or international repository. The most common such repository is GenBank which is housed in the National Center for Biotechnology Information (NCBI) at the National Institutes of Health (NIH) in Bethesda, MD, USA. (Two other large repositories also exist — one in Europe and one in Japan.) The most common sequence comparison tool is the well-known Basic Local Alignment Search Tool (BLAST) [5] program, which is also available for download for local execution. While it is possible for anyone to download the contents of GenBank (as of October 1999 containing 3 841 163 011 bases, from 4 864 570 reported sequences) for processing of queries on a dedicated server, this is rarely done. Rather, many thousands of single queries "hit" a large bank of database server systems at NCBI on a daily basis. Not only does this cluster of servers continue to diminish in its ability to serve the ever mounting numbers of requests, but the network traffic generated by this load is also becoming intolerable. The databases themselves are growing at an increasing rate, and single queries on a dedicated high-performance system can also be time consuming. While some efforts have been made in the past to parallelize BLAST searches, none of the methods exploit all levels of available parallelism, and none of these tools has been implemented for easy public access. Lastly, a comprehensive understanding, and a robust, near-optimal solution to this problem is a necessary first step toward development of parallel architectures, and hardware assists for this soon-to-be ubiquitous and primitive computer operation.

In this paper, we describe a comprehensive approach to exploitation of three distinct types of parallelism in BLAST searches. The three types derive from various granularities of searches, and different inherent parallel aspects of BLAST searching. Fig. 1 summarizes the three complementary approaches.

Conceptually, the lowest level involves speeding up the comparison of a single pair of DNA sequences — a subject and target — by performing all the alignments of the comparison in parallel. These operations may be performed in an "embarrassingly parallel" manner with no data dependencies between them. It should be noted that this lowest level may be best performed on a node with special parallel capabilities as well — beyond those of the typical symmetric multi-processor (SMP) system. In the second case, a large target database can be partitioned into subsets, and distributed to the static (disk) storage devices of a cluster of workstations — possibly replicating each subset some number of times on several nodes of the cluster. Single queries would then be replicated to the multiple nodes and the comparisons against each partition of the database would proceed in parallel. While non-trivial, the merging of the results is feasible, and can be done efficiently at an appropriate level of

granularity. Finally, if a large set of queries is to be processed in a "batch" mode, partitioning of the multiple query requests can also be done to allow even more parallelization.

Our current implementation exploits only the last of these three approaches in production. Implementation of the second method is nearly complete and is expected to be deployed in December 1999. The final level is a subject of further research. The eventual goal is to implement all three of these granularities in a hybrid cluster-server architecture for use in the local BLASTing of expressed sequence tag (EST) and functional genomics study at the University of Iowa. All developed software would be made available to the research community. Architecture enhancements to greatly improve exploitation of fine-grained parallelism will be a natural extension of this work.

2. Background

BLAST is a heuristic search algorithm employed by a number of genetic search tools. These tools are used by researchers to identify similarity between genetic sequences. Typically, there is an input sequence that is *BLASTed* against a database of known sequences — the target set. The result of a BLAST search is a list of sequences from the target set that were found to have significant regions matching regions of the input sequence. In large-scale sequencing projects, this data can be useful to determine if a particular sequence has already been discovered or if contamination has occurred. BLAST can also be useful in a broader sense by providing insight into a sequence's function by matching it with some sequence of known function.

The specific implementation of BLAST used at the University of Iowa is NCBI BLAST (freely available from *ftp://ncbi.nlm.nih.gov/blast*). NCBI BLAST recognizes two types of sequences: *nucleotide* and *peptide*. Both sequence types are represented as a string of ASCII characters. Nucleotide sequences are made up of four letters (A, T, C, and G). These represent the four bases in DNA — adenine, thymine, guanine, and cytosine. Thus, a short nucleotide sequence input into BLAST might be ACCTGACTACCT. This string also codes for the complementary DNA strand TGGACTGATGGA (A bonds with T, and C bonds with G). One can imagine these two sequences of nucleotides being placed in parallel and then twisted to create the familiar DNA double helix. For nucleotide to nucleotide queries, NCBI BLAST takes the complementary strand of the query into account when searching. A peptide sequence (a protein sequence) is also made up of a string of letters but, in this case, each represents one of 20 amino acids. Peptides are encoded by triplets of nulceotides, as specified by the genetic code. There are three possible reading frames (+0, +1, +2) in both directions, and therefore, there are six ways to translate a nucleotide sequence into a peptide sequence.

The NCBI BLAST distribution contains five variations of BLAST — `blastn, blastx, tblastx, blastp, and tblastn`. `blastn` compares a nucleotide sequence against a nucleotide database and is relatively quick. `blastx` compares a nucleotide sequence against a protein database. To do this, the nucleotide subject needs to be translated into a peptide sequence. Since there are six different translations, the basic BLAST algorithm must be applied six times to complete the query. Like `blastn, tblastx` compares a nucleotide sequence to a nucleotide database only in this case each is translated (in all six reading frames) into a peptide sequence before BLASTing. This is the most computationally intensive of the blast programs since the BLAST algorithm must be invoked 36 times for each sequence to sequence comparison. `blastp` compares a peptide sequence to a peptide database and is relatively quick. `tblastn` compares a peptide sequence against a nucleotide database. As with `blastx`, each sequence to sequence comparison requires six calls to BLAST.

A sample output from a `blastn` BLAST run is shown in Fig. 2. The query sequence was 296 bases long and 15 sequences were found to have significant alignments. The two figures of merit for an alignment are its *score* and *E* value. Long matches get high scores. In the sample output, the hit that matched 296 bases received a score of 587 while the hit that matched 13 bases only received a score of 26. The *E* value (or "expect" value) represents the number of significant alignments one would expect to see by chance. It is dependent on the size of the target database and it decreases exponentially with score. An *E* value of zero is ideal.

```
BLASTN 2.0.6 [Sept-16-1998]

Reference:
Altschul, Stephen F., Thomas L. Madden, Alejandro A. Sch ffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs",  Nucleic Acids Res. 25:3389-3402.

Query=
         (296 letters)

Database: CustomBlastDB
           104 sequences; 43,616 total letters

Searching..................................................done

If you have any problems or questions with the results of this search
please refer to the BLAST FAQs

                                                                 Score     E
Sequences producing significant alignments:                      (bits)  Value

UI-R-A0-ah-b-10-0-UI       800      0    800  ABI        587  e-170
UI-R-A0-av-f-10-0-UI.s1    798      0    798  ABI         26  0.15
UI-R-A0-ar-g-06-0-UI       790      0    790  ABI         26  0.15
UI-R-A0-ar-h-09-0-UI       790      0    790  ABI         24  0.59
UI-R-A0-ax-e-06-0-UI       718      0    718  ABI         24  0.59
UI-R-A0-bi-d-06-0-UI.s1    719      0    719  ABI         22  2.3
UI-R-A0-bk-f-10-0-UI.s1    789      0    789  ABI         22  2.3
UI-R-A0-ae-d-01-0-UI       697      0    697  ABI         22  2.3
UI-R-A0-ak-h-08-0-UI.s1    461      0    461  ABI         22  2.3
UI-R-A0-aj-f-02-0-UI.s3    753      0    753  ABI         22  2.3
UI-R-A0-aj-d-08-0-UI.s3    632      0    632  ABI         22  2.3
UI-R-A0-ao-a-12-0-UI.s1    802      0    802  ABI         22  2.3
UI-R-A0-aw-d-02-0-UI.s1    779      0    779  ABI         22  2.3
UI-R-A0-bd-a-10-0-UI.s1    766      0    766  ABI         22  2.3
UI-R-A0-bl-a-03-0-UI.s1    729      0    729  ABI         22  2.3

 UI-R-A0-ah-b-10-0-UI    800      0    800  ABI
           Length = 391

 Score =  587 bits (296), Expect = e-170
 Identities = 296/296 (100%), Positives = 296/296 (100%)

Query: 1   ctaaaaacatggtgttcgttaaagcgggacctgggatggaggaactgcagacaaggcatt 60
           ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 55  ctaaaaacatggtgttcgttaaagcgggacctgggatggaggaactgcagacaaggcatt 114

Query: 61  gcaagcagaaagtgcattcgaaaccaataagcgtgcactcctggctctctggccacccag 120
           ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 115 gcaagcagaaagtgcattcgaaaccaataagcgtgcactcctggctctctggccacccag 174

Query: 121 gggcagcaaggcaagtggaggcccaaaggctcactccttagaatgcccactagggagggg 180
           ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 175 gggcagcaaggcaagtggaggcccaaaggctcactccttagaatgcccactagggagggg 234

Query: 181 aggccagaactccaccatcgtggagggagggaaatgagagaacaggaacgtgagaaggg 240
           ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 235 aggccagaactccaccatcgtggagggagggaaatgagagaacaggaacgtgagaaggg 294

Query: 241 gtgccagaccaaggggtcatgggacaaagaacagaccccaagcatctggcacctag 296
           ||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct: 295 gtgccagaccaaggggtcatgggacaaagaacagaccccaagcatctggcacctag 350

 UI-R-A0-av-f-10-0-UI.s1    798      0    798  ABI
           Length = 372

 Score = 26.3 bits (13), Expect = 0.15
 Identities = 13/13 (100%), Positives = 13/13 (100%)

Query: 124 cagcaaggcaagt 136
           |||||||||||||
Sbjct: 243 cagcaaggcaagt 231


CPU time:     0.06 user secs.       0.06 sys. secs          0.12 total secs.

  Database: CustomBlastDB
    Posted date:  May 19, 1999  8:13 AM
  Number of letters in database: 43,616
  Number of sequences in database:  104

Lambda     K      H
    1.37    0.711     1.31

Gapped
Lambda     K      H
    1.37    0.711     1.31
```

Fig. 2. Sample BLAST output.

3. Approach and options

3.1. Local BLAST

Although NCBI, along with several other research centers, maintains public BLAST servers, these have their limitations. Foremost, depending on the application, is the speed at which the searches are performed. During peak times, the servers become extremely saturated. Many projects rely on data being processed in a timely manner. In such cases, this possible delay and reliance on an outside provider is unacceptable. Additionally, the databases which can be selected for use, although current, are limited to those available on the remote server. The ability to create custom, real-time databases to BLAST against, is increasingly becoming a necessity for many projects. For these reasons, local installation of a BLAST server and the basic set of public databases is essential for any large-scale sequencing or functional genomics effort.

3.2. Local parallel BLAST

Given the need for local BLAST services, the need to complete these searches in a timely manner becomes evident. Parallelizing the BLAST algorithm pays dividends by effectively reducing the processing time in relation to the number of compute nodes utilized. In addition to reducing the processing time, in some cases parallelizing can reduce costs by utilizing commodity workstations and even PCs. Finally, a locally scheduled parallel algorithm allows for prioritization and a level of control over individual searches not afforded by any other option.

3.3. Types of parallelization

Three basic approaches to parallelizing BLAST can be readily identified, and are currently in various stages of implementation at Iowa. They are summarized in Fig. 1, and are described below.

3.3.1. Pairwise multiple alignment parallelization

The notion of multiple alignments of a single pair of DNA subsequences was described in Section 2. It is clear that if two subsequences are to be compared — of lengths n and m, respectively — then there are $O(n \times m)$ possible alignments to be examined for possible similarity for the pair. Since these comparisons are mutually independent, the parallelization of the comparisons is potentially very efficient. The granularity of this operation is such that effective implementation would greatly benefit from specialized hardware, or a large-scale MPP system with a custom interconnect. Implementation on a modest-sized SMP would be feasible using threads. However, single nodes of a cluster utilizing this level of parallelism could possibly be an IBM SP-2 class system. Of greatest importance would be a high-speed, low-latency interconnection network to allow rapid selection and scoring of the best possible alignment.

3.3.2. Database (target set) partitioning

The next larger grain of parallelization involves distributing "chunks" of the database(s) across a collection of compute nodes. This allows for less demanding memory requirements as smaller pieces of the database are held in persistent storage, and loaded into memory of each node. Additionally, when not at peak load across compute nodes, individual jobs can be completed much more quickly as the power of multiple nodes is leveraged. In this scheme, a master node coordinates the scheduling of jobs and collates the results from each submission. A typical scenario might involve 8–10 workstations with several different chunks of the database, which may have been broken into 4–6 pieces. This will allow for a useful level of redundancy for robustness and reliability, and to incrementally add compute power as databases grow, and more queries are being submitted.

3.3.3. Batch query (subject set) partitioning

The largest grain method of parallelization involves batch processing and scheduling of sets of queries, while keeping full copies of the database stored on each compute node. While this does not reduce the time for an individual search to complete, it is quite effective when used in situations where multiple searches need to be performed in a sustained way on a daily basis, or at the same time. Several methods of optimization can be used when scheduling, including prioritization of interactive jobs if necessary. One disadvantage of this method is the large amount of memory still necessary to allow the entire database to be stored and loaded on each node.

4. Current implementation

To date, only the coarse-grained batch approach has been implemented and used in a production setting. This allows for efficient processing of daily workloads generated by the many ongoing large-scale sequencing projects at the University of Iowa [6,7]. Without this system in place, it would be impossible to complete BLASTing each day's sequence data within a 24 h period. This course-grained approach is detailed in the following section.

The medium-grained implementation is nearing completion and is scheduled to be deployed in a limited fashion in December 1999. Only the interfacing and automation components remain to be completed, and most of these will be shared with the already in place course-grained architecture. The non-trivial changes required to extend the course-grained solution are outlined at the end of this section.

4.1. The batch scheduler

The foundation of the local batch BLAST system is the Portable Batch System (PBS) developed for NASA for their diverse set of high-performance computing resources [8]. At the University of Iowa, this system is used to manage an on-site heterogeneous cluster of SUN, HP, and SGI workstations. In addition, PBS can be used with commodity architectures such as Intel PCs. Such systems, when combined with powerful operating systems such as Linux, allow for the low-cost, high-performance addition of computer resources to a local batch BLAST system. A recent addition to the compute power to be used on this project at the University of Iowa is a 16 node Pentium III Linux cluster. These high-performance machines with Gigabit Ethernet will provide a tremendous leap in computing power which will be managed by the PBS scheduler. A PBS system consists of three distinct parts: the *job server*, the *scheduler*, and *compute nodes* (see Fig. 3).

4.1.1. Server

The PBS job server is responsible for managing a queue of incoming jobs. In the current system, all jobs are BLAST jobs but this need not be the case. Other computationally intensive jobs such as sequence clustering and the creation of radiation hybrid maps can just as easily share computing resources with BLAST. There are currently two job queues in the system, one for batch BLAST jobs and a second for jobs interactively submitted to the Local BLAST Server through a web interface.

4.1.2. Scheduler

The PBS scheduler applies a scheduling algorithm to allocate compute nodes to jobs in the two incoming job queues. Some compute nodes in the current cluster of workstations have several CPUs and therefore can handle more than one simultaneous BLAST job. The PBS scheduler knows about this and will assign multiple jobs to such nodes.

4.1.3. Compute nodes

Each compute node has a PBS node monitor running on it that communicates with the PBS job server. This allows the server to query the node for job status information and also ensure that the node is still on-line. Should a node go off-line, the PBS job server marks it as down and the scheduler ceases to schedule jobs to it. Each compute node has its own set of the sequence databases. This requires that nodes each have enough disk storage for the databases in use (currently over 2.5 GB).

4.2. Job types

There are two types of jobs that are submitted to the Local BLAST Server: batch and web. Batch jobs can be executed at any time and can be restarted if necessary. Web jobs are created when users submit BLAST jobs via the UI Local Blast Web Interface. These are time-critical and should therefore have priority over batch jobs. Ideally, if a web job arrives and all of the compute nodes are busy executing batch jobs, one of the batch jobs should be rescheduled and the web job should take its place on the compute node. If possible, the batch job should be check pointed before being interrupted so that no work is lost.

In the current implementation, we have opted for an approach in which only 75% of the compute nodes are allowed to execute batch jobs. The remaining 25% of the compute nodes are always available for time-critical web jobs. If there are no batch jobs

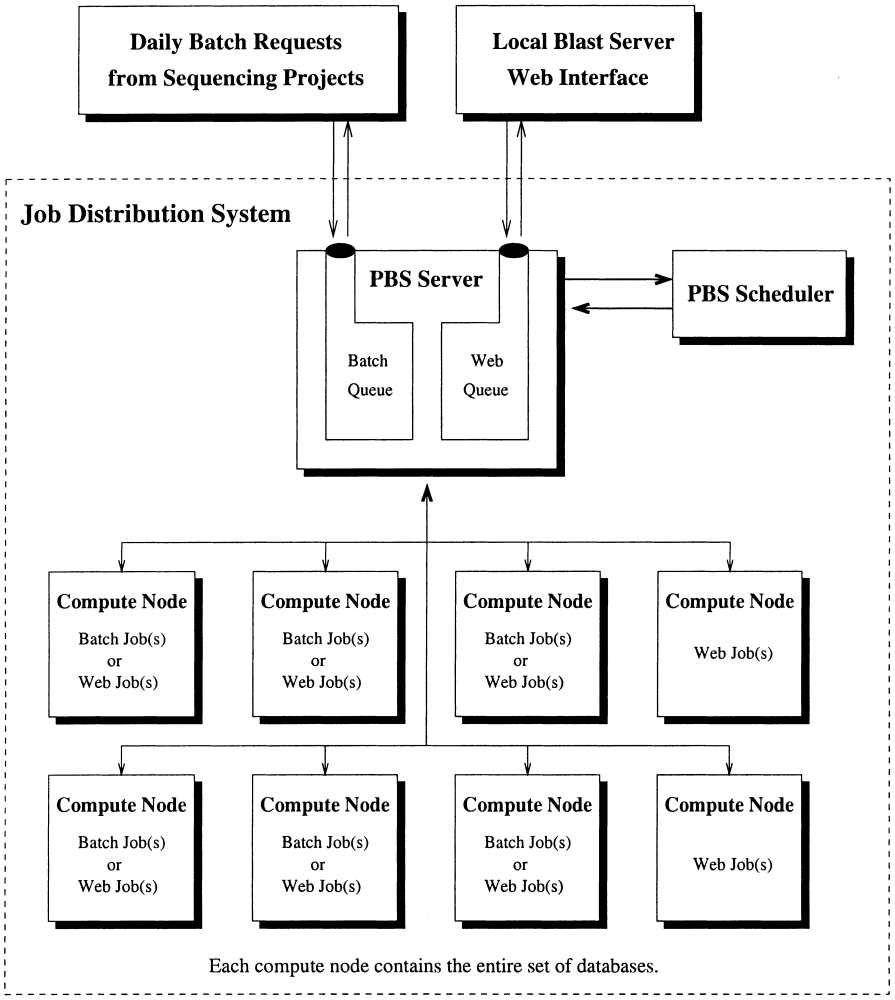


Fig. 3. Software architecture of current implementation.

executing, then 100% of the computing resources are available for web jobs. Although this approach limits the overall throughput for both web and batch jobs under heavy loads, it means that neither will ever be starved of resources. Although this was meant to be a short-term implementation, our experience to date has led us to maintain this as our continuing production configuration.

One of the benefits of using PBS is that it is relatively easy to write a custom scheduling algorithm. We are currently developing a scheduling policy that interrupts executing batch BLAST jobs when web jobs arrive. In addition, once a more fine-grained parallel BLAST has been deployed, it will be possible to speedup individual web jobs by allocating them to a group of compute nodes. PBS allows for this flexibility. There are a number of different scheduling policies and database "chunk" distribution strategies that we are evaluating.

4.3. Database updates and performance tuning

It is critical that all of the replicated databases on the compute nodes in the Local BLAST Server are

updated periodically to reflect the most recent contents of the globally shared databases. In addition, it is also necessary to assure that all nodes' copies of the database(s) are consistent with each other. Otherwise, the results obtained from a BLAST query would depend upon the compute node on which it was executed. In the current implementation, we have chosen to replicate databases in their entirety on each of the compute nodes. This is a trade-off between performance and the ease of maintaining database consistency. At the time of implementation, it was decided that using some sort of networked file system would be too large a bandwidth bottleneck given the large database sizes. However, the databases themselves are now reaching sizes that are making full replication less attractive. These databases not only consume (and require) large amounts of local disk space on each node, but they also require large amounts of internet, and intranet bandwidth at the time of updating. The intranet load is particularly problematic. If the local bandwidth consumed to update replicated databases becomes substantial, it will eventually interfere with the ability of the system to serve actual requests for BLAST service from the PBS scheduler.

As an alternative, with faster commodity networks such as Gigabit Ethernet, and with large file system caches on each node, the performance penalty of a networked file system solution may not be as large now as it was when that initial decision was taken. Such a network has recently been put into production at the Coordinated Laboratory for Computational Genomics at the University of Iowa. This may accelerate consideration of a change in strategies, as computing attributes which were once a liability become a strength. A hybrid solution which we are pursuing, is to have several *I/O servers* in the system, each with a complete copy of the database set in a switched partition of the intranet. Compute nodes would rely on these I/O servers for access to the databases. As long as the ratio of compute nodes to I/O nodes is below a computed threshold, the performance loss should be negligible. As compute nodes are added to the Local BLAST Server, such a system will be necessary since the network traffic caused by database updates will become excessive. These issues, as well as others, will need to be addressed continuously as networked architectures for local intranet BLAST service continue to evolve.

4.4. Medium-grained implementation

The partitioned database implementation utilizes many of the scheduling and interface concepts developed for the course-grained implementation, but adds a new set of tools to deal with the more complex logistics and operations.

The PBS based scheduler remains in place, and is capable of dealing with the multiple jobs required to complete a single BLAST query. By specifying what nodes have what segments of the database, the scheduler can select the proper machines to execute the queries on. If the database were split into four segments, a job submission to PBS requesting it to be run on a node with each segment would result in four BLAST outputs.

These outputs then must be combined into a single output file which should appear identical to that which would be produced by a single node with the complete database. This non-trivial merge operation is the centerpiece of this implementation. Once the results have all been received, the merge program must parse, sort, and correct the data from the nodes.

This merge operation is accomplished through software developed for this purpose at the University of Iowa. Written in PERL, the software accepts *n*-input files and performs the merge operation using only the data from these files. No additional processing or outside data is necessary, and processing time for this operation is negligible.

Since the queries are naturally divisible, no expensive computations are required during the merge. First, the *n*-input files are parsed by section. The total number of sequences and letters in the database is calculated as the sum of the numbers indicated by the input files. The summary and detailed match sections are then sorted by score. Finally, the *E* values are corrected to reflect the larger combined database size. This is a simple operation based on the ratio of current database to combined database size. Finally, the statistics at bottom are corrected and the output file is created. Except for insignificant ordering differences, the output exactly matches that which would have been created by a single node, single database query. This final output can then be saved to disk in batch mode or delivered to the user in interactive, single query mode.

5. Status and discussion

The Coordinated Laboratory for Computational Genomics at the University of Iowa currently operates 10 sequencing systems, three shifts per day, with 96 sequencing lanes each. At full capacity, 2880 sequences are generated each day. Of these, approximately 80% would be expected to pass through an initial verification step and move on to be BLASTed. The length of a typical sequence is about 450 bases.

The daily sequence dataset has BLAST run on it three times. First, `blastn` is run against the NCBI non-redundant nucleotide database *nt* (404 657 sequences). Second, `blastx` is run against the NCBI non-redundant peptide database *nr* (356 412 sequences). Finally, `tblastx` is run against the NCBI non-redundant nucleotide database *dbest* (2 119 879 sequences). The `blastn` and `blastx` queries are relatively quick and take a minute at most. The `tblastx` queries take substantially longer — averaging about 15 min each.

For a daily load of 2310 sequences, running `tblastx` alone would take approximately 576 h (over 3 weeks) on a single CPU. Clearly, there is a need for some level of parallelism. In the current implementation, there are 25 CPUs in the Local BLAST Server. This can handle the maximum daily load, requiring about 20 h to run the `tblastx` program. As the number of sequences generated each day increases, it will be a simple matter to add compute nodes to the Local BLAST Server.

Thus far, the compute load generated by the web interface to the Local BLAST Server has been negligible compared to the batch service load. This trend is likely to continue in the future because users generally run quick programs such as `blastn` and `blastp` when using this interface. If the load were to increase significantly, it would again be straight forward to modify the scheduling policy to favor web jobs more. Alternatively, more compute nodes could be purchased.

6. Conclusion

As projects such as the Human Genome Project near completion, the size of sequence databases and the frequency of BLAST queries against them will grow exponentially. In this paper, we have discussed three ways to exploit parallelism in BLAST searches. A coarse-grained approach has been described that is currently in production use at the University of Iowa. Without such a system, it would be impossible to meet the daily computational demands of BLAST searches. Work on a medium-grained solution is nearing completion and is scheduled to be put in production in December 1999. A fine-grained solution has been outlined for future research.

References

[1] Deparment of Energy, Five years of progress in the Human Genome Project, Human Genome News 7 (3/4) (1995). Available at www.ornl.gov in Tech-Resources/Human_Genome/publicat/hgn/v7n3/04progre.html (September, 1997).

[2] J.A. Blake, J.E. Richardson, M.T. Davisson, J.T. Eppig, The Mouse Genome Database (MGD), a comprehensive public resource of genetic, phenotypic and genomic data, Nucleic Acids Res. 25 (1) (1997) 85–91.

[3] M. Berks, The C. elegans genome sequencing project, Genome Res. 5 (1995) 99–104.

[4] R.W. Hockney, C.R. Jesshope, Parallel Computers 2: Architecture, Programming, and Algorithms, IOP Publishing, Philadelphia, PA, 1988.

[5] S. Altschul, T. Madden, A. Schäffer, J. Zhang, Z. Zhang, W. Miller, D. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, Nucleic Acids Res. 25 (1997) 3389–3402.

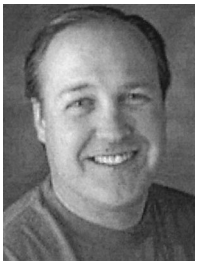
[6] T.E. Scheetz, C.L. Birkett, T.A. Braun, D. Nishimura, V.C. Sheffield, M.B. Soares, T.L. Casavant, Informatics for preparation of EST reads in a mixed-tissue cDNA library setting, in: Proceedings of the 1998 Meeting on Genome Mapping, Sequencing, and Biology, Cold Spring Harbor, New York, Departments of Electrical and Computer Engineering, Pediatrics, Physiology and Biophysics, University of Iowa, Iowa City, IA, 205 pp.

[7] M.B. Soares, G. Beck, B. Berger, C.L. Birkett, E.A. Black, M.F. Bonaldo, R.C. Braun, T.A. Braun, M. Donahue, S. Kaliannan, R. Kincaid, V. Miljokovic, K.J. Munn, D. Nishimura, K.T. Pedretti, T.E. Scheetz, L.H. Stier, T.L. Casavant, V.C. Sheffield, A program for rat gene discovery and mapping, in: Proceedings of the 1998 Meeting on Genome Mapping, Sequencing, and Biology, Cold Spring Harbor, New York, Departments of Electrical and Computer Engineering, Pediatrics, Physiology and Biophysics, University of Iowa, Iowa City, IA, 212 pp.

[8] D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Under the Portable Batch System, Lecture Notes in Computer Science, Vol. 949, Springer, Berlin, 1995.

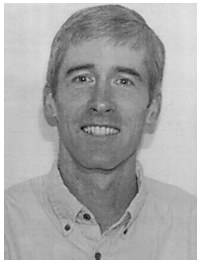
Ryan Carl Braun received his BS degree in Electrical Engineering from the University of Iowa in 1999. During his undergraduate program, he performed research in the Coordinated Laboratory for Computational Genomics. He is currently employed by the Enterprise Systems Group of Dell Computer Corporation. His research interests include parallel and distributed processing, computer networking, and operating systems.

Kevin Thomas Pedretti received his BS degree in Electrical Engineering from the University of Iowa in 1999. He is currently enrolled in the Electrical and Computer Engineering graduate program at the University of Iowa and performs research at the Coordinated Laboratory for Computational Genomics. His research interests include parallel and distributed processing, Beowulf-style commodity computing, and DNA sequence clustering and analysis.



Thomas Lee Casavant received his BS degree in Computer Science, his MS degree in Electrical and Computer Engineering, and his PhD degree in Electrical and Computer Engineering from the University of Iowa in 1986. From 1986 to 1989 he was with the School of Electrical Engineering at Purdue University. In 1989, he joined as the faculty of the University of Iowa where he is presently a Professor of Electrical and Computer Engineering, and Director of the Coordinated Laboratory for Computational Genomics and the Parallel Processing Laboratory. Dr. Casavant has published over 70 technical papers on parallel and distributed computing, and computational genomics, and has presented his work at tutorials, invited lectures, and conferences in the United States, Asia and Europe. Dr. Casavant is a senior member of the Institute of Electrical and Electronics Engineering (IEEE). He has served on the editorial boards of IEEE Transactions on Parallel and Distributed Processing and the Journal of Parallel and Distributed Computing (JPDC). His current research interests include large-scale methods for gene discovery, mapping, and disease gene identification, parallel computer architecture, scheduling, trace recovery, and visualization.

Todd Edward Scheetz received his BS degree in Electrical Engineering in 1993 and his MS degree in Electrical and Computer Engineering from the University of Iowa in 1995. He is currently enrolled in the PhD program in Genetics at the University of Iowa. Research interests include bioinformatics, parallel and distributed processing, and operating systems.



Clay Birkett received his BS in Electrical Engineering from Virginia Technology in 1985. He worked as a Biomedical Engineer for 2 years at the Veterans Administration Hospital of Richmond, VA. In 1987 he moved to Iowa City, IA, where he worked for 8 years as a Biomedical Engineer in the Department of Cardiology at the University of Iowa Hospital. In the Cardiology Department he did research in nerve recording analysis, cardiac defibrillation, and intra-arterial imaging. In 1991 he received his MS in Biomedical Engineering from the University of Iowa. Since 1997 he has worked as a Senior Programmer Analyst at the University of Iowa in the Department of Electrical Engineering. He is currently working in the Computational Genomics Laboratory on gene discovery and mapping projects.

Chad Andrew Roberts received his BA degree in Mathematics from the University of Iowa in 1993. He is currently a staff member at the Coordinated Laboratory for Computation Genomics at the University of Iowa.