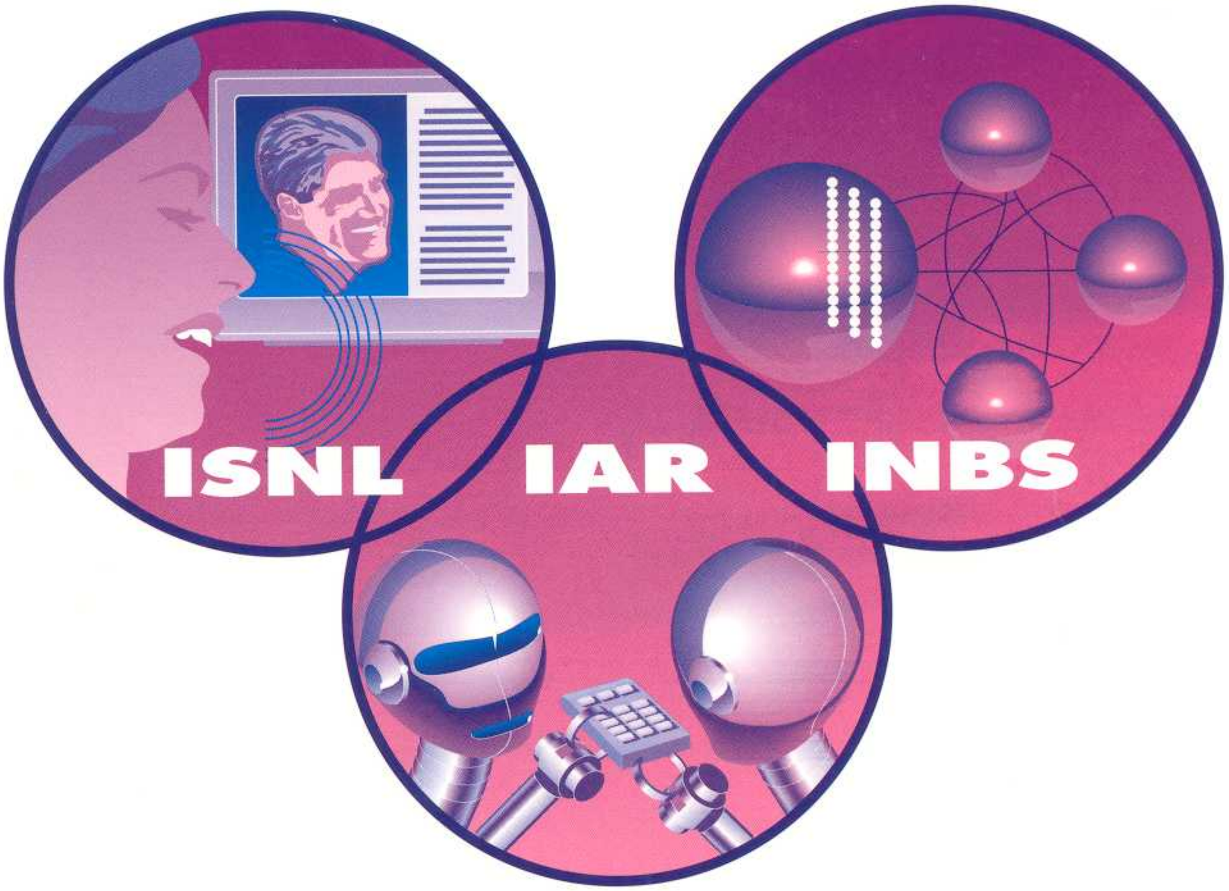
PROCEEDINGS

1999 IEEE INTERNATIONAL CONFERENCE ON

INFORMATION INTELLIGENCE AND SYSTEMS



October 31–November 3, 1999
Bethesda, Maryland

Sponsored by The IEEE Computer Society

IEEE COMPUTER SOCIETY

In the Proceedings of The International Conference of Information, Intelligence and Systems, Washington D. C., November 1999, pp. 48 – 55

A Distributed Scheme for Efficient Pair-wise Comparison of Complete Genomes

Valerian S. Anderson and Arvind K. Bansal[1]
Kent State University
Kent, Ohio 44242, USA
{vanderso, arvind}@mcs.kent.edu

Abstract

The comparisons of newly sequenced genomes against a genome with known functionality of genes provide important clues to the structure and function of genes and identification of metabolic pathways in newly sequenced organisms. New and more complex organisms are being added to biological databases at an increasing rate. Time-efficient, automated computational methods are needed to analyze the increasing amount of data in realistic time. This paper describes a distributed technique and a CORBA-based implementation to compare and align gene sequences in large complete genomes, using multiple heterogeneous distributed processors on a distributed network. The performance evaluation suggests that the distributed technique can significantly reduce the computational time.

Keywords: CORBA, cluster computing, distributed computing, functional genomics, genome comparison

1. Introduction

Genome comparison is an important technique to identify the functionality of genes [3, 8, 10, 14], which is necessary to map metabolic pathways [11, 13] of newly sequenced organisms. The identification of gene function and metabolic pathways and their variations will facilitate the identification of the cause of diseases.

The first microbial genome was sequenced [7] in late 1995. Since then, 26 genomes have been completely sequenced and archived at the National Center for Biotechnology Information (ftp://ncbi.nlm.nih.gov/genbank/genomes), and more are underway. It is anticipated that a major portion of the human genome will be sequenced by the year 2002. As genomes of complex organisms become available, the computational demands for their analysis are becoming unmanageable.

A genome is modeled as an ordered set of genes. Each gene is a sequence of nucleotide base-pairs which translate to proteins — sequences of amino acid characters. A single gene ranges in length from less than 100 to more than 1000 amino-acid characters. The comparison of a pair of complete genomes entails a comparison of gene-pairs in the Cartesian product of the two sets of genes that represent two genomes. BLAST [2] — an approximate string matching technique based on segment matching — is an efficient technique for similarity matching, but more accurate alignments are provided by the Smith-Waterman algorithm [15, 16] — a dynamic matrix technique. Accurate alignment is necessary to identify variations in the structure and functions of similar genes.

A technique to improve the efficiency of genome comparison while preserving accuracy provided by the Smith-Waterman algorithm is to first use BLAST to prune out dissimilar gene-pairs in genomes and then align the filtered gene-pairs using the Smith-Waterman algorithm on the filtered gene-pairs. Since the number of genes similar to a given gene after the BLAST phase is bounded by a small constant c (usually 6) for most of the genes, this technique reduces the time complexity of a genome comparison based upon the Smith-Waterman algorithm by a factor of M/c where M is the number of genes in a genome.

Current techniques for the automated comparison of genomes are sequential, and suitable for small to medium sized microbial genomes [3, 4]. There are approximately 4000 gene-pairs after the BLAST filtering for medium size microbes with 1600 genes. The current sequential method [3] takes approximately 5000 seconds

[1] Corresponding author

on current-day desktops to compare two complete microbial genomes with about two million amino-acid characters and around 1600 genes. The sequential scheme is still quite slow, and is not suitable for higher order organisms, which have up to 50 times more genes than medium sized microbes. It is estimated that the human genome has around 80,000 proteins.

Distributed computing has recently emerged as a means of speeding up computation due to the fast Internet connectivity and the drastic drop in the processor cost. As Internet connections become faster in the next couple of years, a cluster of thousands of inexpensive processors over the Internet will work as a super fast virtual computer suitable for distributing coarse grain independent subtasks on individual processors.

The comparison of genomes is suitable for distributed computing because each gene comparison in BLAST phase or gene alignment in the Smith-Waterman phase is a coarse grain independent task — the work involved takes on the order of a few milliseconds for small genes and about one second for large genes. Communication overhead does not pose a concern since communication time between processes is much smaller compared to string comparison time.

In this paper, we apply distributed cluster of machines as one large virtual machine, and use the Common Object Request Broker Architecture (CORBA) [12] to map multiple processes to a heterogeneous set of architectures and operating systems. The distributed architecture exploits two kinds of concurrency as follows.

First, coarse grain pipelining is exploited between the BLAST phase and the Smith-Waterman phase of genome comparisons. Very similar gene-pairs in the BLAST phase are sent through a pipeline incrementally to the Smith-Waterman phase. This pipelining enables the BLAST phase and the Smith-Waterman phase to run concurrently. The interleaving of the execution removes the effect of data-transfer overhead between the two phases.

Second, the BLAST and the Smith-Waterman phases spawn independent processes for the concurrent comparison of genes. This task is scalable, as the comparisons are independent of each other. The execution time efficiency is linearly proportional to the number of processors, and is restricted only by the number of genes in the genomes.

A prototype software has been implemented, and is suitable for comparing large genome sequences. The software can be easily extended to compare multiple complete genomes in a time-efficient manner on a distributed network of heterogeneous architectures.

The major contribution of this paper is that we have applied the concept of distributed computing to the grand challenge problem of effectively identifying the functionality of higher organisms in realistic time. The model and the software are easily scalable due to the use of CORBA, which scales to any size of network. The system can harness thousands of inexpensive computers over the Internet, improving the performance/price ratio to solve a grand challenge problem.

The remainder of this paper is organized as follows: Section 2 briefly describes the necessary background needed to compare gene sequences, distributed computing, and CORBA. Section 3 presents the overall scheme of a distributed technique. Section 4 details the execution of various components of the distributed system. Section 5 describes a CORBA implementation of the distributed system. Section 6 analyzes the performance of the system. In the last section, we conclude the paper.

2. Background

In this section, we describe background concepts related to genome organization, similarity analysis of gene-pairs, alignment of gene-pairs, distributed computing, and CORBA.

2.1. Genome and similarity analysis

A genome is a set of interacting genes, and encodes the blueprint of an organism [1]. Each gene is a sequence of four types of DNA (deoxyribonucleic acid) molecules, collectively called nucleotides. These nucleotides make up the genes of an organism, which in turn are translated to proteins, which are sequences of 20 types of amino acids.

BLAST [2] is a popular and a time-efficient method for identifying genes in a database (of genes) which are similar to a given gene. A single BLAST comparison against a database of genes identifies all the genes in the database similar to a given gene. The BLAST software looks for the matching segments in the given sequence and a database of sequences, evaluates the statistical significance of matches, and reports the matches that satisfy a user-defined threshold of significance. However, the BLAST software does not produce accurate alignments.

Accurate alignments are generated by the *Smith-Waterman* algorithm [15]. The algorithm dynamically builds a matrix, with one of the sequences forming the top edge and the other the left edge. Comparisons are started at the top left corner of the matrix. At each row/column intersection of the matrix, a score for the comparison of the two intersecting letters is recorded. The similarity score between two sequences is defined as the cumulative sum of all individual elementary similarities: matches, mismatches, deletions and insertions.

2.2. Distributed systems

Distributed systems distribute coarse grain independent tasks [9] — tasks taking at least more than few hundred milliseconds on one processing unit — on a distributed set of processors. Independent tasks have little data transfer overhead, and distributing the subtasks provides a nearly linear speed up. Fast interconnections can harness a large number of inexpensive distributed processors as a massive parallel virtual computer to give a better performance/price ratio.

2.3. CORBA preliminaries

The Common Object Request Broker Architecture (CORBA) [12] provides a seamless integration of distributed objects in a loose client server environment. CORBA is a generic client/server system that maps an object-oriented computation and communication between various objects on a heterogeneous set of processors in a user transparent way. CORBA objects behave as both clients and servers. The architecture is isolated from the actual transport protocols thereby allowing an open-ended standard.

3. A distributed architecture

In this section we describe the overall scheme of the distributed genome comparison. The distributed architecture is generic, and can be easily extended for further processing after genome comparison.

3.1. Generic template

The system consists of a set of loosely connected computers — computers can be either on the same local network or somewhere on the Internet. The environment can be heterogeneous — computers with different data formats, operating systems, and architectures can be in the system at the same time. The software is composed of objects, which correspond to the responsibilities and data requirements of each phase. The system maps objects to a heterogeneous, distributed environment by assigning different objects to possibly separate machines, as shown in Figure 1.

A *coordinator object*

i. spawns and interacts with various objects in the corresponding phase,
ii. manages the queues needed to hold the information generated by the server objects.
iii. monitors the size of the queue to regulate the spawning of *server objects*. Regulation of *server objects* optimizes the comparison of genes in the BLAST phase by keeping the size of the BLAST queue smaller while avoiding the number of idle *server objects* — processes used to compare gene-pairs — in the Smith-Waterman phase.

A queue object stores relevant information about processed gene-pairs for each phase, and provides the communication facilities between the phases. A *queue object* and the corresponding *coordinator object* are on the same processor.

A server object passes sequence data to BLAST software for the BLAST phase or Smith-Waterman software for the Smith-Waterman phase, and collects corresponding results, and passes the data to *data-handling objects* to transfer data to the next phase.

There are three types of data handling objects as follows. *A collection object* collects data from server objects, and transfers the data to a *queue object* for next processing stages. *A data-sink object* collects data from *serve objects* of Smith-Waterman phase, and writes in a central file in a sorted manner. A g*enome object* selects data from an indexed file of genes to pass to various *server objects* of processing phases. In that way, a database can be accessed remotely whether on a local network, or over the Internet.

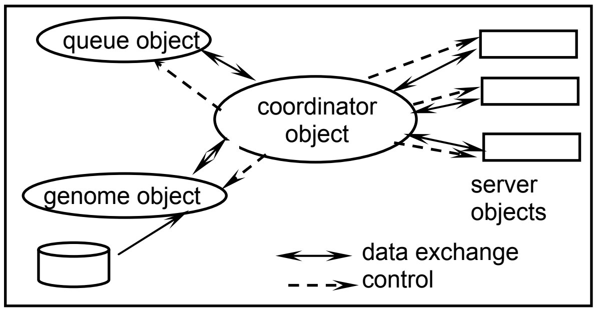


Figure 1. Object organization in each phase

3.2. The overall scheme

There are two phases in the system: the BLAST phase and the Smith-Waterman phase. Each phase consists of a *coordinator object*, a *queue object*, *server objects*, and a *genome object*. In addition there is a *data-sink object* controlled by the *coordinator object* of the last phase. The Smith-Waterman phase is the last stage in our case. Typically, a *coordinator object* and the corresponding *queue object* reside on the same machine, while the *server objects*, *genome objects*, and *data-sink objects* may be mapped to separate machines. The over all scheme is illustrated in Figure 2.

The BLAST *coordinator object* starts a *genome object*, BLAST *server objects*, and a BLAST *queue object*. The *genome object* constructs a map of genes and their positions in the file. The purpose of the map is to reduce searching time by indexing the corresponding protein sequence by name. Since sequences are requested several times, this saves considerable time.

The gene comparison process starts after the *genome object* has built an indexed map of genes in the genomes. The *coordinator object* requests the corresponding *genome objects* for a gene-pair. The two sequences are sent to a *server object* for processing, and similarity score is obtained. The names of the sequences with similarity scores higher than a user defined threshold are sent along with their scores to a *queue object*. The *coordinator object* requests the *genome object* to send the next pair of sequences. The process continues until all the gene pairs are consumed.

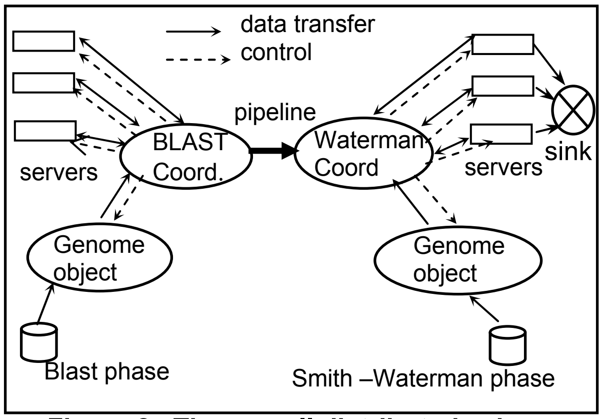


Figure 2. The overall distributed scheme

The BLAST *queue object* stores the name of filtered gene-pairs along with the similarity scores after the BLAST comparisons. The Smith-Waterman *queue object* stores the names of the filtered gene-pairs before they are transferred to the Smith-Waterman *genome object*. When the BLAST *queue object* has a predetermined number of filtered gene-pairs, the BLAST *coordinator object* moves the entries over to the Smith-Waterman *queue object*. Finally, the data is transferred to the *data-sink object*, which writes the names of the aligned sequences and the corresponding alignment scores in a centralized file. The *data-sink object* sorts the data according to the order of genes in a genome for better comprehension.

3.3. Load balancing

Each *coordinator object* continuously monitors the number of entries in its *queue object*. If the number of filtered gene-pairs is above a threshold value in the BLAST *queue object*, then a BLAST *server object* is released to the processor pool, and the capability-maps in both the *coordinator objects* are updated. At least one BLAST *server object* is always active. If the number of filtered gene-pairs in the BLAST output *queue object* falls below a certain threshold level, more BLAST *server objects* are spawned, if available. The maps in the *coordinator objects* are updated before spawning the processor. A Smith-Waterman *coordinator object* releases a *server object* if the number of gene-pairs is zero. After the termination of the BLAST phase, the Smith-Waterman phase captures all the available processors to spawn *server objects*.

4. Execution of distributed objects

This section details the execution of the coordinator objects and the termination conditions for the BLAST phase and the Smith-Waterman phase.

4.1. Execution of coordinator objects

The coordinators initiate execution by starting *server-objects*, *queue objects*, and *data-handling objects*. During execution, the *server objects* carry out the gene-pair comparisons. The *coordinator objects* (see Figures 3a and 3b) tie the system together by monitoring the load, dynamically spawning or releasing the *server objects* to balance the load, spawning *data-handling objects* to store and transfer results, and checking for the termination conditions of the corresponding phase.

```
BLAST coordinator-object
Input: Capability map of machines in the cluster;
Output: A list of filtered gene-pairs with similarity scores;
Internal processing:
{
start genome-object to start picking the gene-pairs;
start server-objects according to capability map;
repeat
  check for termination conditions of the BLAST phase;
  if (not terminating conditions) {
     receive sequence name pairs and similarity scores from
     BLAST  server objects;
  if (similarity-score > threshold)
     store name pairs in BLAST queue object;
     transfer name pairs to Smith-Waterman queue object;
  if (queue_count(BLAST) > max_threshold
     reduce BLAST server objects;
  else if (queue_count(BLAST) < min_threshold) and
        (are_available(servers))
     spawn more BLAST server objects;}
until (termination conditions for the BLAST phase)}
```

Figure 3a. A BLAST coordinator

4.2. Terminating conditions

There are two conditions for the BLAST coordinator object to complete the BLAST phase:

i. the genome object has collected all the gene-pairs, and

ii. No more data remains be collected from the server objects

The BLAST coordinator is not terminated even after the BLAST phase has processed all the genes. The rationale is that since the *queue object* is a part of the BLAST *coordinator object,* and termination of the *coordinator object* will terminate the transfer from BLAST *queue object* to Smith-Waterman phase prematurely terminating the Smith-Waterman phase.

In order for the BLAST phase to terminate, both the BLAST coordinator and Smith-Waterman coordinator keep corresponding counters to keep track of the processed gene-pairs. In addition to the above described two conditions, the counter *gene_pair_count(BLAST) must* be equal to *gene_pair_count(Smith-Waterman) + queue_ count(Smith-Waterman)* where *queue_count(Smith-Waterman)* represents the number of the gene-pairs in the Smith-Waterman queue-object still to be processed by the Smith-Waterman phase.

```
Smith-Waterman coordinator-object
Input:  1. Capability map of machines in the cluster;
        2. Filtered gene-pairs from the BLAST phase;
Output: A list of filtered gene-pairs with alignment scores;
Internal processing:
{
start genome object to pick the gene-pairs from genomes;
start queue object to pick up filtered gene-pairs from the pipe;
start server objects according to capability map of servers;
repeat
  check sequence names in Smith-Waterman queue-object;
  check the Smith-Waterman phase termination conditions;
  if (not terminating-conditions){
    if (queue_count(Smith-Waterman) == 0)
          reduce Smith-Waterman server objects;
    else if (queue_count(Smith-Waterman) >
              max_threshold and are_available(servers))
          spawn more Smith-Waterman server objects;
    if (queue_count(Smith-Waterman) > 0)
        transfer gene-pair names to genome object for the
        sequence pickup;}
until (the termination conditions)
}
```

Figure 3b. A Smith-Waterman coordinator

The conditions for the completion and termination of the Smith-Waterman phase include all the conditions for the termination of the BLAST phase and the conditions that both *queue_count*(*Smith-Waterman*) and *queue_ count*(*Smith- Waterman*) are zero. The final terminating conditions for the distributed system are that the BLAST phase has terminated, and there are no more data to be collected from the Smith-Waterman servers. The following equations should hold:

1. *genePairCount*(BLAST) = *genePairCount*(Smith-Waterman)
2. *queueCount*(Smith-Waterman) = 0.

The last two conditions ensure that all the data have been collected by the *data-sink object* into a sorted file before the distributed system terminates.

5. A CORBA based implementation

The system obtains the initial machine configuration from a table, which maps machines to capabilities. The *coordinator objects* are started on the appropriate systems by Unix scripts that log into the appropriate machine and start the appropriate processes. The version of CORBA used for this project does not include, as an integral CORBA service, the capability of starting and stopping processes on other machines. The software produced for this project implemented this capability through a *server object*, which starts and stops processes through system calls. CORBA provides a framework for interfacing with system calls on foreign machines.

After the *coordinator objects* have been started, the process is automatic. The appropriate *server objects* are contacted via the CORBA name service, a basic service provided by CORBA. Objects on separate machines are referred to by a unique name, which renders their functionality transparent to objects on other machines.

Figure 4 shows an example of code in the BLAST *coordinator object* to obtain foreign references to the *queue objects*.

```
Queue_var blastQueue, smithQueue;
CosNaming_Name, bCoordName, blastQueueName,
smithQueueName;
try {// construct name of BLAST queue-object
blastQueueName.length(1);
blastQueueName[0].id = CORBA_string_dup("blastqueue");
blastQueueName[0].kind = CORBA_string_dup("");
// obtain reference for BLAST queue object
CORBA_Object_var object = context->resolve(blastQueue
Name);
assert(!CORBA_is_nil(object));
//make the generic queue object of a specific user-defined type
blastQueue = Queue::_narrow(object);
assert(!CORBA_is_nil( blastQueue ));
// construct name of Smith-Waterman queue object
smithQueueName.length(1);
smithQueueName[0].id=CORBA_string_dup("smithqueue");
smithQueueName[0].kind = CORBA_string_dup("");
//obtain reference for Smith-Waterman queue object
object = context->resolve(smithQueueName);
assert(!CORBA_is_nil(object));
// Make the queue object from generic user-defined type
smithQueue = Queue::_narrow(object);
assert(!CORBA_is_nil( smithQueue ));
catch {…..}
```

Figure 4. Code for foreign object interface

The code shows the construction of the BLAST *queue object* and Smith-Waterman *queue object*. First the name for the BLAST *queue object* is created using a CORBA naming variable. Then the function *context-resolve* obtains the reference for the BLAST queue-object. The process is repeated for the *queue object* in the Smith-Waterman phase.

Figure 5 shows a sample of code from the BLAST *coordinator object*, showing the startup of the BLAST *server objects* and primary processing loop. First, the *coordinator object* obtains a reference for each *server object* in the map of BLAST-capable machines, using the *context-resolve* mechanism as explained above. Then, a BLAST *server object* is started on each machine via the *startProcess* method of the process server. Next, the main processing loop begins to transfer sequence pairs to the Smith-Waterman phase.

Transfer continues until the termination conditions are met: the BLAST *server objects* have finished, and the number of gene-pairs processed by the Smith-Waterman phase equals the number processed by the BLAST phase. The BLAST *coordinator object* monitors the size of the BLAST *queue object*. The *coordinator object* attempts to start a process if the queue-count is below the minimum threshold, or attempts to stop a process if the queue-count is above the maximum threshold.

```
// start the BLAST servers as indicated by process map
for (curMachine = map.begin( ); curMachine != map.end( );
curMachine++) {
CORBA_Object_var object = context-> resolve(curMachine.
first ( ) );
Machine_var machine = Machine::_narrow(object);
machine->startProcess( );}
// start the processing loop
while (! blastQueue->areBlastsDone( ) &&
smithQueue->getSmithCount( ) != blastQueue-> getBlast-
Count( ) )
{//blast pairs pushed into queue by BLAST servers
int size = blastQueue->size( );
if (size >= SIZE || count % 10 == 0)
    blastQueue->transferEntries(smithQueue, SIZE );
// balance the number of processors based upon number of
filtered gene-pairs
if ( size > max_threshold)
    success = blastProcessServer->reduce( );
else if (size < min_threshold)
    success = blastProcessServer->add( );
if (!success)
  diagFile << "Unable to load balance: code " << success <<
endl;
```

Figure 5. Execution of the BLAST coordinator

5.1. CORBA considerations

CORBA imposes a set of conventions on the programmer, and each version of CORBA interprets the standard (set by the Object Management Group, or OMG) slightly differently. Selection of a version of CORBA involves consideration of platform availability and language support. For this project, the stability of the naming service was crucial. C++ was the primary language utilized for the project due to its speed compared to other object based Internet languages.

String handling was a priority in this software. To handle memory management in CORBA is incumbent on the programmer — if a function returns a string value via an out or inout parameter, or as a return value, the function must duplicate, and the calling process must release, this value. To duplicate the value, CORBA provides the function, *CORBA_stringdup* ( ). CORBA provides two means for the programmer to release the memory: One explicitly calls the function CORBA_stringfree(), and the other is to assign the value to a variable of type CORBAstringvar, which releases the memory automatically.

6. Performance evaluation

We executed the prototype system on a cluster of three computers – a four processor SGI machine and two HP machines. The execution was done in a configuration when there was heavy load to simulate realistic conditions. We tested the configurations manually with the load balancing module off to get a performance evaluation chart. The performance of the four different configurations was tested as follows:

Configuration 1: One server-object in the BLAST phase and one server-object in the Smith-Waterman phase were mapped on the same computer SGI to remove any effect of the data-transfer overhead. Only pipelining was exploited between the BLAST phase and the Smith-Waterman phase.

Configuration 2: One server-object in the BLAST phase was mapped on an SGI machine, and one server-object in the Smith-Waterman phase was mapped on an HP machine. Only pipelining was exploited between the BLAST phase and the Smith-Waterman phase. Configuration 2 has overhead of transferring data between two different machines.

Configuration 3: Two server-objects in the BLAST phase were mapped on an SGI machine, and one server-object of Smith-Waterman was mapped on an HP machine. This configuration exploited concurrency in the BLAST phase, and pipelining between the BLAST phase and the Smith-Waterman phase.

Configuration 4: Two server-objects in the BLAST phase were mapped on the SGI machine, and two server-objects in the Smith-Waterman phase were mapped on two different HP computers. Concurrency was exploited in both phases, and pipelining was exploited between the BLAST phase and the Smith-Waterman phase.

Table I summarizes the results. The results show that the maximum execution time is controlled by the execution time of the BLAST phase. The data transfer overhead between the Smith-Waterman *server object* and the BLAST *server object* does not show up because of pipelining between the two phases. The comparison between Configurations I and III shows that the speed up is linear with the increase in BLAST server-objects. Spawning of more BLAST *server objects* feeds the gene-pairs in the pipeline faster resulting into the linear reduction in time.

Table 1: Performance evaluation

	Time in seconds			
Config.	20 genes	40 genes	50 genes	100 genes
1	107	221	253	510
2	137	231	258	508
3	69	144	173	343
4	72	144	173	345

7. Conclusion

A pair-wise, cross-species computational comparison of a newly sequenced genome is a cost-effective way to help identify gene functions. Such a comparison is an important step in the process of identifying disease genes.

We have presented a distributed scheme for the cross-species comparison of complete genomes. The scheme has three major advantages:

i. The scheme uses the efficiency of BLAST to prune out dissimilar genes before using accurate alignment algorithms.

ii. The linear speed up facilitated by this architecture allows the comparison of organisms previously considered too large in a realistic time.

iii. The system can harness thousands of inexpensive processors on the Internet and local area networks in a scalable manner.

Future work includes further automating machine selection, data partitioning, and developing a distributed algorithm for the identification of orthologous gene groups [3, 4, 6] needed to identify operons [14] — a group of genes having a well defined functionality within a metabolic pathway [11, 13].

Acknowledgements

This research was supported in part by the Ohio Board of Regent's Ph.D. enhancement program. The authors acknowledge Warren Gish for the BLAST software and Paul Hardy for the Smith Waterman software library. The authors also acknowledge Liquid Crystal Institute, Institute of Computational Mathematics, and the OCARNET project at Kent State University for the use of high performance equipment.

References

[1] Alberts, B., D. Bray, J. Lewis, M. Raff, K. Roberts, and J. D. Watson, *Molecular Biology of THE CELL,* Publishers: Garland Publishing, Inc., 1994

[2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic Alignment Search Tools," *J. Mol. Biol.*, 1990, Vol. 215, pp. 403-410

[3] A. K. Bansal, P. Bork, and P. Stuckey, "Automated Pair-wise Comparisons of Microbial Genomes," *Mathematical Modeling and Scientific Computing*, Principia Scientia, Vol. 9, 1998, pp. 1 - 23.

[4] A. K. Bansal and P. Bork, "Applying Logic Programming to Derive Novel Functional Information in Microbial Genomes," *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages, Springer Verlag, LNAI 1551,* 1999, pp. 274-289

[5] C. J. Bult, O. White, G. J. Olsen, et. al., "Complete Genome Sequence of the Methanogenic Archaeon, Methanococcus jannaschii," *Science*, 1996, Vol. 273, pp. 1067-1044

[6] W. M. Fitch, "Distinguishing Homologous from Analogous Proteins," *Systematic Zoology*, 1970, pp. 99-113

[7] R. D. Fleischmann, M. D. Adams, O. White, et. al., "Whole-Genome Random Sequencing and Assembly of Haemophilus influenzae Rd," *Science*, 1995, Vol. 269, pp. 496-512

[8] E. V. Koonin, A. R. Mushegian, M. Y. Galperin, and D. R. Walker, "Common and Distinctive Features of Archeal and Bacterial Genomes Revealed by Computer Analysis of Protein Sequences," *Molecular Microbiology*, 1997, Vol. 25, pp. 619-637.

[9] Lynch, N., *Distributed Algorithms*, San Francisco, CA: Morgan Kaufman Publishers, Inc., 1996

[10] A. R. Mushegian and E. V. Koonin, "A Minimal Gene Set for the Cellular Life Derived by Comparison of Bacterial Genomes," *Proc. Natl. Acad. Sci.,* USA, 1996, Vol. 93, pp. 10268 -10273

[11] E. Selkov, N. Maltsev, G. J. Olsen, R. Overbeek, and W. B. Whitman, "A Reconstruction of the Metabolism of Metanococcus jannaschi from Sequence Data," *Gene*, 1997, 197(1-2):GC 11-26

[12] Siegel, J., *CORBA Fundamentals and Programming*, New York, NY: John Wiley & Sons, 1996

[13] R. L. Tatusov, A. R. Mushegian, P. Bork et. al., "Metabolism and Evolution of Haemophilius Influenzae Deduced From a Whole-Genome Comparison with Escherichia Coli," *Current Biology*, 1996, Vol. 6, Issue 3, pp. 279-291

[14] A. Vitreschak, A. K. Bansal, and M. S. Gelfand, "Conserved RNA structures regulation initiation of translation of Escerichia coli and Haemophilus influenzae ribosomal protein operons*," First International Conference on Bioinformatics of Genome Regulation and Structure*, Novosibirsk, Russia, 1998, Vol 1, pp. 229, the detailed version to appear in *Biophysics* (in press)

[15] M. S. Waterman, "General Methods for Sequence Comparison," *Bull. Math. Biol*., 1984, Vol. 46, pp. 473-500

[16] Waterman, M. S., *Introduction to Computational Biology: Maps, Sequences, and Genomes*, Chapman & Hall, 1995