INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

A DESIGN OF A BLAST SERVER WITH JINI TECHNOLOGY

Liusong Yang

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

April 2002

Abstract

A Design of a BLAST Server with Jini Technology

Liusong Yang

The explosion of biological data, particularly the nucleic acid and protein sequences, has been proved to be both an opportunity and challenge for biologists as well as computer experts. The burgeoning of huge-sized sequence databases necessitates the invention and renovation of powerful computational tools to efficiently manage and utilize these sequence data. Among the various bioinformatics tools, BLAST (Basic Local Alignment Search Tool) is the most commonly used one for DNA sequence database searching. To enhance its working efficiency, in this thesis, we have tried to design a new distributed system for BLAST service with the recently developed Jini technology. Since the service is involved in sequence databases query, the RMI(Remote Method Invocation) proxy was chosen in its architecture. The structures on both the client and the server sides were designed mainly based on Java technology and demonstrated with class diagrams of Unified Modeling Language(UML).

Acknowledgments

I would like to express my sincere respect and gratitude to my thesis supervisor Dr Gregory Butler for his guidance, encouragement and support throughout the course of this research work.

I would like to thank my colleagues in my research group for their invaluable discussions, suggestions, comments and help on my thesis.

I would like to give a special thanks to my parents, my lovely son and my husband. It was their love and patience and continued support and encouragement that made this thesis possible.

Contents

List of Figures

Chapter 1

Introduction

The last few decades have seen an explosion in the number of DNA and protein sequences available through public databases. The major nucleotide sequence databases double in size approximately every 14 months, and the number of completely sequenced organism genomes is constantly increasing.

Sequence databases are typically searched using keywords or a sequence. Keyword search engines such as the Entrez system provided by the National Center for Biotechnology Information (NCBI) search the annotation section of the sequence database record. The major application of these databases involves sequence similarity searching, where sequences similar to a query sequence supplied by the investigator are identified in a sequence database. In most cases, this operation identifies potential homologues of the query sequence, that is, the database sequence sharing a common evolutionary history with the query sequence. Homologous sequences can provide a clue as to the function of the query sequence, its evolutionary history and its structure. For example, if an unknown protein sequence is found to be markedly similar over its whole length to the sequence of a protein of known structure in the database, the two proteins are likely to have similar structures and related functions. Orthologs and paralogs are two types of homologous sequences. Orthology describes genes in different species that derive from a common ancestor. Orthologous genes may or may not have the same function. Paralogy describes homologous genes within a single species

that diverged by gene duplication. Other applications include the identification of coding regions in uncharacterized genomic DNA by looking for DNA regions that can be translated into an amino acid sequence similar to that of a known protein [1].

BLAST is the most commonly used suite of programs for sequence database similarity searching. It allows rapid searching for sequences similar to a query sequence.

1.1 What is BLAST?

BLAST [2](Basic Local Alignment Search Tool) is a set of similarity search programs designed to explore the available sequence databases. One of the main reasons we are interested in the similarity between biomolecule sequences is that similar sequences hold a similar function, structure, and they have a similar evolutionary history. We assess sequence similarity with the concept of sequence alignment. An alignment is simply a correspondence between the sequences, in which each symbol in one sequence is assigned no more than one (maybe none) of the symbols in the other sequence, and in which the order of the symbols in the sequence is maintained. Often, a global alignment between two sequences has little biological meaning, for instance, when aligning two genomic regions containing a gene. In such cases, it may be more relevant to align those local regions in the sequence showing high similarity than to attempt to align the whole sequences.

1.2 What is Jini Network Technology?

Jini network technology provides a simple infrastructure for delivering services in a network and for creating spontaneous interaction between programs that use these services regardless of their hardware/software implementation.

Any kind of network made up of services (applications, databases, servers, devices, information systems, mobile appliances, storage, printers, etc.) and clients (requesters of services) of those services can be easily assembled, disassembled, and maintained

on the network using Jini Technology. Services can be added or removed from the network, and new clients can find existing services — all without administration [3].

Jini technology uses a lookup service with which devices and services register. When a device plugs in, it goes through an add-in protocol, called discovery and join-in. The device first locates the lookup service (discovery) and then uploads an object that implements all of its services' interfaces (join). To use a service, a person or a program locates it using the lookup service. The service's object is copied from the lookup service to the requesting device where it will be used. The lookup service acts as an intermediary to connect a client looking for a service with that service. Once the connection is made, the lookup service is not involved in any of the resulting interactions between that client and that service.

It does not matter where a service is implemented — compatibility is ensured because each service provides everything needed to interact with it. There is no central repository of drivers, or anything else for that matter.

The Java programming language is the key to making Jini technology work. Devices in a network employing Jini technology are tied together using the Java Remote Method Invocation (RMI). By using the Java programming language, the Jini network architecture is secure. The discovery and join protocols, as well as the lookup service, depend on the ability to move Java objects, including their code, between Java virtual machines.

Jini technology not only defines a set of protocols for discovery, join, and lookup, but also a leasing and transaction mechanism to provide resilience in a dynamic networked environment. The underlying technology and services architecture is powerful enough to build a fully distributed system on a network of workstations. Also the Jini network infrastructure is small enough that a community of devices enabled by Jini network software can be built out of the simplest devices. For example, it is entirely feasible to build such a device community out of home entertainment devices or a few cellular telephones with no "computer" in sight [4].

1.3 Problem Statement

Currently, there are the following ways to use BLAST service.

1) Local installation of BLAST software The widely used solutions for BLAST now require local installation of the analysis software, expertise in the UNIX operating system, and, in some cases, programming skill. These are skills and infrastructure that are frequently not present in the same groups or institutions that produce the DNA sequences. Thus, there is still a widespread need for a bioinformatics solution that removes the burden of providing hardware and software support that could otherwise detract from the core mission of lab oratories that specialize in DNA sequencing.

2) Web-based BLAST tool There are several web-based BLAST tools provided by NCBI, EBI and so on. This kind of BLAST service can be accessed by fixed URL. If the service changes its address, the users can not access it.

3) A distributed BLAST system with CORBA A distributed DNA sequence analysis system with CORBA was developed by J.T. Inman et al [5, 6]. Current versions of CORBA pass remote object references, rather than complete object instances. Each CORBA object lives within a server, and the object can only act within this server. This is more restricted than Jini, where an object can have instance data and class files sent to a remote location and execute there [7].

1.4 Contribution of the Thesis

In this thesis, we designed a BLAST server with Jini network technology. The service architecture chosen is the RMI thin proxy type. The structures on the client side and on the server side are shown with UML class diagrams.

1.5 The Organization of the Thesis

In the thesis, we first introduce the development of BLAST, including some different versions of the BLAST algorithm, the current public database structures and the parser frequently used now. Second, we introduce the Jini technology, including the general Jini architecture, the general structures of client side and server side. Then we introduce the design of our BLAST server with Jini technology. The last part is a summary and the future work of the design.

Chapter 2

BLAST

Dynamic Programming Algorithms are used for finding shortest paths in graphs, and in many other optimization problems, but in the comparison or alignment of strings (as in Biological DNA, RNA and protein sequence analysis, speech recognition and shape comparison). It provides a rigorous mathematical approach toward sequence alignment. It is guaranteed to find the best alignment between a pair of sequences given a particular choice of scoring matrix and gap penalties [8]. There are several variants of the dynamic programming algorithm that yield different kinds of alignments.

The rigorous dynamic programming algorithms for calculating the optimal global alignment similarity score between two protein or DNA sequences began with the heuristic homology algorithm of Needleman and Wunsch [9], which first introduced an iterative matrix method of calculation. Sellers [10] developed a true metric measure of the distance between sequences. The variant known as the Smith-Waterman algorithm [11] yields a local alignment. A local alignment aligns the pair of regions within the sequences that are most similar given the choice of scoring matrix and gap penalties. Database searches generally use local alignments as opposed to global alignment. This allows the database search to focus on the most highly conserved regions of the sequences without having to overcome interference from less well-conserved regions of the sequences. It also allows similar domains within sequences to be identified.

such as nucleotide binding domains, even though the sequences may not be related over their entire length. Smith and Waterman developed a dynamic programming algorithm for calculating local similarity scores.

The alignment may start and end anywhere in the two sequences, as long as it produces the best similarity score and allows for arbitrary length deletions and insertions. So the algorithm finds the most similar region shared by the two sequences in a way that reflects the minimum evolutionary distance in the similar region. This method is very slow. FASTA [12] is a fast approximation to Smith-Waterman. FASTA is a two step algorithm. The first step is a word search with a specific word size which finds regions in a two dimensional table that are likely to correspond to highly similar segments of the two sequences. These regions are a diagonal or a few closely spaced diagonals in the table that have a high number of identical word matches between the sequences. The second step is a Smith-Waterman alignment centered on the diagonals that correspond to the alignment of the highly similar sequence segments.

The region for the Smith-Waterman alignment is bounded by the window size. The window size limits the number of insertions or deletions one sequence can accumulate with respect to the other sequence in the alignment. Thus, the significant speedups observed in a FASTA search relative to a full Smith-Waterman search is due to the prior restriction in alignment space as well as skipping the Smith-Waterman step when no diagonals are found that correspond to alignments between highly similar sequence segments. The FASTA algorithm is a heuristic approximation to the Smith-Waterman algorithm. The heuristics used by FASTA allow it to run much faster than the Smith-Waterman algorithm but at the cost of some sensitivity. Two heuristics are employed. Both can be interpreted as restrictions on the model of sequence evolution that is used in comparing the sequences. The first restriction is implemented by the word size parameter, usually two for proteins and six for nucleic acids. This means that FASTA constrains the evolution between a pair of sequences to preserve a number of unchanged dipeptides or hexanucleotides. The second heuristic used by FASTA is the window size. Its effect is more variable than that of the word size. If

the best alignment lies entirely within the window defined by the window size and the concentrated identities, there is no effect. However if the best alignment, as defined by a full Smith-Waterman analysis, goes outside the window then a lower scoring alignment will be found by FASTA. This can lead the user to conclude that the sequences are not homologous when in fact they are and homology could have been inferred from a full Smith-Waterman alignment.

The first BLAST program is delivered by NCBI in 1990 [13]. The BLAST algorithm uses a word based heuristic similar to that of FASTA to approximate a simplification of the Smith-Waterman algorithm known as the maximal segment pairs algorithm. Maximal segment pairs alignments do not allow gaps and have the very valuable property that their statistics are well understood [14]. Thus, we can readily compute a significance probability for a maximal segment pair alignment.

2.1 BLAST Algorithm

Like other similarity measures, the BLAST begins with a matrix of similarity scores for all possible pairs of residues. Identities and conservative replacements have positive scores, while unlikely replacements have negative scores. For DNA sequence comparisons, identities are scored +5, and mismatches are scored -4. The similarity score for two aligned segments of the same length is the sum of the similarity values for each pair of aligned residues.

MSP, a maximal segment pair, is defined to be the highest scoring pair of identical length segments chosen from the two sequences. The boundaries of an MSP are chosen to maximize its score, so an MSP may be of any length. The MSP score, which BLAST heuristically attempts to calculate, provides a measure of local similarity for any pair of sequences. Because a molecular biologist may be interested in all conserved regions, not only in the highest scoring pair, a segment pair is defined to be local maximum if its score can not be improved either by extending or by shortening both segments. BLAST can seek all locally maximal segment pairs with scores above some cutoff.

Let a word pair be a segment pair of fixed length W. The main strategy of BLAST is to seek only segment pairs that contain a word pair with a score of at least a threshold word score T. Scanning through a sequence, one can determine quickly whether it contain a word pair of length w that can pair with the query sequence to produce a word pair with a score greater than or equal to the threshold T. Any such hit is extended to determine if it is contained within a segment pair whose score is greater than or equal to S (cutoff score). The lower the threshold T, the greater the chance that a segment pair with a score of at least S will contain a word pair with a score of at least T. A small value for T, however, increases the number of hits and therefore the execution time of the algorithm. Random simulation permits us to select a threshold T that balances these considerations.

In the implementations of this approach, the following are the details of the 3 algorithmic steps.

Step 1: Neighborhood Construction: compile a list of high scoring words This step is parameterized by the query sequence Q, the score function M, the word size W and the threshold word score T. This step outputs the neighborhood, N. The query sequence Q is scanned. Each query word may have zero or more neighbors. The set of neighbors of all query words is the neighborhood. The neighborhood is a set of tuples of the form (neighbor, offset) where neighbor is the word that matched the query word at offset.

Step 2: Hit Detection: scanning the database for hits This step is parameterized by a subject sequence S and the neighborhood N. A word hit H is an alignment of a query word and subject word. The subject S is scanned for matches to a member of the neighborhood. When a match is found, the extension step is invoked.

Step 3: Hit Extension This step is parameterized by the word hit H, the word size W, a scoring function M, the falloff score X, the threshold alignment score R

and the query and subject sequences Q and S. This step attempts to extend a hit into a longer, potentially higher-scoring alignment. The extension terminates if the cumulative score falls below X. The initial score is that of the word hit. Extension is first attempted in the left direction. Residue pairscores are accumulated in sum. If the sum is positive, it is added to score and reset to zero. The alignment is extended by one residue pair. If sum falls below X, extension terminates. The extension is then attempted in the right direction. The maximal positive-scoring alignment is stored in the HSP set if the score meets the threshold R.

The three steps may vary somewhat depending on whether the database contains protein or DNA sequences. For proteins, the list consists of all words that score at least T when compared to some word in query sequence. Thus, a query word may be represented by no words in the list or by many.

The price for being able to readily compute a significance probability is that the alignments cannot have gaps [14]. Thus, the evolutionary model requires that there be a fairly long stretch of sequence that has evolved without insertions or deletions, or at least with a complimentary pattern of insertions and deletions that do not significantly disrupt the alignment.

The BLAST algorithm is less sensitive than Smith-Waterman and in appropriate circumstances more selective. For proteins, the BLAST word based heuristic is more sensitive than the FASTA heuristic even though BLAST uses a word size of three for proteins while FASTA uses a word size of two.

2.2 The Variations of the Algorithm

After BLAST is created, it is modified and updated continuously. The most used versions will be introduced in the following.

2.2.1 Gapped BLAST

Gapped BLAST [15] is a variation of the original BLAST for protein database search programs, which was developed by NCBI in 1997. By using a new criterion for triggering the extension of word hits and a new heuristic for generating gapped alignments, it yields a gapped BLAST program that runs at approximately three times the speed of the original.

The major refinements to BLAST are the following:

For increased speed, the criterion for extending word pairs has been modified. The central idea of the BLAST algorithm is that a statistically significant alignment is likely to contain a high-scoring pair of aligned words. BLAST first scans the database for words that score at least T when aligned with some word within the query sequence. Any aligned word pair satisfying this condition is called a hit. The second step of the algorithm checks whether each hit lies within an alignment with score sufficient to be reported. This is done by extending a hit in both directions, until the running alignment's score has dropped more than X below the maximum score yet attained. This extension step is computationally quite costly: with the T and X parameters necessary to attain reasonable sensitivity to weak alignments, the extension step typically accounts for more than 90 percent of BLAST's execution time. It is therefore desirable to reduce the number of extensions performed.

The refined algorithm is based upon the observation that an HSP of interest is much longer than a single word pair, and may therefore entail multiple hits on the same diagonal and within a relatively short distance of one another. (This diagonal of a hit involving words starting at positions ($x1$,$x2$) of the database and query sequences may be defined as $x1$-$x2$. The distance between two hits in the same diagonal is the difference between their first coordinates). This signature may be used to locate HSPs more efficiently. Specifically, they choose a window length A, and invoke an extension only when two non-overlapping hits are found within distance A of one another on the same diagonal. Any hit that overlaps the most recent one is ignored. Efficient execution requires an array to record, for each diagonal, the first coordinate

of the most recent hit found. Since database sequences are scanned sequentially, this coordinate always increases for successive hits.

Because they require two hits rather than one to invoke an extension, the threshold parameter T must be lowered to retain comparable sensitivity. The effect is that many more single hits are found, but only a small fraction have an associated second hit on the same diagonal that triggers an extension. The great majority of hits may be dismissed after the minor calculation of looking up, for the appropriate diagonal, the coordinate of the most recent hit, checking whether it is within distance A of the current hit's coordinate, and finally replacing the old with the new coordinate. Empirically, the computation saved by requiring fewer extensions more than offsets the extra computation required for processing the larger number of hits.

The ability to generate gapped alignments has been added. The original BLAST program often finds several alignments involving a single database sequence which, when considered together, are statistically significant. Overlooking any one of these alignments can compromise the combined result. By introducing an algorithm for generating gapped alignments, it becomes necessary to find only one rather than all the ungapped alignments subsumed in a significant result. This allows the T parameter to be raised, increasing the speed of the initial database scan.

By seeking a single gapped alignment, rather than a collection of ungapped ones, only one of the constituent HSPs needs to be located for the combined result to be generated successfully. This means that we may tolerate a much higher chance of missing any single moderately scoring HSP.

In summary, the new gapped BLAST algorithm requires two non-overlapping hits of score at least T, within a distance A of one another, to invoke an ungapped extension of the second hit. The resulting gapped alignment is reported only if it has an E-value low enough to be of interest.

2.2.2 PSI BLAST

PSI BLAST [16], Position-Specific Iterated BLAST, is also a variety of original BLAST for protein database search programs, which was developed by NCBI in 1997. It introduces a method for automatically combining statistically significant alignments produced by BLAST into a position-specific score matrix, and searching the database using the matrix. It runs at approximately the same speed per iteration as gapped BLAST, but in many cases is much more sensitive to weak but biologically relevant sequence similarities.

BLAST searches may be iterated, with a position-specific score matrix generated from significant alignments found in round i used for round $i+1$. Motif or profile search methods frequently are much more sensitive than pairwise comparison methods at detecting distant relationships. However, creating a set of motifs or a profile that describes a protein family, and searching a database with them, typically has involved running several different programs, with substantial user intervention at various stages. The BLAST algorithm is easily generalized to use an arbitrary position-specific score matrix in place of a query sequence and associated substitution matrix. Accordingly, an automatic procedure is developed to generate such a matrix from the output produced by a BLAST search, and adapted the BLAST algorithm is adapted to take this matrix as input. The resulting program may not be as sensitive as the best available motif search programs, but its speed and ease of operation can bring the power of these methods into more common use.

2.2.3 PHI BLAST

The Pattern-Hit Initiated BLAST (PHI-BLAST) [16] is developed to address the problem in analyzing protein family homology. For example, a researcher frequently wishes to evaluate the significance of a specific pattern within a protein, or to exploit knowledge of known motifs to aid the recognition of greatly diverged but homologous family members. The PHI-BLAST program takes as input both a protein sequence

and a pattern of interest that it contains. It searches a protein database for other instances of the input pattern, and uses those found as seeds for the construction of local alignments to the query sequence. The random distribution of PHI-BLAST alignment scores is studied analytically and empirically. In many instances, the program is able to detect statistically significant similarity between homologous proteins that are not recognizably related using traditional single-pass database search methods.

For each instance of the input pattern in a database sequence, paired with an instance in the query, PHI-BLAST attempts to find the optimal local alignment containing the aligned patterns. This can be done rigorously by applying dynamic programming to the parts of the two sequences proceeding and the parts following the pattern. The alignment returned is required to begin at the corner of the path graph, but is permitted to end anywhere within the graph. The difficulty with this approach is that, to guarantee optimality, a very large portion of the path graph needs to be searched, and this requires inordinate time in a database search. Basically, path graph cells are considered only if the score of the best alignment leading into them falls no more than X below the best score yet found. For sufficiently large values of the X parameter, this approach almost always returns the optimal local alignment. Because PHI-BLAST performs a gapped extension whenever an instance of the input pattern is encountered in the database, reasonable execution times depend upon such instances being relatively rare. Therefore, the only allowed patterns are those that are expected to occur less frequently than once per 5000 database residues. Any pattern that contains four completely specified residues, or three specified residues whose average background frequency is 5.8 percent, passes this test. Of course, the more specific the input pattern, the faster PHI-BLAST will run. The frequency with which a pattern will occur within the database can be estimated easily from background amino acid frequencies.

2.2.4 WU-BLAST

WU-BLAST [17], developed by Warren Gish at Washington University, is a powerful software package for gene and protein identification, using sensitive, selective and rapid similarity searches of protein and nucleotide sequence databases. WU-BLAST 2.0 and NCBI-Gapped BLAST are distinctly different software packages. In spite of a common lineage for some portions of their code, in many important ways the two packages do their work differently and, consequently, obtain different results and offer different features.

The classical ungapped BLAST algorithm has not been changed in WU-BLAST 2.0, thus retaining the sensitivity and control characteristics of this algorithm that users have become accustomed to with previous versions of BLAST. Due to the aforementioned optimization, when assessed at the same sensitivity level, the classical BLAST algorithm in WU-BLAST 2.0 is essentially the same speed and uses much less memory than the 2-hit algorithm.

When run on a multiprocessor computer system, the NCBI software uses one CPU or thread by default, whereas WU-BLAST will attempt to use all available CPUs by default. As the use of multiple CPUs is never as efficient as using a single CPU, speed comparisons should be made between the two programs when using the same number of CPUs.

NCBI TBLASTX does not perform gapped alignments, whereas WU-TBLASTX produces gapped alignments by default with the option to turn them off.

When searching very large databases, virtual memory requirements are dramatically reduced in WU-BLAST 2.0, eliminating program failures that occurred when system resource limits were unexpectedly reached. Virtual databases are supported in licensed BLAST 2.0. Virtual databases can be specified on the command line as a white space-delimited list of component database names.

2.2.5 Parallel Implementations

2.2.5.1 Parallel Hardware

There are two basic methods of mapping sequence comparison to a parallel processor, one with coarse-grain parallelism, and the other with fine-grain parallelism [18].

The coarse-grain approach is appropriate for large database searches. In this case, the sequences are partitioned among the processing elements (PEs) so as to have similarly sized subsets of the database assigned to each PE. Then, multiple independent sequence analyses are performed. Each processing element must have sufficient memory for its sequences (or significant segments of its sequences), as well as for two rows of the dynamic programming matrix: the previous row, and the row being currently formed. If only a small number of sequence comparisons are required, coarse-grain parallelism will offer little speedup because the database sequences will not be spread across all processing elements.

2.2.5.2 Smith-Waterman Algorithm with MMX* Technology

Since 1995, Intel Corporation's general-purpose processor line has supported Single-Instruction, Multiple-Data (SIMD) instructions, termed "MMX* Technology". A significant majority of the installed base of personal computers, and almost every PC sold today, supports it. These instructions can be used to take advantage of the parallelism of Smith-Waterman and get competitive performance from general-purpose, inexpensive processors.

A form of parallel processing capability termed the single instruction, multiple data (SIMD) technology [19] enables microprocessors to perform the same operation (logical, arithmetic or other) in parallel on several independent data sources. It is possible to exploit this by dividing wide registers into smaller units in the form of microparallelism or SIMD within a register. However, modern microprocessors have added special registers and instructions to make the SIMD technology easier to use. The technology is included in some of the most widely used modern microprocessors.

including the Intel Pentium MMX, II and III.

The ParAlign algorithm [20] has been specifically designed to take advantage of this technology. It is a faster and more sensitive algorithm for sequence similarity searching in view of the rapidly increasing amounts of genomic sequence data available. The algorithm initially exploits parallelism to perform a very rapid computation of the exact optimal ungapped alignment score for all diagonals in the alignment matrix. Then, a novel heuristic is employed to compute an approximate score of a gapped alignment by combining the scores of several diagonals. This approximate score is used to select the most interesting database sequences for a subsequent Smith-Waterman alignment, which is also parallelised. The resulting method represents a substantial improvement compared to existing heuristics. The sensitivity and specificity of ParAlign was found to be as good as Smith-Waterman implementations when the same method for computing the statistical significance of the matches was used. In terms of speed, only the significantly less sensitive NCBI BLAST v.2 program was found to outperform the new approach. The algorithm of ParAlign has the following improvements:

1) Efficient parallel computation of the optimal ungapped alignment score for all diagonals. In the first phase BLAST scans for short regions of 1-3 consecutive amino acids that are identical or very similar between the two sequences. By default, BLAST requires 3 amino acids to be very similar, i.e., the sum of the three score matrix elements corresponding to the three pairs of amino acids must exceed a certain limit. NCBI BLAST v.2 additionally requires that there are two such groups on the same diagonal within a distance of 40 amino acids. Hence, if the similarity is more distributed along one diagonal it may not be detected by the present heuristic methods. This weakness is overcome in ParAlign by calculating the exact ungapped alignment score for each diagonal.

2) Computation of an approximate gapped alignment score. In the second phase of sequence alignment, sequence similarities may pass undetected if the similarity is

evenly distributed on several diagonals, in which case the optimal alignment contains many gaps. Many heuristic algorithms select a small fraction of the database sequences for a more rigorous examination using an optimal alignment algorithm within a band (FASTA) or region (NCBI BLAST v.2) surrounding the most interesting initially identified regions. BLAST uses only the score of the highest scoring initial region to determine whether a more accurate gapped alignment should be constructed. it is hence unable to recognize an otherwise significant alignment if none of the high scoring segment pairs found has a score above about 40 with a typical length query. ParAlign employs a new heuristic for computing an estimated gapped alignment score, which is used to select the most interesting fraction of database sequences for further examination. When the ungapped alignment scores for all diagonals have been found, the scores for each diagonal are combined in a new inter-diagonal scoring function, which is also a kind of maximum partial sum of scores function. Using this function, high scoring regions on neighbouring diagonals will receive a high score. The chosen inter-diagonal scoring function has been found to be very effective in filtering the sequence database.

2.2.6 Distributed Implementation — TurboBLAST

TurboBLAST [21] is a Java-based application released by TurboGenetic in April, 2001. It can run on Macs, PCs and Linux boxes, helping researchers to maximize the performance potential of their computer networks regardless of platform. It is also scalable - if researchers want more speed, all they have to do is add more machines. It provides a high-performance BLAST service based on the use of multiple executions of the unmodified NCBI BLASTALL program. BLASTALL is one of the implementations of BLAST available from the NCBI that can be used to perform all five variations of the BLAST algorithm, BLASTN, BLASTP, BLASTX, tBLASTN, and tBLASTX. It accelerates the BLAST throughput and has greater scientific insights and tremendous cost-savings.

TurboBLAST achieves high performance for large numbers of BLAST searches in

two ways: by use of batch queuing, and by splitting individual BLAST search requests into multiple parts by dividing up both the set of input (query) sequences and the database(s) against which the search is to be conducted. In addition, TurboBLAST manages some of the chores related to updating genomic databases in a multi-machine environment.

The TurboBLAST system consists of three main components arranged in a classic 3-tier system containing an end-user client, a BLAST Master, and a number of BLAST Workers.

Client is the part of the system accessed by the bench scientist performing BLAST queries. The Client generates BLAST requests and submits them to the BLAST Master, waits for the results, and, after receiving them, stores them appropriately.

BLAST Master accepts a BLAST request from a Client and converts it into one or more requests to the Piranha Backend System, a sophisticated parallel execution environment that manages all BLAST requests and BLAST Workers. It waits for the results, combines them into a unified result that exactly matches ordinary BLAST output, and delivers the unified result to the Client.

BLAST Workers are components that execute BLASTALL against all or some portion of a genomic database, taking as input one or more of the query sequences.

TurboBLAST accelerates BLAST by automatically splitting BLAST requests into multiple small tasks, executing the tasks in parallel, and then merging the task results into a single unified result. Each BLAST request contains a number of input query sequences to be compared against one or more nucleotide or protein databases. The result is an ordered list of the sequences in the database(s) that have the greatest homology to one of the query sequences.

In order to deliver high performance overall, TurboBLAST provides the following features: 1) Fault Tolerance: any of the BLAST Workers may fail without affecting the correctness of the system behavior or of the BLAST results. Of course, the overall computation will take a little longer because there are fewer BLAST Workers available. 2) Dynamic Load Balancing: each of the BLAST Workers manages its own

workload. In general, if there are an ample number of tasks, the workload will be shared fairly, and very effective load balancing will be achieved. This is true even when the BLAST Workers are heterogeneous, e.g., contain different processor types, use different operating systems, run at different speeds, or contain different amounts of RAM. 3) Worker Specialization: once a BLAST Worker has executed a particular task, it will attempt to remain "focused" on tasks that reference the same database or database partition, in order to avoid the costs associated with reloading its memory. Since each BLAST Worker manages its own workload, this is handled in a completely decentralized way, with individual BLAST Workers deciding for themselves when they should look for a different type of task.

2.3 The Public Database for BLAST

International Nucleotide Sequence Database Collaboration includes GenBank (NCBI) at Bethesda in MD of USA, EMBL Nucleotide Sequence Database (EBI) in Hinxton of UK and DNA Data Bank of Japan (DDBJ) in Mishima of Japan. Data is exchanged amongst the collaborating database on a daily basis.

2.3.1 GenBank

GenBank [22] is a public database of all known nucleotide and protein sequences with supporting bibliographic and biological annotation, built and distributed by the National Center for Biotechnology Information (NCBI), a division of the National Library of Medicine (NLM), located on the campus of the US National Institutes of Health (NIH).

2.3.1.1 Organization of the Database and Record Format

Each GenBank entry includes a concise description of the sequence, the scientific name and taxonomy of the source organism, bibliographic references, and a table of features that identifies coding regions and other sites of biological significance, such

as transcription units, repeat regions, sites of mutations or modifications and other sequence features. Protein translations for coding regions are also in the feature table.

The files in the GenBank distribution have traditionally been divided into 'divisions' that roughly correspond to taxonomic divisions, e.g., bacteria, viruses, primates and rodents. In recent years divisions have been added as needed for specific initiatives in biology, such as divisions for EST sequences, genome survey sequences and high throughput genomic sequences. There are currently 16 divisions. For convenience in file transfer, the larger divisions, e.g., EST and primate, are divided into multiple files when posting the bimonthly GenBank releases on NCBI's FTP site.

Expressed sequence tag (EST) data are the major source of new sequence records and genes. ESTs also continue to provide the major source of new gene discoveries. As part of its daily processing of EST data, NCBI identifies through BLAST searches all homologies for new EST sequences and incorporates that information into the companion dbEST database. In order to organize the EST data in a useful fashion, NCBI maintains the UniGene collection of unique human, mouse and rat genes.

Sequence-tagged site (STS) data division of GenBank currently contains anonymous STSs based on genomic sequence as well gene-based STSs derived from the 3' ends of genes and ESTs. These STS records usually include primer sequences, annotations and PCR reaction conditions. The ultimate purpose for creating high resolution physical maps of the human genome is to create a scaffold for organizing large scale sequencing. Physical maps based on STS landmarks are used to develop so-called 'sequence-ready' clones consisting of overlapping cosmids or BACs. As the HTG sequence data derived from these clones are submitted to GenBank, STSs become crucial reference points for organizing, presenting and searching the data. NCBI uses 'electronic PCR' to compare all human sequences with the contents of the STS division of GenBank; This identifies primer-binding sites on the human sequences that may be amplified in a PCR reaction. This tool permits the assignment of an initial location on the map for sequence data and the association of existing GenBank entries to the new reference sequence. The electronic PCR tool is also being made

publicly available on the web to enable any researcher with a new human sequence to relate that sequence to existing maps and HTG sequence data.

Genome Survey Sequence (GSS) data division of GenBank has been the fastest growing division. GSS records represent 'random' genomic sequences, but are predominantly represented by 'BAC ends' which are single reads from bacterial artificial chromosomes used in a variety of genome sequencing projects.

High throughput genomic (HTG) data division of GenBank are unfinished large-scale genomic records that are in transition to a finished state, after which they will be placed in the appropriate organism division. These records are designated as Phase 0-3 depending on the quality of the data. Phase 0 records consist of survey sequences generated to characterize clones and may or may not progress to Phase 1. Phase 1 records contain unfinished sequence, and may consist of unordered, unoriented contigs with gaps. Phase 2 records contain unfinished sequence as ordered, oriented contigs, with or without gaps. Phase 3 records consist of finished sequence, with no gaps and may have annotations. When a HTG record reaches phase 3 it is moved from the HTG division into the appropriate organismic division of GenBank. It is now clear that a great number of human sequences will remain in the unfinished (HTG) division of GenBank as working draft sequence, while completed sequences will continue to move to the corresponding organismic division (PRI). Together these two divisions should add some 2000 Mb of new genomic sequences from US-sponsored laboratories within the next year.

GenBank record format was shown as Figure 1.

2.3.1.2 Submission of Sequence Data

Virtually all records enter GenBank as direct electronic submissions, with the majority of authors using the BankIt or Sequin programs.

About 40 percent of individual submissions are received through a Web-based data submission tool, BankIt. With BankIt, authors enter sequence information directly into a form, edit as necessary and add biological annotation. Free-form

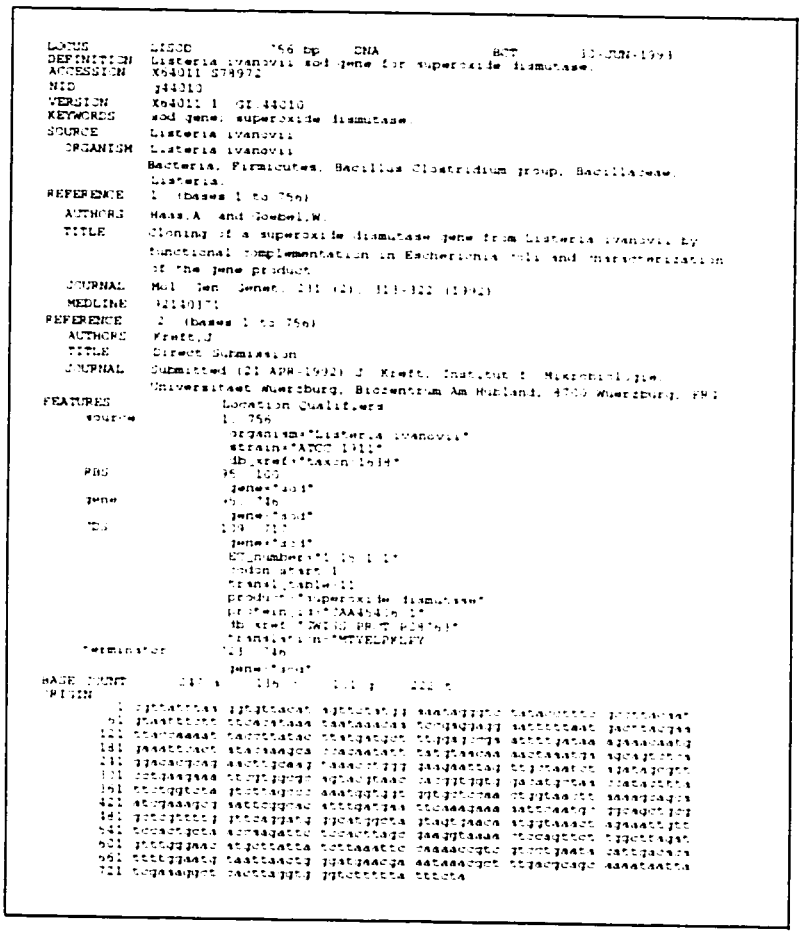


Figure 1: The Record Format of GenBank

text boxes allow the submitter to further describe the sequence, without having to learn formatting rules or use restricted vocabularies. BankIt creates a draft record in GenBank flat-file format for the user to review and revise. BankIt is the tool of choice for simple submissions, especially when only one or a small number of records are submitted. Submitters to update their existing GenBank records can also use bankIt.

NCBI has developed a stand-alone multi-platform submission program called Sequin which can also be linked online to NCBI. Sequin handles simple sequences, as well as long sequences and segmented entries, for which BankIt and other web-based submission tools are not well-suited. Sequin has convenient editing and complex annotation capabilities and contains a number of built-in validation functions for enhanced quality assurance. It is also designed to facilitate the submission of sequences from phylogenetic, population and mutation studies, and can incorporate alignment data. Sequin can be used to edit and update sequence records, as well as to perform sequence analysis. For example, Sequin can now incorporate any analysis tool available on the web that accepts FASTA or ASN.1 formatted data as its input. In addition, Sequin is able to work on large records and read in all of its annotations via simple tables. Versions for Macintosh, PC and Unix computers are available via anonymous FTP to 'ncbi.nlm.nih.gov' in the 'sequin' directory. Once a submission is completed, users can Email it to the address: gb-sub@ncbi.nlm.nih.gov.

2.3.1.3 Retrieving GenBank Data

The Entrez system is an integrated database retrieval system that accesses DNA and protein sequence data, genome mapping data, population sets, the NCBI taxonomy, protein structures from the Molecular Modeling Database (MMDB) and MEDLINE references via PubMed. The DNA and protein sequence data are integrated from a variety of sources and therefore include more sequence data than are available within GenBank alone. Entrez searching is provided on NCBI's web site, via the Query Email server, and as a network client that can be downloaded by FTP.

The most frequent type of analysis performed using GenBank is the search for sequences similar to a query sequence. NCBI offers the BLAST family of programs to locate good alignments between a query sequence and database sequences. BLAST searching is provided on NCBI's web site, via an Email server, and as a set of stand-alone programs distributed by FTP.

NCBI uses the ASN.1 data format for internal maintenance of GenBank, but distributes the GenBank releases in the traditional flat-file format. The full GenBank release and the daily updates are available by anonymous FTP from 'ncbi.nlm.nih.gov'. The full release in flat-file format is available as compressed files in the directory, 'genbank'. A cumulative update file is contained in the sub-directory, 'daily', and a non-cumulative set of updates is contained in 'daily-nc'. A set of sequence-only files in FASTA format, corresponding to the GenBank database subsets searched by BLAST and including the non-redundant nucleotide and protein databases, is available in the 'BLAST/db' directory. Software developers creating their own interfaces or analysis tools for GenBank data are offered the NCBI ToolKit to assist in developing specialized applications.

2.3.2 EMBL

Nucleotide sequence database in European Molecular Biology Laboratory (EMBL) [23] is a comprehensive collection of primary nucleotide sequences maintained at the European Bioinformatics Institute (EBI). New data are released daily into the EMBLNEW database and are immediately available. The EMBL and EMBLNEW databases are stored and maintained in an Oracle data management system and can be searched on the internet with the Sequence Retrieval System (SRS), the EBI search engine for molecular biology databanks. Since interconnectivity between biomolecular databases has become essential for utilizing the wealth of information becoming available, EMBL database entries are cross-referenced to other databases like the eukaryotic promoter database TRANSFAC, FlyBase, TrEMBL and SWISS-PROT. SWISS-PROT itself is linked to more than 30 different databases thus providing a focal point for database

interconnectivity.

2.3.2.1 Organization of the Database and Record Format

A complete overview of the dataflow to and from the EMBL Database is provided in Figure 2.

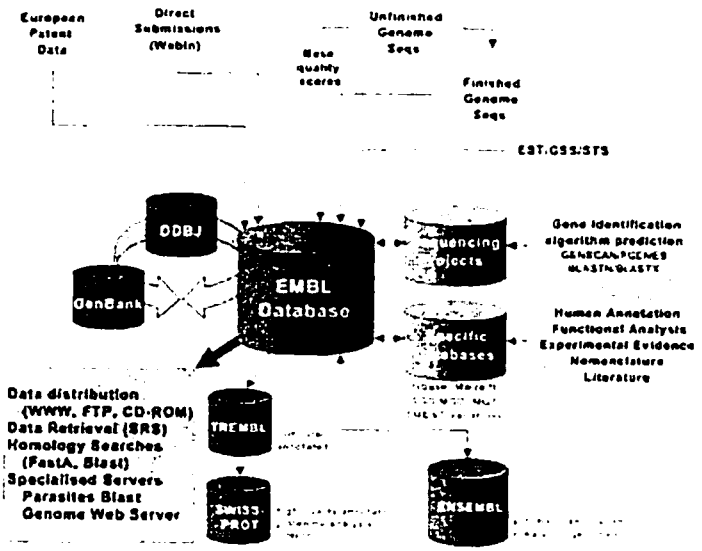


Figure 2: An Overview of Dataflow for EMBL Database

Divisions provide subsets of the database, which reflect the areas of interest of users. The EMBL Database currently consists of 18 divisions with each entry belonging in exactly one division. In each entry the division is indicated using the three letter codes, e.g., PRO = Prokaryotes, HUM = Human, PHG = Bacteriophages, PLN = Plants etc. The grouping is mainly based on taxonomy with a few exceptions like the HTG, EST, STS, and GSS divisions. For these divisions, grouping is based on the specific nature of the underlying data.

Expressed sequence tags (ESTs). The EST division files contain sequence and mapping data on 'single-pass' cDNA sequences or ESTs from a number of organisms.

In addition to the EST division files in the EMBL database release, the EBI provides ESTLIB, which includes further information about the libraries from which the EST sequences were derived.

High-throughput genome sequences (HTGs). In order to make genome sequences produced by high-throughput sequencing projects available to the user community as soon as possible, the HTG division includes 'unfinished' genome project data with annotation for many of these records being generated through computer analyses. Entries in this division all contain keywords to indicate the status of the sequencing. A single accession number is assigned to one clone, and as sequencing progresses and the entry passes from one phase to another, it will retain the same accession number with only the most recent version of a HTG record remaining in EMBL. Once 'finished', HTG sequences are moved into the relevant primary EMBL taxonomic division.

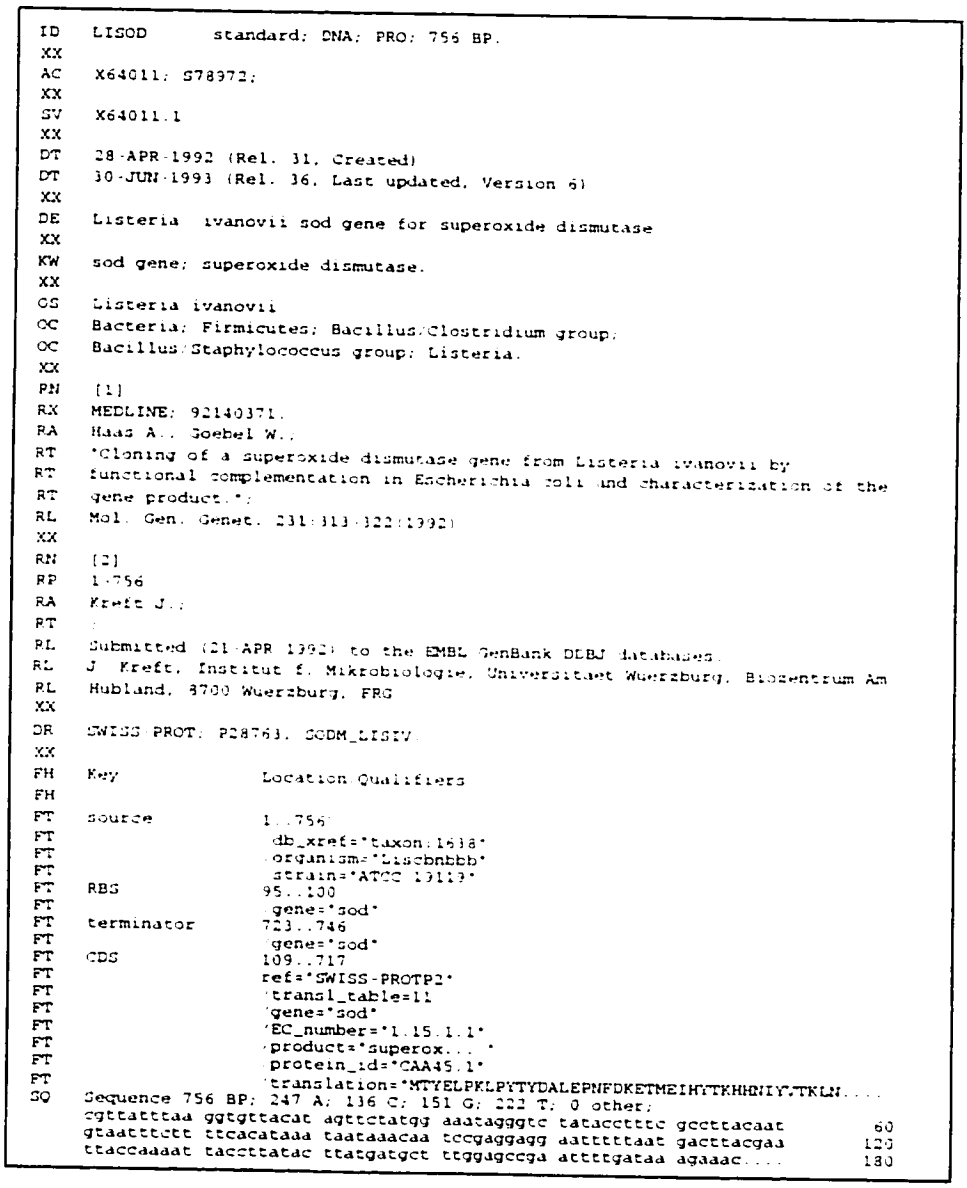
Patent sequence data. The EMBL database continues to collaborate with the European Patent Office to capture patent sequences from patent applications and integrate US and Japanese patent sequence data provided by the DDBJ and GenBank collaborators. Patent data can be retrieved from the EMBLNEW and EMBL databases and are also available via FTP.

EMBL record format is shown as Figure 3.

2.3.2.2 Submission of Sequence Data

Two major sources contribute to the EMBL Database: individual scientists, who submit data directly to the collaborating databases, and genome project groups which produce very large volumes of nucleotide sequence data over an extended period of time, including bulk submissions of ESTs, STSs, GSSs or large genomic records (high-throughput and finished data). Researchers submitting new sequences directly to the EMBL database use either the Internet (WEBIN) or a stand-alone software tool (SEQUIN).

The EBI's submission tools incorporate facilities for providing and checking biological information.



```
ID   LISOD      standard; DNA; PRO; 756 BP.
XX
AC   X64011; S78972;
XX
SV   X64011.1
XX
DT   28-APR-1992 (Rel. 31, Created)
DT   30-JUN-1993 (Rel. 36, Last updated, Version 6)
XX
DE   Listeria  ivanovii sod gene for superoxide dismutase
XX
KW   sod gene; superoxide dismutase.
XX
OS   Listeria ivanovii
OC   Bacteria; Firmicutes; Bacillus/Clostridium group;
OC   Bacillus/Staphylococcus group; Listeria.
XX
RN   [1]
RX   MEDLINE; 92140371.
RA   Haas A., Goebel W.;
RT   "Cloning of a superoxide dismutase gene from Listeria ivanovii by
RT   functional complementation in Escherichia coli and characterization of the
RT   gene product.";
RL   Mol. Gen. Genet. 231:313-322(1992)
XX
RN   [2]
RP   1-756
RA   Kreft J.;
RT   ;
RL   Submitted (21-APR-1992) to the EMBL GenBank DDBJ databases.
RL   J. Kreft, Institut f. Mikrobiologie, Universitaet Wuerzburg, Biozentrum Am
RL   Hubland, 8700 Wuerzburg, FRG
XX
DR   SWISS-PROT; P28763, SODM_LISIV.
XX
FH   Key             Location/Qualifiers
FH
FT   source          1..756
FT                   /db_xref="taxon:1638"
FT                   /organism="Listeria ivanovii"
FT                   /strain="ATCC 19119"
FT   RBS             95..100
FT                   /gene="sod"
FT   terminator      723..746
FT                   /gene="sod"
FT   CDS             109..717
FT                   /ref="SWISS-PROT:P28763"
FT                   /transl_table=11
FT                   /gene="sod"
FT                   /EC_number="1.15.1.1"
FT                   /product="superox...."
FT                   /protein_id="CAA45.1"
FT                   /translation="MTYELPKLPYTYDALEPNFDKETMEIHYTKHHNIYVTKLN....
SQ   Sequence 756 BP; 247 A; 136 C; 151 G; 222 T; 0 other;
     cgttatttaa ggtgttacat agttctatgg aaatagggtc tatacctttc gccttacaat        60
     gtaatttctt ttcacatdaa taataaacaa tccgaggagg aatttttaat gacttacgaa       120
     ttaccaaaat taccttatac ttatgatgct ttggagccga attttgataa agaaac....       180
```

Figure 3: The Record Format of EMBL

VECTOR SCANNING, a WWW-based interactive vector scanning service is available for submitters to assist in the screening of sequences for vector contamination before submission. The vector screening service uses the latest implementation of the BLAST algorithm and the special sequence databank EMVEC, comprised of an extraction of sequences from the synthetic division of EMBL commonly used in cloning and sequencing experiments. EMVEC is updated with each release of EMBL and is available from the EBI's FTP server.

WEBIN is an Internet based tool for submission of nucleotide sequences to the EMBL database. WEBIN is designed to allow fast submission of either single, multiple or even very large numbers of sequences. Sequence annotation in WEBIN is added from the 'Summary and Sequence Features' page. Any number of relevant features can be easily added to the sequence feature table from the comprehensive list and by filling out the specific feature forms. To assist submitters in selecting features for their sequence, WebFeat provides a full description of all EMBL features and qualifiers while the EMBL annotation examples illustrate how these features and qualifiers should be used within standard EMBL entries.

2.3.2.3 Retrieving EMBL Data

The Sequence Retrieval System (SRS) server at the EBI integrates and links a comprehensive collection of specialized databanks along with the main nucleotide and protein databases. The SRS system allows the databases to be searched using, for example, sequence, annotations, keywords, and author names. Complex, cross-database queries can also be executed and users should refer to the detailed instructions which are available online.

The EBI provides a comprehensive set of sequence database searching algorithms that can be accessed both interactively from the EMBL WWW site or by Email. EMBL may be searched as a whole or by individual taxonomic division. The most commonly used algorithms available are FASTA3, WU-BLAST, and NCBI-BLAST.

Specialized sequence analysis programs are available from the EBI. Such services

include multiple sequence alignment and inference of phylogenies using CLUSTALW, Gene prediction using GeneMark, pattern searching and discovery using PRATT, as well as applications which have been developed in-house for various projects.

EMBnet, the European Molecular Biology Network was initiated in 1988 to link European laboratories using biocomputing and bioinformatics in molecular biology research as well as to increase the availability and usefulness of the molecular biology databases within Europe. Remote copies of the nucleotide and protein sequence databases, updated daily, as well as other molecular biology resources are held at nationally mandated nodes. As bioinformatics grows, EMBnet plays an important role in support, training, research and development for the European bioinformatics research community.

2.3.3 DDBJ

The DNA Data Bank of Japan (DDBJ) [24] began its activities in earnest in 1986 in collaboration with EMBL and GenBank. Before beginning, there was a series of discussions among Japanese molecular biologists and biophysicists about the organization in which the data bank should be established. The discussion finally resulted in a proposal for the data bank to be founded in the National Institute of Genetics (NIG), which is governed by the Ministry of Education, Sports, Science and Culture (MESSC). The proposal was soon implemented with the endorsement of MESSC. Since then, continuous support by MESSC has made it possible to maintain DDBJ activities, and establish a new center in 1995 at NIG. It is called the Center for Information Biology (CIB).

2.3.3.1 Organization of the Database and Record Format

A large-scale data submission system MSS (Mass Submission System) has been developed and improved in DDBJ. MSS includes an off-line tool, MST (Mass Submission Tool), which functions by arranging data into a form ready for submission and also acts as a parser at a data producing site. Figure 4 is for overview and modules of the

large-scale data submission system.

Yamato II is a data management system at DDBJ. It is an objected-oriented database management system written in C++. DDBJ record format is shown as Figure 5.

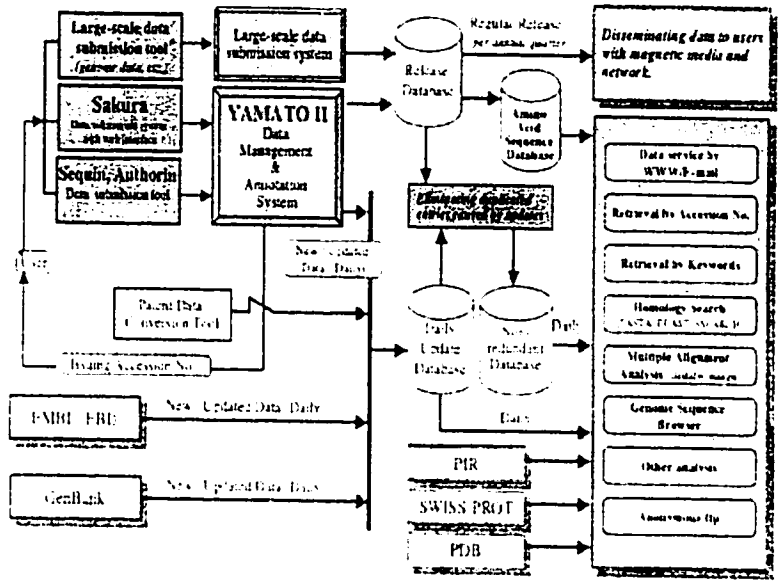
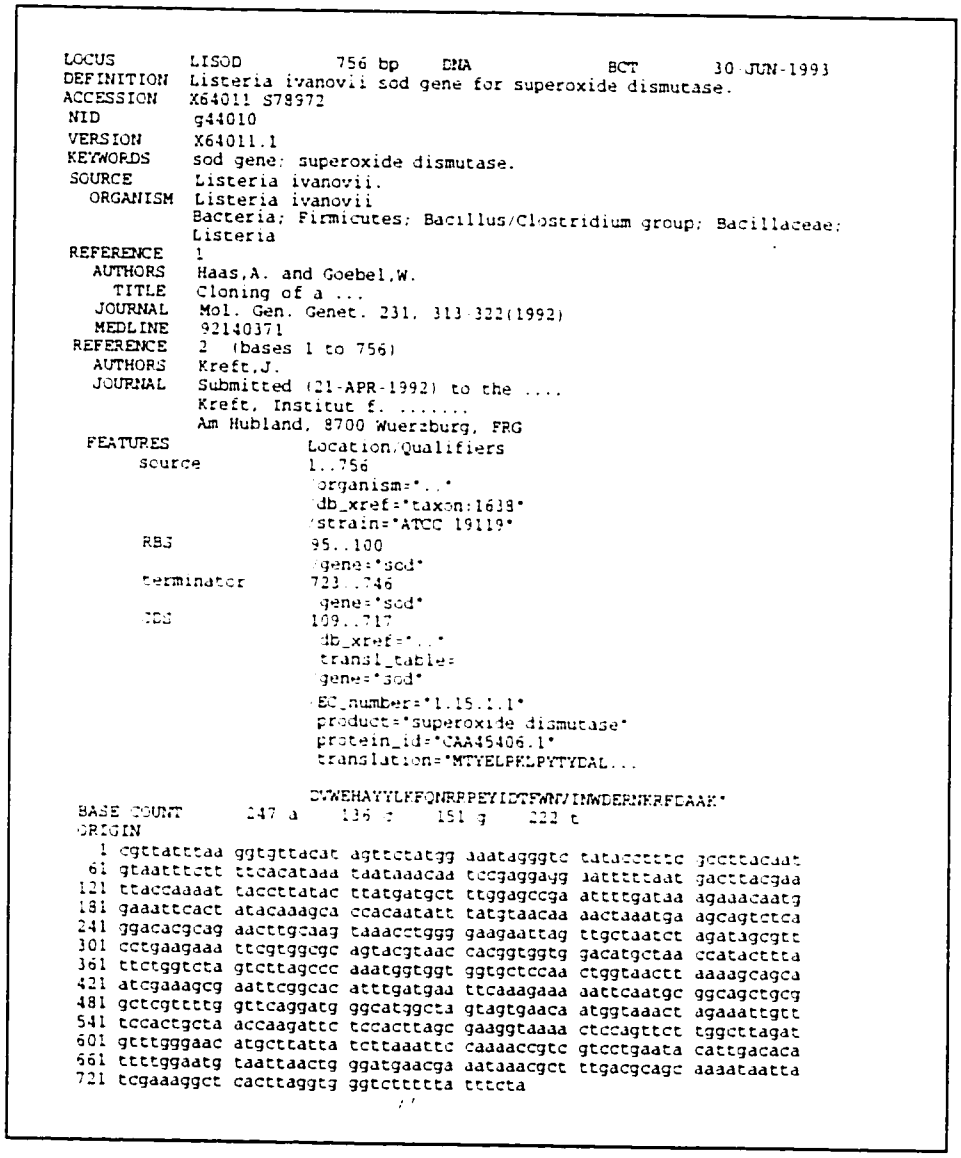


Figure 4: An Overview of DDBJ Database

2.3.3.2 Submission of Sequence Data

DDBJ has introduced a system which has the capacity to accept and annotate this influx of data. When submitting ordinary data, a submitter has several options. Firstly, Sakura, which requires only a WWW browser, like Netscape or MS Explorer. Secondly, Authorin, which was originally developed by GenBank. The tool was specifically aimed at a researcher who directly submitted nucleotide sequence data including citation, source organism, natural host and laboratory host information. Thirdly, Sequin, which was also developed by GenBank, can accept short mRNA sequence, multiple annotations, segment sets of DNA in addition to the previously described features of Authorin.



```
LOCUS       LISOD         756 bp    DNA             BCT       30-JUN-1993
DEFINITION  Listeria ivanovii sod gene for superoxide dismutase.
ACCESSION   X64011 S78972
NID         g44010
VERSION     X64011.1
KEYWORDS    sod gene; superoxide dismutase.
SOURCE      Listeria ivanovii.
  ORGANISM  Listeria ivanovii
            Bacteria; Firmicutes; Bacillus/Clostridium group; Bacillaceae;
            Listeria
REFERENCE   1
  AUTHORS   Haas,A. and Goebel,W.
    TITLE   Cloning of a ...
  JOURNAL   Mol. Gen. Genet. 231, 313-322(1992)
  MEDLINE   92140371
REFERENCE   2  (bases 1 to 756)
  AUTHORS   Kreft,J.
  JOURNAL   Submitted (21-APR-1992) to the ....
            Kreft, Institut f. .......
            Am Hubland, 8700 Wuerzburg, FRG
  FEATURES             Location/Qualifiers
     source            1..756
                       /organism="..."
                       /db_xref="taxon:1638"
                       /strain="ATCC 19119"
     RBS               95..100
                       /gene="sod"
     terminator        723..746
                       /gene="sod"
     CDS               109..717
                       /db_xref="..."
                       /transl_table=
                       /gene="sod"
                       /EC_number="1.15.1.1"
                       /product="superoxide dismutase"
                       /protein_id="CAA45406.1"
                       /translation="MTYELPKLPYTYDAL...

                       DVWEHAYYLKFQNRRPEYIDTFWNVINWDERNKRFDAAK"
BASE COUNT      247 a    136 c    151 g    222 t
ORIGIN
        1 cgttatttaa ggtgttacat agttctatgg aaataqqqtc tatacctttc gccttacaat
       61 gtaatttctt ttcacataaa taataaacaa tccgaggagg aatttttaat gacttacgaa
      121 ttaccaaaat taccttatac ttatgatgct ttggagccga attttgataa agaaacaatg
      181 gaaattcact atacaaagca ccacaatatt tatgtaacaa aactaaatga agcagtctca
      241 ggacacgcag aacttgcaag taaacctggg gaagaattag ttgctaatct agatagcgtt
      301 cctgaagaaa ttcgtggcgc agtacgtaac cacggtggtg gacatgctaa ccatacttta
      361 ttctggtcta gtcttagccc aaatggtggt ggtgctccaa ctggtaactt aaaagcagca
      421 atcgaaagcg aatttcggcac atttgatgaa ttcaaagaaa aattcaatgc ggcagctgcg
      481 gctcgttttg gttcaggatg ggcatggcta gtagtgaaca atggtaaact agaaattgtt
      541 tccactgcta accaagattc tccacttagc gaaggtaaaa ctccagttct tggcttagat
      601 gtttgggaac atgcttatta tcttaaattc caaaaccgtc gtcctgaata cattgacaca
      661 ttttggaatg taattaactg ggatgaacga aataaacgct ttgacgcagc aaaataatta
      721 tcgaaaggct cacttaggtg ggtcttttta tttcta
                                        //
```

Figure 5: The Record Format of DDBJ

The DDBJ's system for managing large-scale data submissions primarily consists of four separate parts which are (i) the WWW data submission system, (ii) large-scale data submission system/off-line (installed at the submitter side), (iii) data submission management system, and (iv) data storing system .

The WWW data submission system incorporates the internet environment so that a submitter is able to interactively send an inquiry to DDBJ, and to receive an ID, a password and a template ID from DDBJ. The system is, therefore, designed to properly handle all pieces of information necessary for the data submission. An interface of the system is available in both Japanese and English depending on the submitter's preference.

The large-scale data submission system/off-line is publicly available by downloading the program from an FTP server at DDBJ. The system rigorously checks the file and annotations automatically excluding any invalid characters. It also allows a submitter to verify the file formats and consistency concerning annotation and sequence prior to actually sending the data files by Email. Two types of files are used for the submission: one is used for annotation and the other is employed for recording sequences. The file for annotation is in a tabular format which popular word processors, spreadsheet and database management systems can handle. Therefore, a submitter is not required to purchase an expensive platform in addition to their conventional system. Nucleotide sequences are submitted in the FASTA file format. The large-scale data submission system/off-line has another important function, which is to automatically verify the file format and consistency. This is important for submitting large-scale genome data, because it greatly alleviates the number of efforts, which exhaust the human resources of reviewers and annotators at DDBJ.

Data submission management system manages the submitted files and monitors the submission progress. After receiving the Excel and FASTA files sent from a submitter, a record is made for each submission by operating the system, and the message is sent to other staff members of DDBJ by Email.

Data storing system performs more rigorous checking before completely loading

the files to the master database in addition to checking the consistency regarding the annotation and sequence. The template information ranging from the locus, definition, accession number, the feature information and to the source organism is also loaded to the master database by the system. Finally, the system issues an accession number to the administrator of DDBJ who notifies the number to a submitter by Email.

There was a special case for submitting GSS (Genome Survey Sequence) data. Kitasato University submitted the data using a tool called the GSSin which was specially developed by DDBJ. In this case, the files, which are formatted in FASTA, are sent to DDBJ through FTP. After receiving the files by a server named the supernig, they are then automatically checked and registered to the Sybase database by using another tool called the GSSsub, which was also developed by DDBJ on a UNIX server, Generous. During this process GSSsub also automatically monitors the submitted files to ensure that they are valid. In addition, the system automatically registers and releases the updated GSS data submitted from the university every night. In the event of a registration error the system sends an error message to the DDBJ administrator by Email.

DDBJ is now considering upgrading the data submission and processing systems by introducing a new type of architecture such as CORBA, which is an object oriented distributed platform. The new systems will be well suited for more efficiently managing the submitted data by reducing data handling labor and time.

2.3.3.3 Retrieving DDBJ Data

A device is developed by which to reorganize the entire database into a species-oriented database in which the data are divided into a species as a unit. They have developed a tool by which one can first specify a species of interest by its scientific, and then carry out a keyword or homology search against the data for that species alone. This tool is expected to be useful particularly for examining whether a particular gene sequence is available for the species in question. As data accumulate in the database

at an ever-increasing rate, the tool will provide a means of reducing retrieval time.

The same tool as above is applied to ESTs. If one is interested in ESTs of a particular species, one might carry out a homology search against the data of that species only by giving a probe sequence and specifying the scientific name of the species. As the amount of ESTs grows tremendously, this tool will help reduce retrieval time when one is concerned with a particular species. This is better understood when one realizes that is greater than 70 percent of the rapidly increasing data are ESTs .

Another device they developed is a tool that allows one to give multiple probes at once and individually retrieve homologous or similar sequences to those probes. If one uses those two tools together, one would easily examine whether a set of sequences for a particular biological function is available for a species of interest.

For retrieval of the complete genome data, the Genome Information Broker (GIB) has been actively used worldwide web. Since the first implementation of GIB, we have repeatedly revised it and installed new complete genome sequence data into it whenever such data becomes available. By use of GIB one can now search for a particular gene not only for one species but also across the 23 species. In this way, one can study, for example, the genomic organization of the gene and its neighbour for different species.

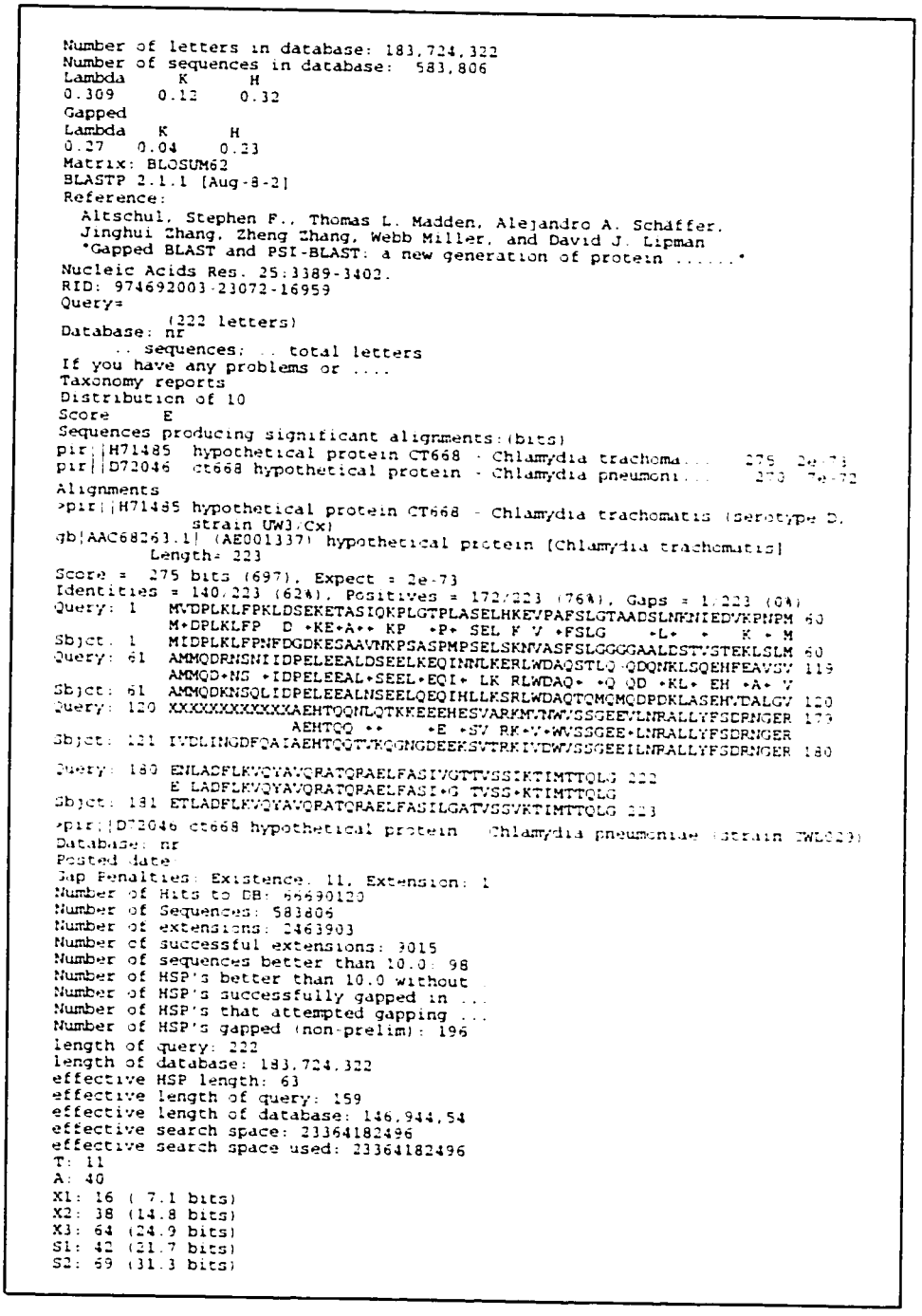
2.4 The Parser of the BLAST Result

A BLAST output file is structures, consisting of a header, a number of sequence alignments and a summary. BLAST Parser can read the result file from a BLAST search and break it down into a tab-delimited file, such as it can be opened with spreadsheet applications such as Microsoft Excel.

A typical alignment block from BLASTP output is as Figure 6.

A parsed file may be like Figure 7.

There are many kinds of parser versions now which implemented by Perl, Python, C or Java. Some names and descriptions are following.



```
Number of letters in database: 183,724,322
Number of sequences in database:  583,806
Lambda     K      H
0.309    0.12    0.32
Gapped
Lambda   K      H
0.27   0.04    0.23
Matrix: BLOSUM62
BLASTP 2.1.1 [Aug-8-2]
Reference:
  Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
  Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman
  "Gapped BLAST and PSI-BLAST: a new generation of protein ......"
Nucleic Acids Res. 25:3389-3402.
RID: 974692003-23072-16959
Query=
         (222 letters)
Database: nr
     ... sequences; ... total letters
If you have any problems or ....
Taxonomy reports
Distribution of 10
Score     E
Sequences producing significant alignments:(bits)
pir||H71485  hypothetical protein CT668 - Chlamydia trachoma...    275  2e-73
pir||D72046  ct668 hypothetical protein - Chlamydia pneumoni...    270  7e-72
Alignments
>pir||H71485 hypothetical protein CT668 - Chlamydia trachomatis (serotype D,
          strain UW3/Cx)
gb|AAC68263.1| (AE001337) hypothetical protein [Chlamydia trachomatis]
         Length= 223
Score =  275 bits (697), Expect = 2e-73
Identities = 140/223 (62%), Positives = 172/223 (76%), Gaps = 1/223 (0%)
Query: 1   MVDPLKLFPKLDSEKETASIQKPLGTPLASELHKEVPAFSLGTAADSLNFNIEDVKPNPM 60
           M+DPLKLFP  D +KE+A++ KP  +P+ SEL F V +FSLG     +L+  +   K + M
Sbjct: 1   MIDPLKLFPNFDGDKESAAVNKPSASPMPSELSKNVASFSLGGGGAALDSTVSTEKLSLM 60
Query: 61  AMMQDRNSNIIDPELEEALDSEELKEQINNLKERLWDAQSTLQ-QDQNKLSQEHFEAVSV 119
           AMMQD+NS +IDPELEEAL+SEEL+EQI+ LK RLWDAQ+  +Q QD +KL+ EH +A+ V
Sbjct: 61  AMMQDKNSQLIDPELEEALNSEELQEQIHLLKSRLWDAQTQMQMQDPDKLASEHVDALGV 120
Query: 120 XXXXXXXXXXXXAEHTQQNLQTKKEEEEHESVARKMVTWVSSGEEVLNRALLYFSDRNGER 179
                     AEHTQQ ++     +E +SV RK+V+WVSSGEE+LNRALLYFSDRNGER
Sbjct: 121 IVDLINGDFQAIAEHTQQTVKQGNGDEEKSVTRKIVDWVSSGEEILNRALLYFSDRNGER 180

Query: 180 ENLADFLKVQYAVQRATQPAELFASIVGTTVSSIKTIMTTQLG 222
           E LADFLKVQYAVQRATQPAELFASI+G TVSS+KTIMTTQLG
Sbjct: 181 ETLADFLKVQYAVQRATQPAELFASILGATVSSVKTIMTTQLG 223
>pir||D72046 ct668 hypothetical protein - Chlamydia pneumoniae (strain TWL029)
Database: nr
Posted date:
Gap Penalties: Existence: 11, Extension: 1
Number of Hits to DB: 66690120
Number of Sequences: 583806
Number of extensions: 2463903
Number of successful extensions: 9015
Number of sequences better than 10.0: 98
Number of HSP's better than 10.0 without
Number of HSP's successfully gapped in ...
Number of HSP's that attempted gapping ...
Number of HSP's gapped (non-prelim): 196
length of query: 222
length of database: 183,724,322
effective HSP length: 63
effective length of query: 159
effective length of database: 146,944,54
effective search space: 23364182496
effective search space used: 23364182496
T: 11
A: 40
X1: 16 ( 7.1 bits)
X2: 38 (14.8 bits)
X3: 64 (24.9 bits)
S1: 42 (21.7 bits)
S2: 69 (31.3 bits)
```

Figure 6: An Alignment Block from BLAST

Start Subj	End Subj	Query Star	Query End	Score	Bits	Score2
1 223	1 222	275 697	2e-73 223	140	223	62%
1 224	1 222	270 684	7e-72 224	141	225	62%
5 226	1 222	258 653	3e-68 226	132	223	59%
510 647	10 149	39.5 90	0.031 648	35	142	24%

Figure 7: A Parsed File of BLAST

2.4.1 PERL

2.4.1.1 Boulder

The Boulder data interchange format [25] is an easily parsable hierarchical tag/value format suitable for applications that need to pipe the output of one program into the input of another. It was originally developed for use in the human genome project at the Whitehead Institute/MIT Center for Genome Research, but has since found use in many other areas including system administration and web software development. In addition to its use as a data interchange format, Boulder comes complete with a small database based on the Perl DB-File modules. This database allows you to store arbitrarily complex objects, index them, and later retrieve them using a simple query mechanism.

The Boulder::BLAST class parses the output of the Washington University (WU) or National Cenber for Biotechnology Information (NCBI) series of BLAST programs and turns them into Stone records. You may then use the standard Stone access methods to retrieve information about the BLAST run, or add the information to a Boulder stream. The parser works equally well on the contents of a static file, or on information read dynamically from a filehandle or pipe.

Boulder::BLAST::NCBI parses and read NCBI BLAST files. Specialized parser for NCBI format BLAST output. Loaded automatically by Boulder::BLAST

Boulder::BLAST::WU parses and read WU-BLAST files. Specialized parser for WU-BLAST format BLAST output. Loaded automatically by Boulder ::BLAST.

There are following methods in Boulder:

The parse() method as shown in Figure 8 accepts a path to a file or a filehandle, parses its contents, and returns a Boulder Stone object. The file path may be absolute or relative to the current directly. The filehandle may be specified as an IO::File object, a FileHandle object, or a reference to a glob (FILEHANDLE notation). If you call parse() without any arguments, it will try to parse the contents of standard input.

```
$ stone = Boulder :: Blast -> parse($file_path);

$ stone = Boulder :: Blast -> parse($filehandler);
```

Figure 8: Function of Parse() in Boulder

If you wish, you may create the parser first with Boulder::BLAST new(), and then invoke the parser object's parse() method as many times as you wish to, producing a Stone object each time as shown in Figure 9.

```
$stream = Boulder:: Blast -> new;

$stream = Boulder::Blast -> new($file[,@more_files]);

$stream = Boulder::Blast -> new(\*FILEHANDLER);
```

Figure 9: New() Function in Boulder

The tags defined in the parsed BLAST Stone object are shown in following Figure 10, Figure 11, and Figure 12. Each BLAST-hit is represented by the tag, BLAST-hits as Figure 13. There may be zero, one, or many such tags. They will be presented in reverse sorted order of significance, i.e. most significant hit first.

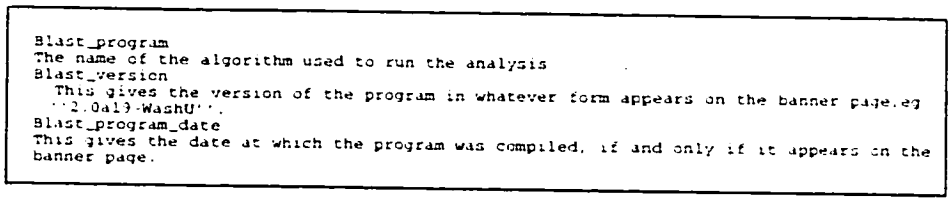


Figure 10: The Tags about the Program in BLAST Stone Object

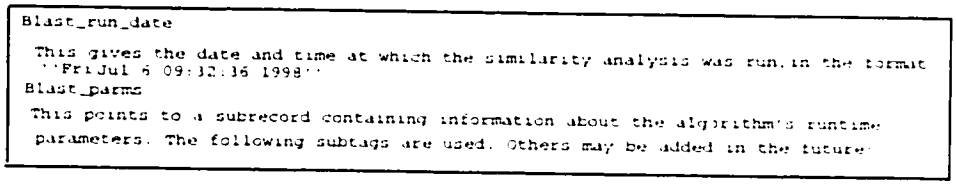


Figure 11: The Tags about the Run in BLAST Stone Object

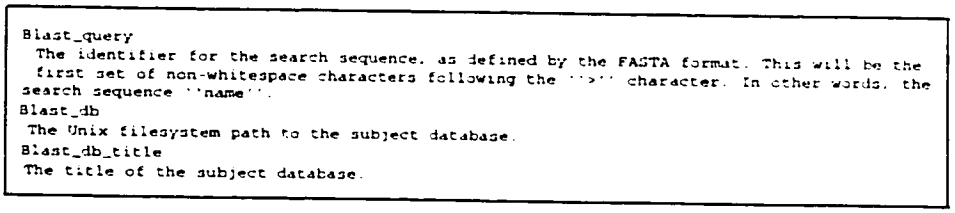


Figure 12: The Tags of the Query Sequence and Subject Database in BLAST Stone Object

```
Name
      The name identifier of the sequence that was hit
Length
      The total length of the sequence that was hit
Signif
      The significance of the hit. If there are multiple HSPs in the hit,
 this will be the most significant (smallest) value.
Identity
      The percent identity of the hit. If there are multiple HSPs, this
 will be the one with the highest percent identity.
Expect
      The expectation value for the hit. If there are multiple HSPs, this
 will be the lowest expectation value in the set.
Hsps
      One or more sub-sub-tags, pointing to a nested record
 containing information about each high-scoring segment pair
 (HSP). See the next section for details.
```

Figure 13: The BLAST-hits Tag in BLAST Stone Object

Each BLAST-hit tag will have at least one, and possibly several HSPs tags, each one corresponding to a high-scoring segment pair. These records contain detailed information about the hit, including the alignments as Figure 14.

Figure 15 show a parsed Stone object. Long lines have again been truncated.

2.4.1.2 Bioperl: BPlite - Lightweight BLAST Parser

BPlite [26] is a package for parsing BLAST reports. The BLAST programs are a family of widely used algorithms for sequence database searches. The reports are non-trivial to parse, and there are differences in the formats of the various flavors of BLAST. BPlite parses BLASTN, BLASTP, BLASTX, TBLASTN, and TBLASTX reports from both the high performance WU-BLAST, and the more generic NCBI-BLAST. BPlite is for those people who would rather not have a giant object specification, but rather a simple handle to a BLAST report that works well in pipes.

BPlite has three kinds of objects, the report, the subject, and the HSP. To create a new report, you pass a filehandle reference to the BPlite constructor as shown in Figure 16 and Figure 17. A subject is a BLAST hit, which should not be confused with an HSP. A BLAST hit may have several alignments associated with it. A useful way of thinking about it is that a subject is a gene and HSPs are the exons. Subjects have one attribute (name) and one method (nextHSP) as Figure 18.

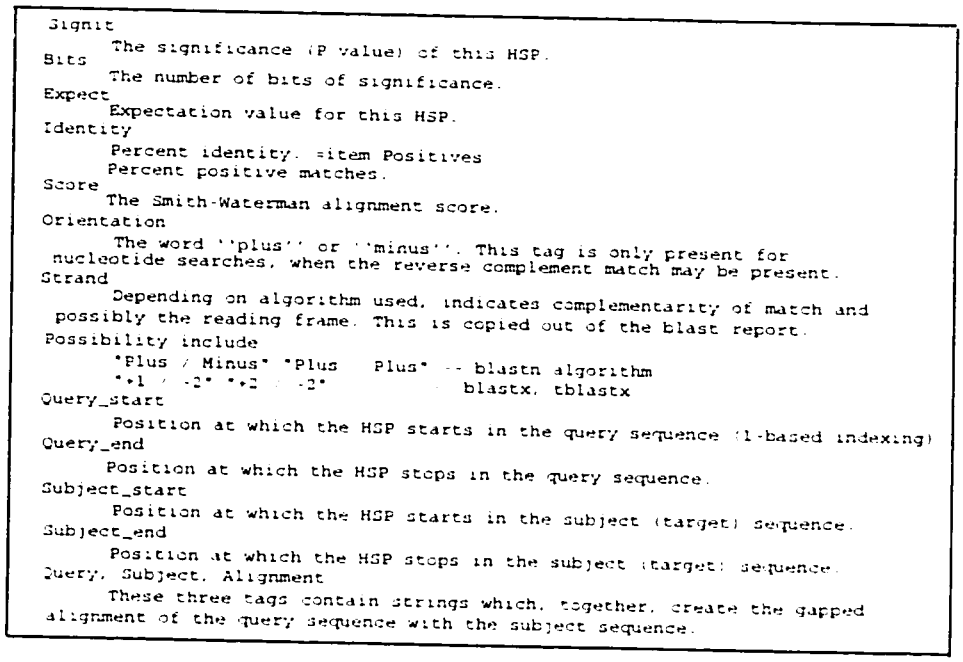


Figure 14: The HSPs Tag in BLAST Stone Object

An HSP is a high scoring pair, or simply an alignment. HSP objects inherit all the useful methods from RangeI/SeqFeatureI/FeaturePair, but provide an additional set of attributes (score, bits, percent, P, match, positive, length, querySeq, sbjctSeq, and homologySeq) that should be familiar to anyone who has seen a BLAST report. For lazy/efficient coders, two-letter abbreviations are available for the attributes with long names (qs, ss, hs) as shown in Figure 19. Ranges of the aligned sequences in query/subject and other information (like seqname) are stored in SeqFeature objects. querySeq, sbjctSeq and homologySeq do only contain the alignment sequences from the BLAST report.

2.4.2 Python

Someone considers that Perl does not scale well past small-to-medium sized projects. Perl is very good for parsing large data files and extracting information, and for writing those parsers quickly and robustly. However, the problems in computational

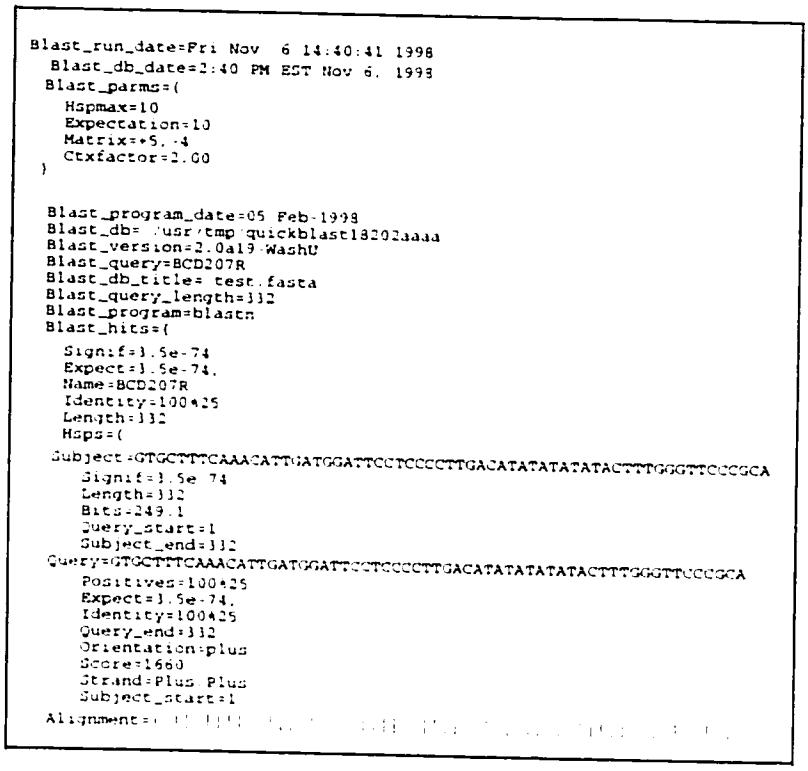


Figure 15: A Parsed Stone Object

```
my $report = new bio::Tools::BPlite(-fh=>\*STDIN);
```

Figure 16: A New Object of Report in BPlite

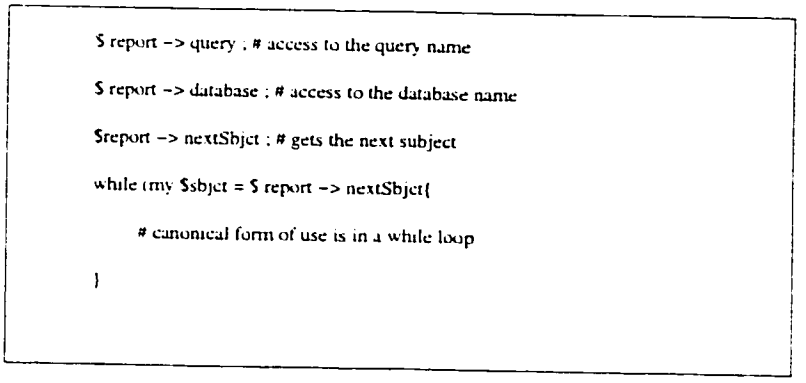


Figure 17: The Attributes and Method of a Report Object

```
$sbjct->name;     # access to the subject name
"$sbjct";         # overloaded to return name
$sbjct->nextHSP; # gets the next HSP from the sbjct
while(my $hsp = $sbjct->nextHSP) {
   # canonical form is again a while loop
}
```

Figure 18: An Object of Subject in BPlite

```
$hsp->score;
$hsp->bits;
$hsp->percent;
$hsp->P;
$hsp->match;
$hsp->positive;
$hsp->length;
$hsp->querySeq;       $hsp->qs;
$hsp->sbjctSeq;       $hsp->ss;
$hsp->homologySeq;    $hsp->hs;
$hsp->query->start;
$hsp->query->end;
$hsp->query->seqname;
$hsp->subject->primary_tag; # "similarity"
$hsp->subject->source_tag;  # "BLAST"
$hsp->subject->start;
$hsp->subject->end;
...
"$hsp"; # overloaded for query->start..query->end bits
```

Figure 19: An HSP Object in BPlite

biology are getting more complicated and the relationships between software packages more complex. Python is much better at expressing those solutions and enabling different components, developed by different people, to work together.

There are several versions of BLAST parser which developed by Jeffrey Chang and et al. The release in March 2001 marks two milestone in the package. First, it has support for sequences and annotations, alignments, substitution matrices, BLAST, ClustalW, ENZYME, FASTA formatting, GenBank, GoBase, Medline, Prosite, Rebase, SCOP, SwissProt, and UniGene. Second, it is the first non-developer's release, which means it is now reasonable to expect people to download it and do work immediately without having to mess around with the code.

Chapter 3

Jini

3.1 How Does Jini Network Technology Work?

In a running Jini system, there are three main players[7] as shown in Figure 20. There is a service, such as a printer, a toaster, a marriage agency, etc. There is a client, which would like to make use of this service. Thirdly, there is a lookup service (service locator), which acts as a broker/trader/locator between services and clients. There is an additional component, and that is a network connecting all three of these, and this network will generally be running TCP/IP. The Jini specification is fairly independent of network protocol, but the only current implementation is on TCP/IP.

Codes will be moved around between these three pieces, and this is done by marshaling the objects. This involves serializing the objects in such a way that they can be moved around the network, stored in this "freeze-dried" form, and later reconstituted by using included information about the class files as well as instance data. This is done using Java's socket support to send and receive objects.

In addition, objects in one JVM (Java Virtual Machine) may need to invoke methods on an object in another JVM. Often this will be done using RMI (Remote Method Invocation), although the Jini specification does not require this and there are many other possibilities.

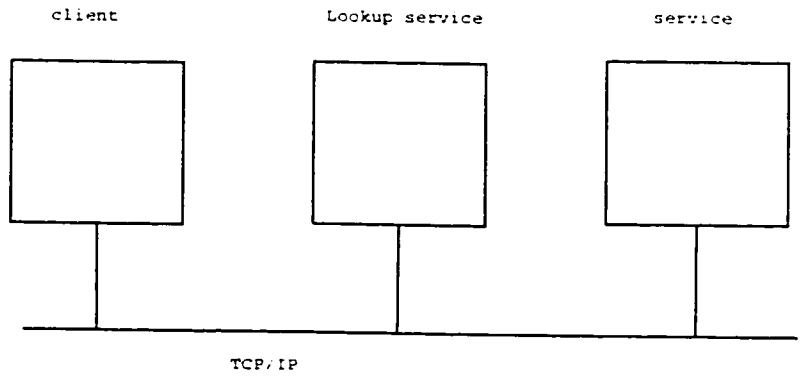


Figure 20: A General View of a Jini System

3.1.1 Service Registration

A service is a logical concept such as a blender, a chat service, a disk. It will turn out to be usually defined by a Java interface, and commonly the service itself will be identified by this interface. Each service can be implemented in many ways, by many different vendors. For example, there may be Joe's dating service, Mary's dating service or many others. What makes them the same service is that they implement the same interface; what distinguishes one from another is that each different implementation uses a different set of objects (or maybe just one object) belonging to different classes.

A service is created by a service provider. A service provider plays a number of roles:

1) It creates the objects that implement the service. It registers one of these - the service object with lookup services. The service object is the "publically visible" part of the service, and will be downloaded to clients.

2) It stays alive in a server role, performing various tasks such as keeping the service "alive".

In order for the service provider to register the service object with a lookup service, the server must first find the lookup service. This can be done in two ways: if the

location of the lookup service is known, then the service provider can use unicast TCP to connect directly to it. If the location is not known, the service provider will make UDP multicast requests, and lookup services may respond to these requests.

Lookup services as shown in Figure 21 will be listening on port 4160 for both the unicast and multicast requests.

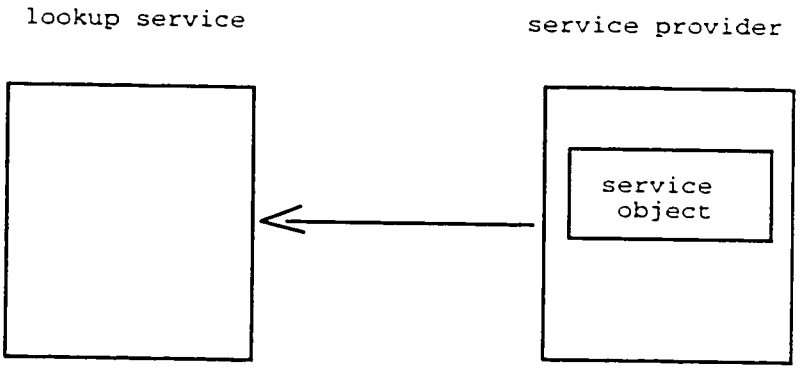


Figure 21: Service Provider Finds a Lookup Service

When the lookup service gets a request on this port, it sends an object back to the server. This object, known as a registrar, acts as a proxy to the lookup service, and runs in the service's JVM (Java Virtual Machine) as shown in Figure 22. Any requests that the service provider needs to make of the lookup service are made through this proxy registrar. Any suitable protocol may be used to do this, but in practice the implementations that you get of the lookup service (e.g from Sun) will probably use RMI. What the service provider does with the registrar is to register the service with the lookup service. This involves taking a copy of the service object, and storing it on the lookup service as shown in Figure 23.

3.1.2 Client Lookup

The client on the other hand, is trying to get a copy of the service into its own JVM. It goes through the same mechanism with the service. First, it finds the lookup service using unicast or multicast as shown in Figure 23.

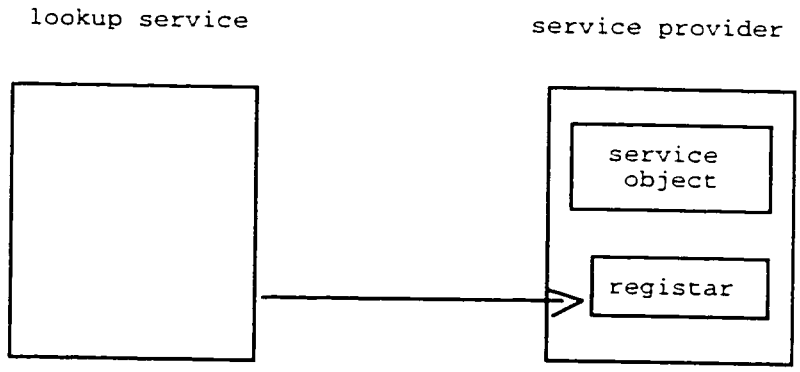


Figure 22: A Registrar is Returned to Service Provider

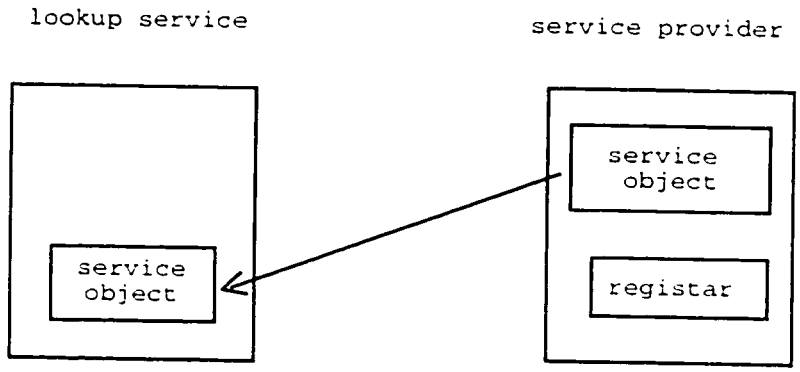


Figure 23: A Registar Registers the Service Object

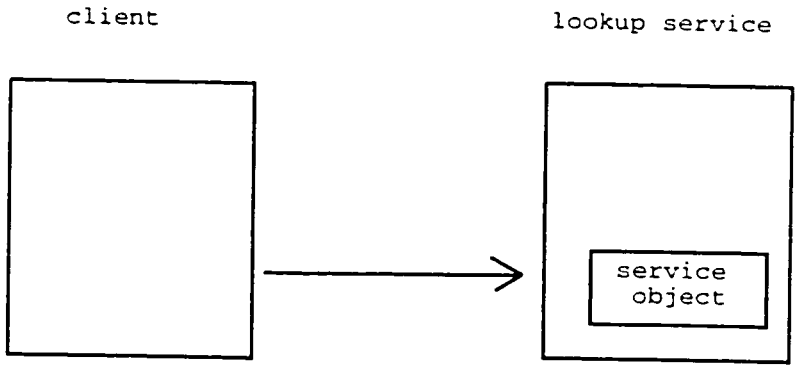


Figure 24: A Client Looking for a Lookup Service

A client gets a registrar from the lookup service as Figure 25. Through the registrar, the client asks for the service object as Figure 26 and a service object is copied to client side as Figure 27.

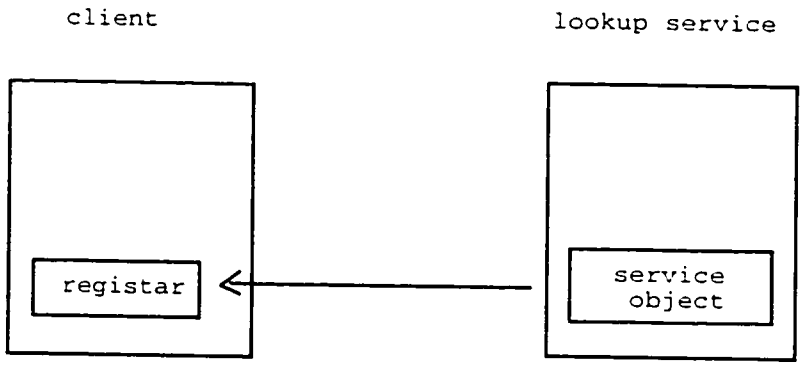


Figure 25: A Registrar is Returned

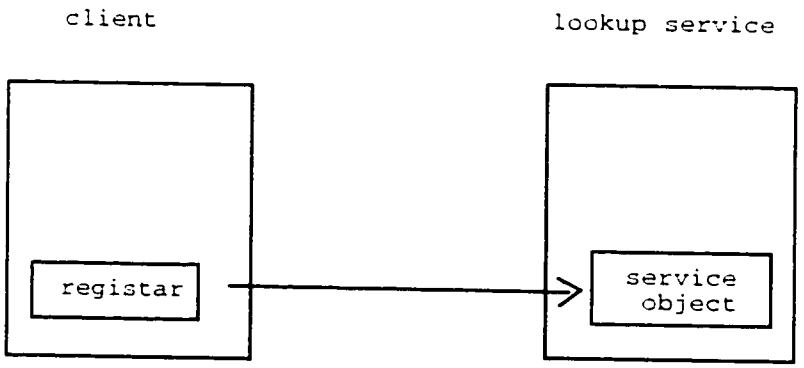


Figure 26: Asking for a Service

3.2 Discovering a Lookup Service

A client locates a service by querying a lookup service. In order to do this, it must first locate such a service. On the other hand, a service must register itself with the lookup service, and in order to do so it must also first locate a service.

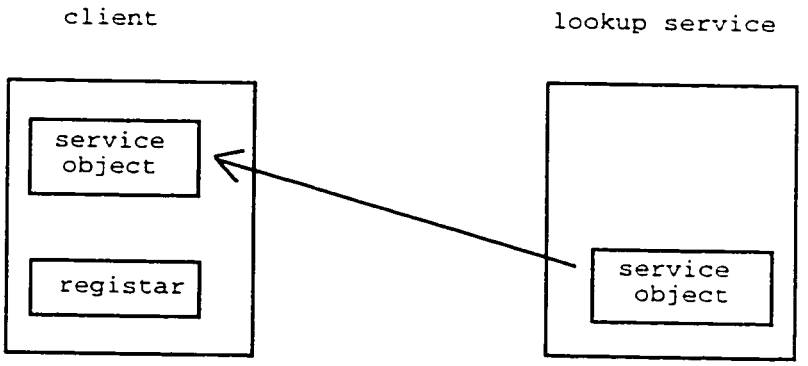


Figure 27: A Service Object is Returned

The initial phase of both a client and a service is thus discovering a lookup service. The search for a lookup service can be done either by unicast or by multicast. In fact, the lookup service is just another Jini service, but it is one that is specialized to store services and pass them on to clients looking for them.

Sun supplies a lookup service called reggie as part of the standard Jini distribution. The reggie requires support services to work: an HTTP server and an RMI daemon, rmid.

A Java object running as a service has a proxy component exported to the service locators and then onto a client. It passes through one JVM in "passive" form and is activated in the client's JVM. An object consists of both code and data, and it cannot be reconstituted from just its data - the class definitions are also required. So class definitions for service proxy objects must also be downloaded, usually from where the service came from. The HTTP server is needed to deliver the class files to clients. This gives two sets of class files: the set needed to run the service and the set needed to reconstitute objects at the client. The mahalo service supplied by Sun as a transaction manager uses the class files in mahalo.jar to run the service, and the class files in mahalo-dl.jar to reconstitute the transaction manager proxy at the client. These files and support services are shown in Figure 28.

The other support service needed for reggie is an RMI daemon(rmid). A proxy

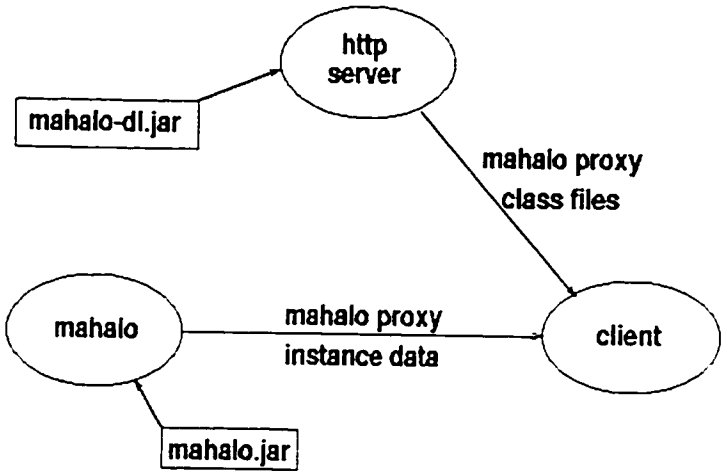


Figure 28: Support Services for Mahalo

service gets exported to the client. In most cases this will need to communicate back to its host service. There are many ways to do this. One mechanism is the Java RMI system. An rmid is a part of the standard Java distribution. This must be run on the same machine as reggie. A rmid is responsible for starting services such as reggie. It will create a new JVM on demand, to run the service. A rmid may look after a number of services, not just reggie, and they will all be run in their own JVMs.

3.2.1 Unicast Discovery

Unicast discovery can be used when you already know the machine on which the lookup service resides, so you can ask for it directly. This is expected to be used for a lookup service that is outside of your local network, which you know the address of anyway

3.2.2 Multicast Discovery

If the location of a lookup service is unknown, it is necessary to make a multicast search for one. UDP supports a multicast mechanism which the current implementations of Jini use. Because multicast is expensive in terms of network requirements, most routers block multicast packets. This usually restricts multicast to a local area network, although this depends on the network configuration and the time-to-live of the multicast packets.

3.3 Service Proxy

A service will be delivered from out of a service provider. That is, a server will be started, to act as a service provider. It will create one or more objects which between them will implement the service. Amongst these will be a distinguished object - the service object. The service provider will register the service object with lookup service, and then wait for network requests to come in for the service. What the service provider will actually export as service object is usually a proxy for the service. The proxy is an object that will eventually run in a client, and will usually make calls back across the network to service backend objects. These backend objects running within the server actually complete the implementation of the service.

The proxy and the service backend objects are tightly integrated: they must communicate using a protocol known to them both, and must exchange information in an agreed manner. However, the relative size of each is up to the designer of a service and its proxy.

Basically, there are two kinds of proxy-fat proxy and thin proxy.

One extreme is where the proxy is so fat, which means it does a lot of processing on the client side. Backend object(s) within the service provider itself are then typically "thin", not doing much at all. The role of the server is to register the proxy with service locators, and just to stay alive (renewing leases on the service locators). The service itself runs entirely within the client. The opposite extreme is where the proxy

is thin, doing nothing more than passing requests between the client and "fat" backend objects, and most processing will be done by these backend objects running on the server side.

Some services can be implemented by a single object, the service object. How does this work if the service is actually a toaster, a printer, or controlling some piece of hardware? By the time the service object runs in the client's JVM, it may be a long way away from its hardware. It cannot control this remote piece of hardware all by itself. In this case, the implementation of the service must be made up of at least two objects, one running in the client and another distinct one running in the service provider. The service object is really a proxy, which will communicate back to other objects in the service provider, probably using RMI as shown in Figure 29. The proxy is the part of the service that is visible to clients, but its function will be to pass method calls back to the rest of the objects that form the total implementation of the service, the "service backend" objects. The proxy needs to communicate with other objects in the service provider.

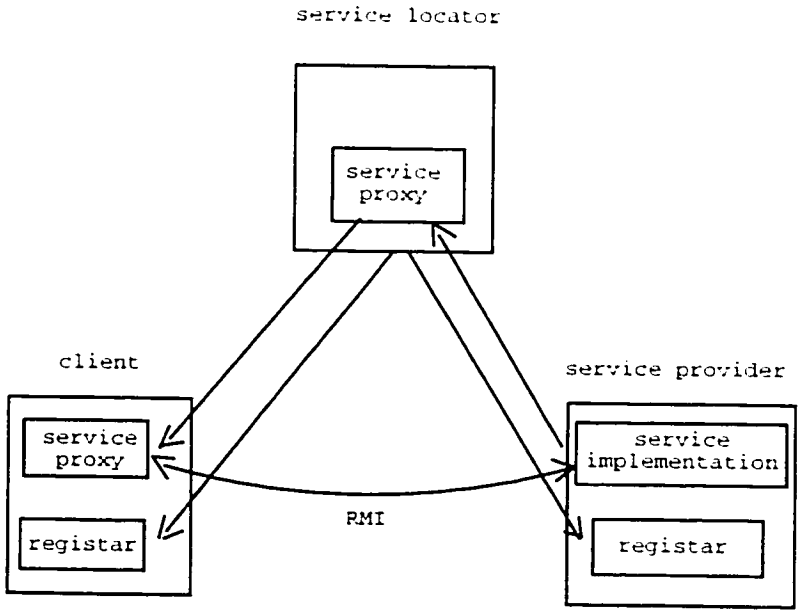


Figure 29: A Thin Proxy

How is the proxy primed? This is not specified by Jini, and can be done in a large number of ways. For example, an RMI naming service can be used such as rmiregistry, where the proxy is given the name of the service. This is not very common, as RMI proxies can be passed more directly as returned objects from method calls, and these can refer to ordinary RMI server objects or to RMI activatable objects. Or the proxy can be implemented without any direct use of RMI, and can then use an RMI exported service or some other protocol altogether such as FTP, HTTP, or a home-grown protocol.

3.4 The Structure of Jini

3.4.1 Client Structure

Internally, a client's pseudocode will include the following steps:

1) Prepare for Discovery

2) Discover a Lookup Service

3) Prepare a Template for Lookup Search

4) Lookup a Service

5) Call the Service

The client side code in Figure 40 is simplified from the real case, by omitting various checks on exceptions and other conditions. As an example, it attempts to find a BLAST service, and then calls the method getBlastResult() on this service. This is just to show how the above schema translates into actual code.

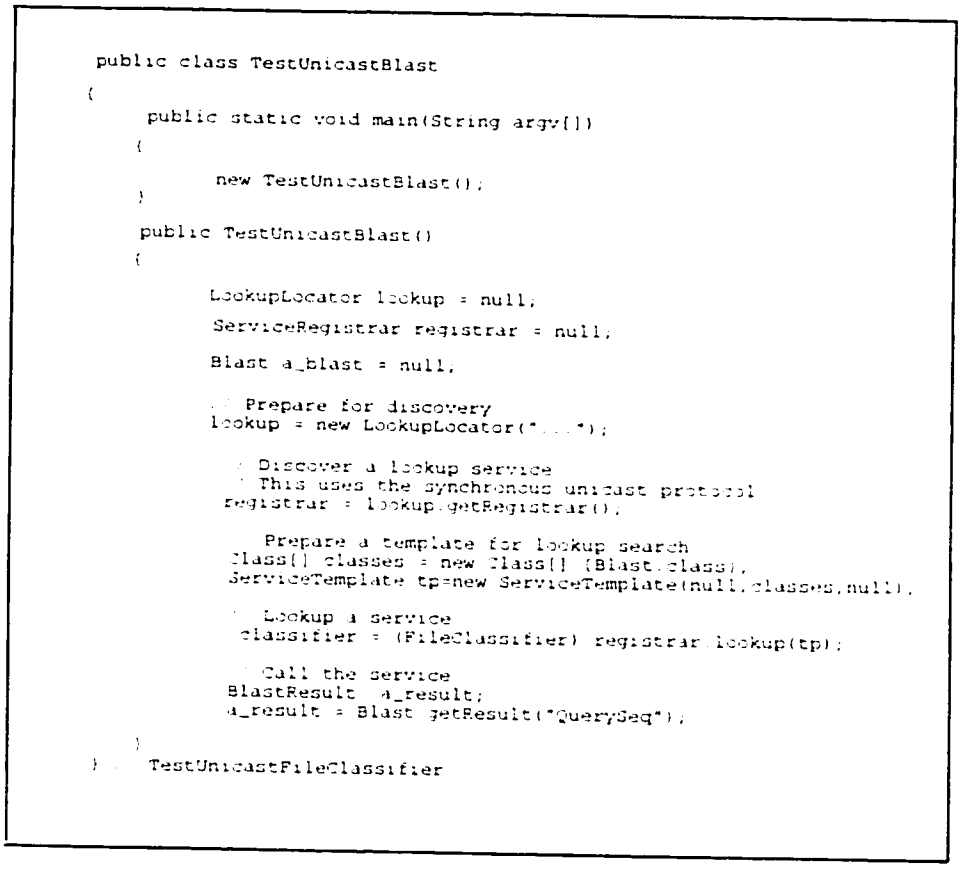
3.4.2 Server Structure

The pseudocode of a BLAST service on the server side will include the following steps:

1) Prepare for Discovery

2) Discover a Lookup Service

3) Create Information about a Service



```
public class TestUnicastBlast
{
     public static void main(String argv[])
    {
          new TestUnicastBlast();
    }
    public TestUnicastBlast()
    {
          LookupLocator lookup = null;
          ServiceRegistrar registrar = null;
          Blast a_blast = null;

          // Prepare for discovery
          lookup = new LookupLocator("...");

            // Discover a lookup service
            // This uses the synchronous unicast protocol
           registrar = lookup.getRegistrar();

             Prepare a template for lookup search
           Class[] classes = new Class[] {Blast.class};
           ServiceTemplate tp=new ServiceTemplate(null,classes,null);

            // Lookup a service
           classifier = (FileClassifier) registrar.lookup(tp);

             Call the service
          BlastResult a_result;
          a_result = Blast.getResult("QuerySeq");
    }
}   TestUnicastFileClassifier
```

Figure 30: Code Structure on Client Side

4) Export a Service

5) Renew Lease Periodically

The server side code is simplified from the real case, by omitting various checks on exceptions and other conditions. As an example, it exports an implementation of a BLAST service, a BlastImpl object. We do not give detailed explanations. This is just to show how the above schema translates into actual code in Figure 35.

```
public class BlastServer implements DiscoveryListener
{
     protected LeaseRenewalManager leaseManager = new LeaseRenewalManager();

    public static void main(String argv[])
    {
         new FileClassifierServer( );

            keep server running (almost) forever to
            allow time for locator discovery and
            keep re-registering the lease
         Thread.currentThread().sleep(Lease.FOREVER);

    }
    public BlastServer()
    {
         LookupDiscovery discover = null;

            Prepare for discovery - empty here
            Discover a lookup service
            This uses the asynchronous multicast protocol,
            which calls back into the discovered() method
         discover = new LookupDiscovery(LookupDiscovery.ALL_GROUPS);

         discover.addDiscoveryListener(this);
    }

    public void discovered(DiscoveryEvent evt)
    {
          ServiceRegistrar registrar = evt.getRegistrars()[0];
            At this point we have discovered a lookup service

            Create information about a service
         ServiceItem item = new ServiceItem(null, new BlastImpl(), null);

            Export a service
         ServiceRegistration reg = registrar.register(item, Lease.FOREVER);

            Renew leasing
         leaseManager.renewUntil(reg.getLease(), Lease.FOREVER, this);

    }
} // BlastServer
```

Figure 31: A Code Structure on Server Side

Chapter 4

BLAST Server Design

4.1 Introduction

Applying the Jini technology, we develop a BLAST service system. We separate our design into four parts. 1) Design for the service architecture. 2) Design for server side. 3) Design for client side. 4) Common classes on server and client sides.

4.2 Design for the Service Architecture

Because the BLAST service is involved in database queries, all of the processing is done on the server side. The proxy just exists on the client to take calls from the client, invoke the method in the service on the server, and return the result to the client. So we choose a thin proxy as shown in Figure 32.

Since the proxy need take calls from the client, the methods of the BLASTImpl are called remotely. The class BLASTImpl has to inherit the interface Remote. The BLASTImpl-Stub is generated from BLASTImpl by use of the rmic compiler. Inheriting from UnicastRemoteObject allows RMI to export the stub rather than the service, which remains on the server. The class structure for this is shown in Figure 33 and Figure 34.

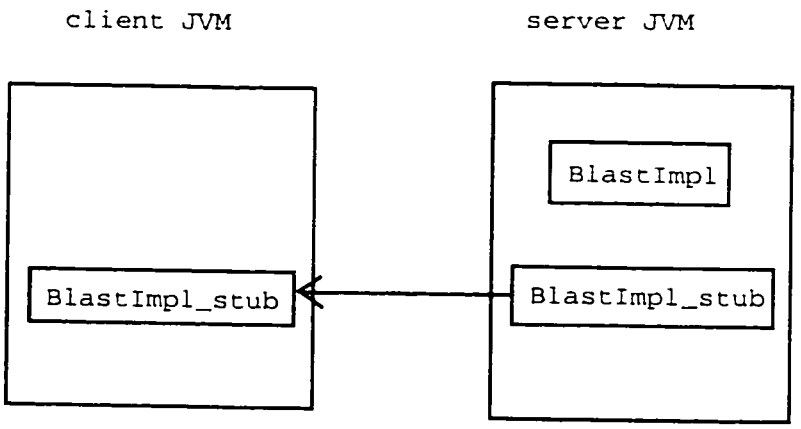


Figure 32: JVM Objects for RMI Proxy

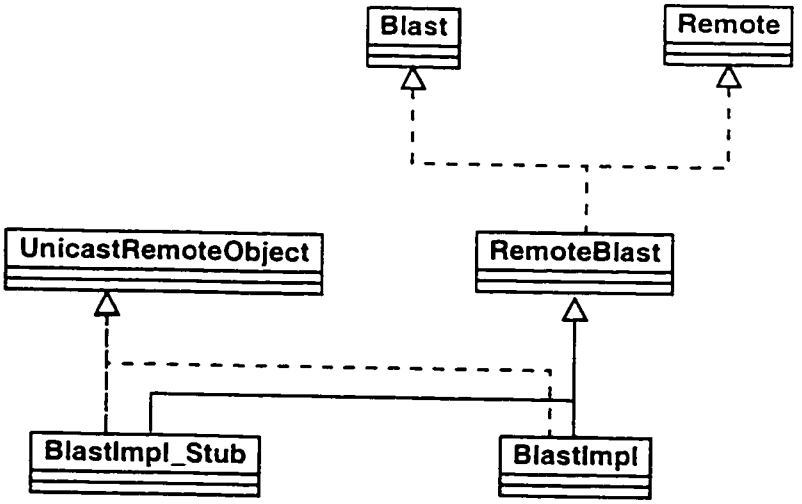


Figure 33: The Class Diagram for RMI Proxy

```
import common.Blast;
import java.rmi.Remote;

/*
RemoteBlast.java
*/
 public interface RemoteBlast implement Blast, Remote
{
}//RemoteBlast

 public class BlastImpl implements RemoteBlast, UnicastRemoteObject
{
   .....
}

 public BlastResult getBlastResult() java.rmi.RemoteException
{
   .....
}
```

Figure 34: A Class Structure in Java

4.3 Design for Server Side

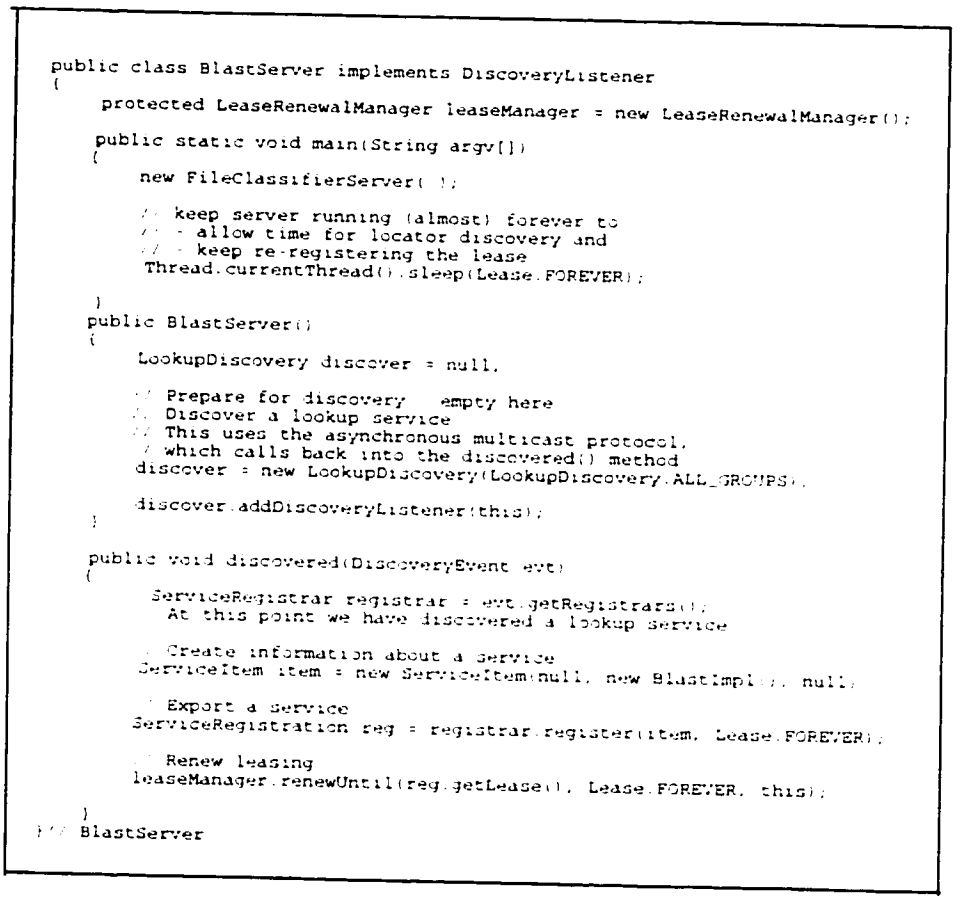
4.3.1 Register a Blast Service

The following code in Figure 35 is simplified from the real case, by omitting the issues about user interface, various checks on exceptions and other conditions. As an example, it exports an implementation of a BLAST service, a BlastImpl object.

4.3.2 Create a User Interface

To post the service, a proxy for BLAST service and user interface to enter a query are needed to post on the lookup service.

User interfaces are not yet part of the Jini standard. But the Jini community (with a semi-formal organization as the "Jini Community") is coming towards a standard way of specifying many things, including user-interface standards and guidelines. Guideline number one from the "serviceUI" group is: user interfaces for a service should be given in Entry objects. The Jini Entry class is designed to allow the additional information to be given in the request by adding extra objects.



```
public class BlastServer implements DiscoveryListener
{
     protected LeaseRenewalManager leaseManager = new LeaseRenewalManager();

    public static void main(String argv[])
    {
         new FileClassifierServer( );

         // keep server running (almost) forever to
         // - allow time for locator discovery and
         // - keep re-registering the lease
         Thread.currentThread().sleep(Lease.FOREVER);

    }
    public BlastServer()
    {
         LookupDiscovery discover = null;

         // Prepare for discovery   empty here
         // Discover a lookup service
         // This uses the asynchronous multicast protocol,
         // which calls back into the discovered() method
         discover = new LookupDiscovery(LookupDiscovery.ALL_GROUPS);

         discover.addDiscoveryListener(this);
    }

    public void discovered(DiscoveryEvent evt)
    {
          ServiceRegistrar registrar = evt.getRegistrars()[0];
          // At this point we have discovered a lookup service

          // Create information about a service
         ServiceItem item = new ServiceItem(null, new BlastImpl(), null);

          // Export a service
         ServiceRegistration reg = registrar.register(item, Lease.FOREVER);

          // Renew leasing
         leaseManager.renewUntil(reg.getLease(), Lease.FOREVER, this);

    }
} // BlastServer
```

Figure 35: A Code Structure on Server Side

To minimize the amount of serialized code that must be shipped around, the user interface is created as much as possible on the client side. So we created user interface on the client side but on the server side. So the user-interface should be exported as a factory object that can create the user interface on the client side. The factory imports the minimum number of classes for the interface to compile and be exported. The other general cless can be get from client VM. A typical Frame factory is the one that returns a JFrame. Here a factory object, a FrameFactory object, is created on the server side and exported to create user interface on the client side, as shown in Figure 36 and Figure 37.

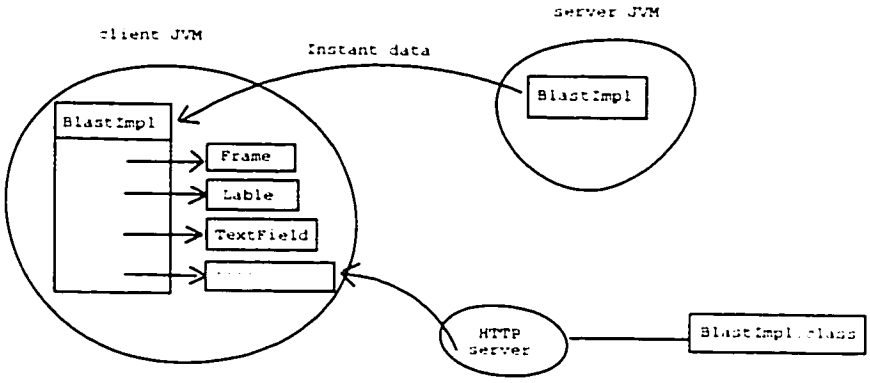


Figure 36: A FrameFactory Object on Client and Server Sides

In the sequenceEntryFrame class, we do not call setvisible(), but pack() as shown in Figure 38.

If a client receives a ServiceItem containing entries with many factory implementation objects, it will need to download the class files for all of these, as it instantiates the entry objects. There is a strong chance that each factory may be bundled into a jar file that also contains the user interface objects themselves. So if the entries directly contain the factories, then the client will need to download a set of class files, before it even goes about the business of deciding which of the possible user interfaces it wants to select.

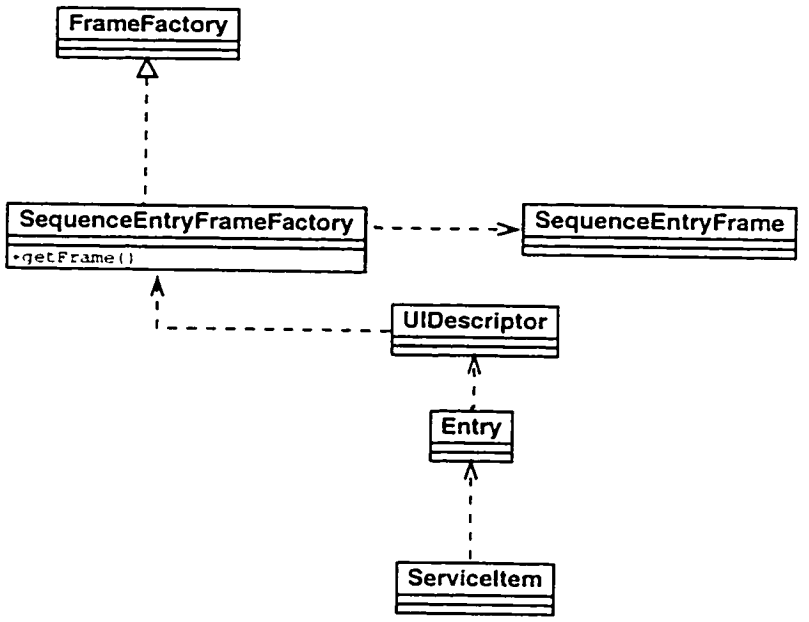


Figure 37: A Class Diagram for Sequence EntryFrameFactory

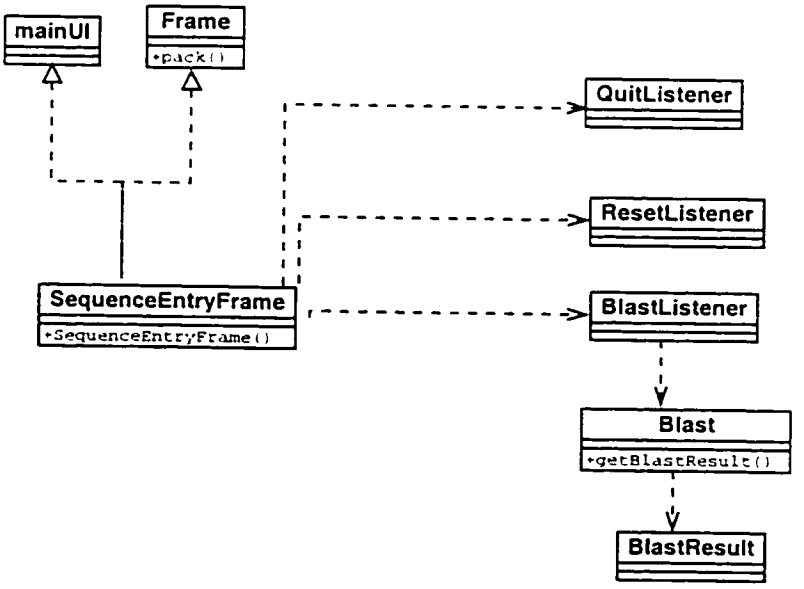


Figure 38: A Class Diagram for SequenceEntryFrame

This downloading may take time on a slow connection, such as wireless or home network link. It may also cost memory, which may be scarce in small devices. So it is advantageous to hide the actual factory classes until the client has decided that it does in fact want a particular class. Then, of course, it will have to download all of the class files needed by that factory.

In order to hide the factories, they are wrapped in a marshalledObject as shown in Figure 39. This keeps a representation of the factory, and also a reference to its codebase, so that when it is unwrapped the necessary classes can be located and downloaded. By putting it into entries in this form, no attempt is made to download its classes until it is unmarshalled.

The decision as to whether or not to unmarshall a class can be made on a separate piece of information, such as a set of strings which hold the names of the factory class.

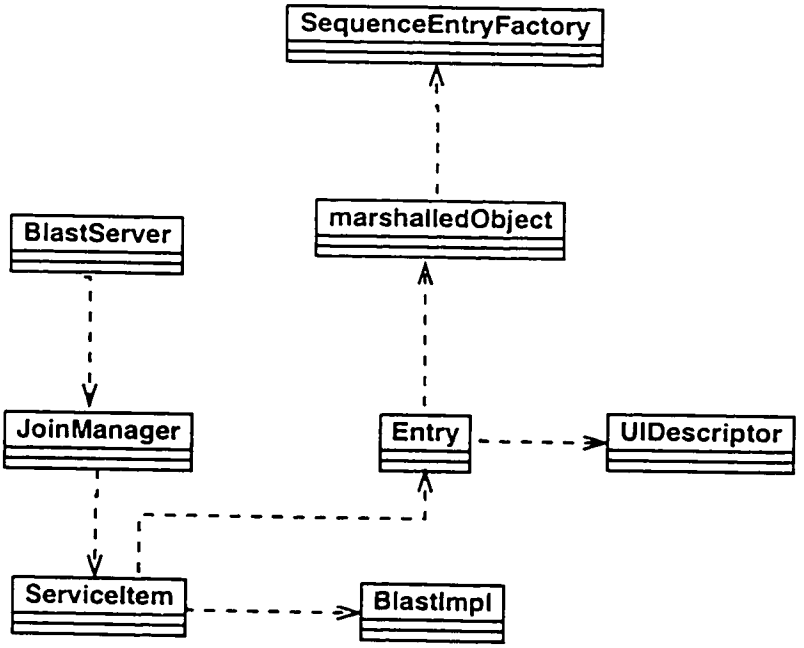


Figure 39: A Class Diagram for BLASTServer

4.4 Design for Client Side

4.4.1 Retrieve a BLAST Service

The client side code in Figure 40 is simplified from the real case, by omitting the issues about user interface, various checks on exceptions and other conditions. As an example, it attempts to find a BLAST service, and then calls the method getBlastResult() on this service.

```
public class TestUnicastBlast
{
     public static void main(String argv[])
    {
          new TestUnicastBlast();
    }
    public TestUnicastBlast()
    {
          LookupLocator lookup = null;
          ServiceRegistrar registrar = null;
          Blast a_blast = null;

           // Prepare for discovery
          lookup = new LookupLocator("...");

             // Discover a lookup service
             // This uses the synchronous unicast protocol
           registrar = lookup.getRegistrar();

              // Prepare a template for lookup search
           Class[] classes = new Class[] {Blast.class};
           ServiceTemplate tp=new ServiceTemplate(null,classes,null);

              // Lookup a service
            classifier = (FileClassifier) registrar.lookup(tp);

           // Call the service
           BlastResult  a_result;
           a_result = Blast.getResult("QuerySeq");

    }
} // TestUnicastFileClassifier
```

Figure 40: Code Structure on Client Side

4.4.2 Get a User Interface

To show a user interface on the client side, a client calls the function getUIFactory() to get the Framefactory object and call the function getFrame() to get the user interface. Finally, the client call the function setVisible() to the frame to show the interface. A class diagram for a client is shown in Figure 41.

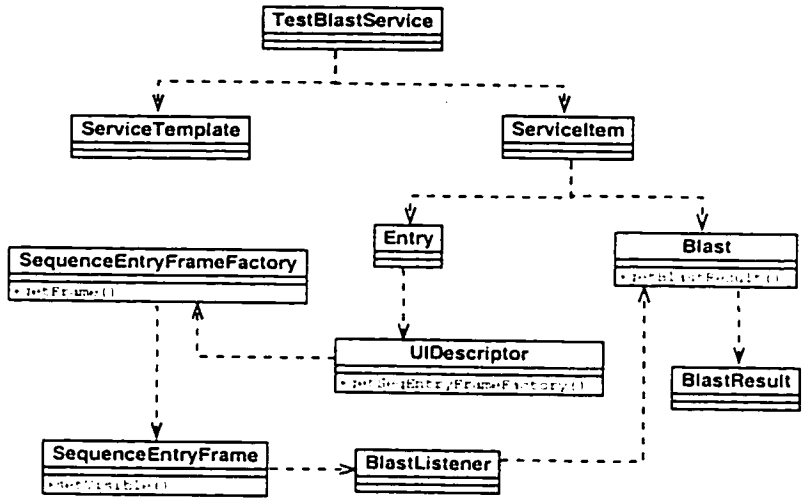


Figure 41: A Class Diagram for a Client

4.5 Common Classes on Server and Client Sides

Because the service is implemented remotely and runs in a separate JVM, then the BLASTResult object must be serialized for transport to the client JVM. For this to be possible, it must implement the Serializable interface. While the class files are accessible to both client and service, the instance data of the BLASTResult object needs to be serializable to move the object from one machine to the other, as shown in Figure 42.

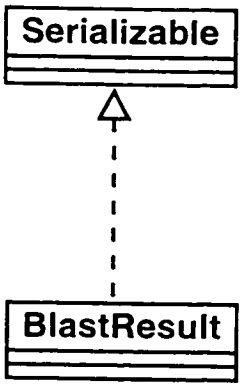


Figure 42: A Class Diagram for BLASTResult

Class BLASTResult is inherence by four classes, as shown in Figure 43. SummaryBR is used to represent the summary of the BLAST result as shown in the conceptual user interface, such as reference ID, number of letters in the query sequence, number of sequences in the database. GraphBR is used to present an overview of the BLAST result, allowing you to quickly identify the similar region. AlignmentBR is used to show the accession number, which can be used to retrieve this sequence from the database, clone name, which is assigned by the researcher. Clone description, the scores, which is linked to a diagram that shows the query sequence aligned with the matching sequence from the database. DBEntryBR is used to retrieve the other information from Medline.

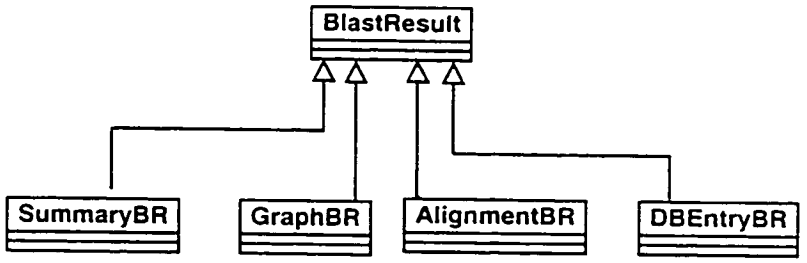


Figure 43: A Class Diagram for BLASTResult Hierarchy

The interface of the BLAST should be accessible for client and server. The method here is defined to throw a java.rmi.RemoteException. This type of exception is used throughout Java (not just the RMI component) to mean "a network error has occurred". This could be a lost connection, a missing server, a class not downloadable, etc. The interface is shown in Figure 44.

```
/**
Blast.java
*/

public interface Blast
{
    public BlastResult getBlastResult(string querySequence)
        throws java.rmi.RemoteException;

} Blast
```

Figure 44: The Interface of BLAST Class

Chapter 5

Conceptual User Interface Design

The conceptual user interface is based on the user interface of NCBI.

5.1 Input

The page is mainly to collect the information, which is needed for BLAST, such as the query sequence, the set sequence, and the chosen database. It also includes the BLAST button to submit your query, RESET button to reset the information you have input as shown in Figure 45.

5.2 Output

We separate the whole BLAST results into four parts. 1) SummaryBR is used to represent the summary of the BLAST result. 2) GraphBR is used to present an overview of the BLAST result, allowing you to quickly identify the similar region. 3) DBEntryBR is used to retrieve the other informatiom from datebase. 4) AlignmentBR is used to show the alignments in detail.

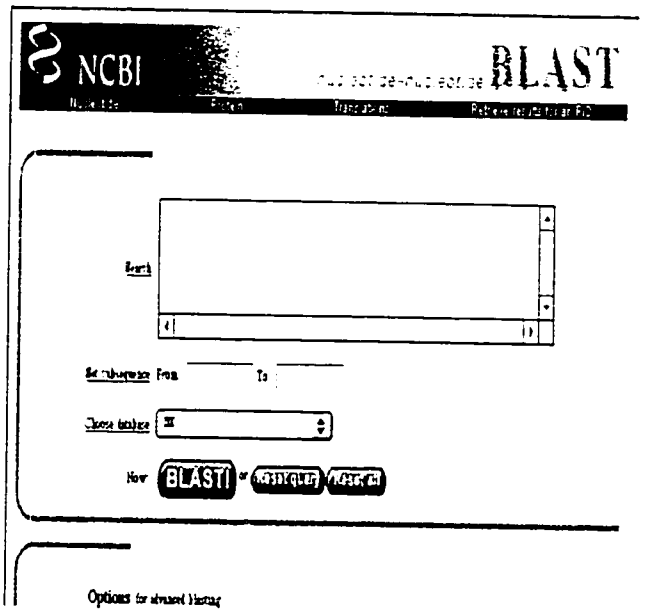


Figure 45: The Input for BLAST Search

5.2.1 SummaryBR

A SummaryBR is shown in Figure 46. It often includes: 1) The number of letters gives the length of the sequence that was posted in the box in the input page. 2) The number of sequence in database. 3) The number of letters corresponds to the total number of nucleotides in the database.

5.2.2 GraphBR

A GraphBR is shown in Figure 46. A graph is display to present an overview of the BLAST result, allowing you to quickly identify the similar regions. The query sequence is labeled and is drawn as a thick red bar. Numbers below the bar give the scales in bases. Sequences with the best match to the query sequence are shown in red. Low scoring sequences show up as black bars.

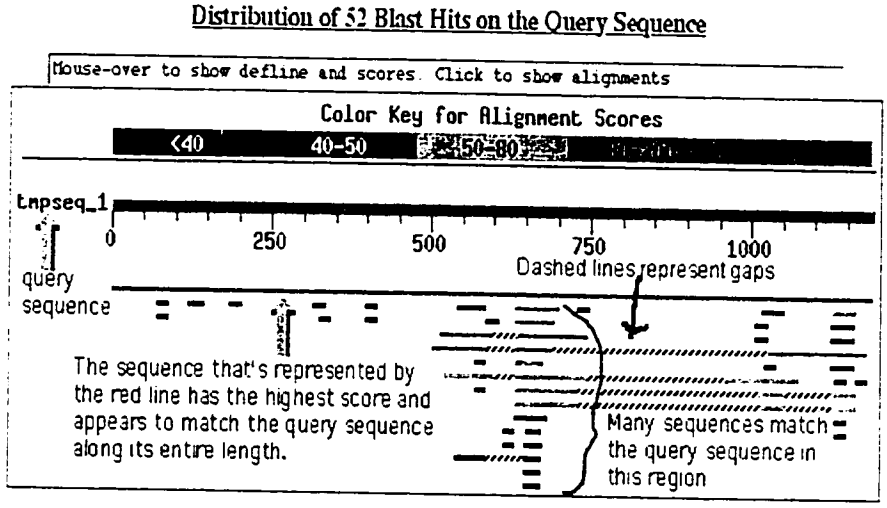


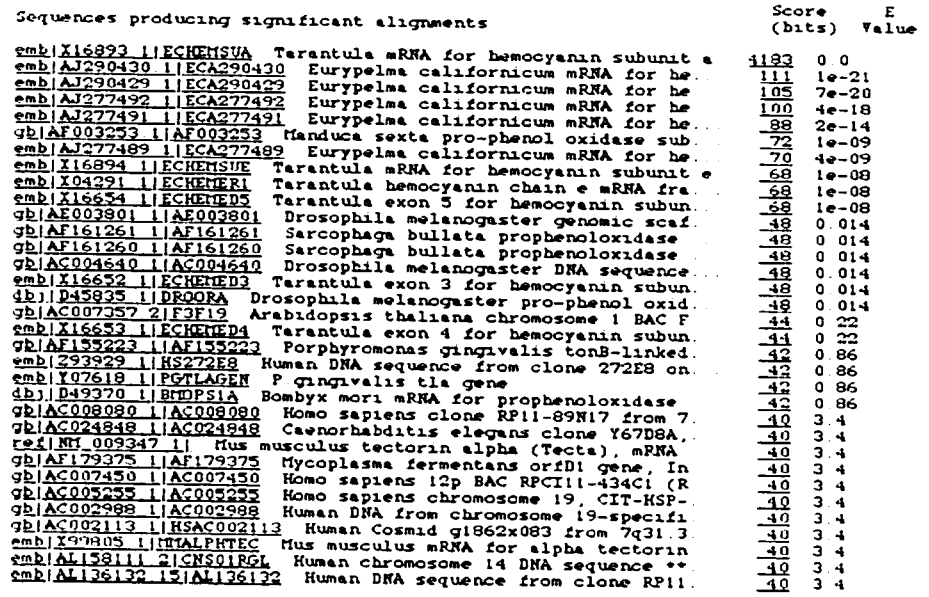
Figure 46: A Summary BLAST Result

5.2.3 DBEntryBR

A DBEntryBR is shown in Figure 47. It is used to retrieve the other information from databases. It includes some links such as following: 1) Link to the database served as the entry point for this sequence. 2) Accession number, which can be used to retrieve this sequence from the database. 3) The scores, which is linked to a diagram that shows the query sequence aligned with the matching sequence from the database.

5.2.4 AlignmentBR

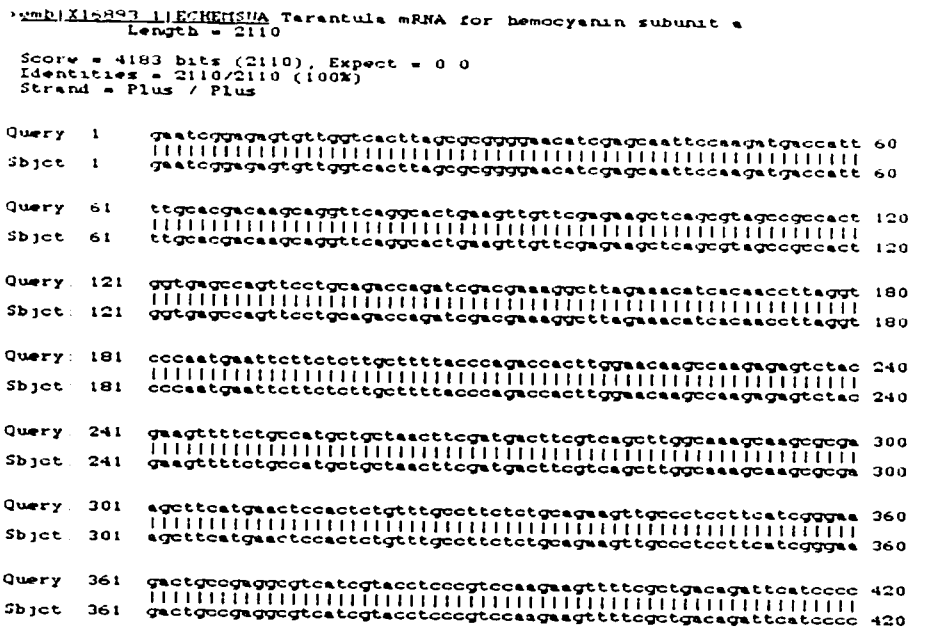
A AlignmentBR is used to show the alignments in detail. it is shown in Figure 48. It can include: 1) The score assigned by BLAST. In general, the higher the score, the better the match between the query sequence and a sequence in the database. 2) Identities. In the example, 100 percent of the nucleotides in a 2110 base stretch of the query sequence are identical to a 2110 nucleotide region in the sequence obtained from GenBank.



```
Sequences producing significant alignments                    Score (bits)   E Value

emb|X16893_1|ECHEMSUA  Tarantula mRNA for hemocyanin subunit a   4183   0.0
emb|AJ290430_1|ECA290430  Eurypelma californicum mRNA for he.     111   1e-21
emb|AJ290429_1|ECA290429  Eurypelma californicum mRNA for he      105   7e-20
emb|AJ277492_1|ECA277492  Eurypelma californicum mRNA for he      100   4e-18
emb|AJ277491_1|ECA277491  Eurypelma californicum mRNA for he..     88   2e-14
gb|AF003253_1|AF003253  Manduca sexta pro-phenol oxidase sub.     72   1e-09
emb|AJ277489_1|ECA277489  Eurypelma californicum mRNA for he.     70   4e-09
emb|X16894_1|ECHEMSUE  Tarantula mRNA for hemocyanin subunit e    68   1e-08
emb|X04291_1|ECHEMER1  Tarantula hemocyanin chain e mRNA fra.     68   1e-08
emb|X16654_1|ECHEMED5  Tarantula exon 5 for hemocyanin subun.     68   1e-08
gb|AE003801_1|AE003801  Drosophila melanogaster genomic scaf.     48   0.014
gb|AF161261_1|AF161261  Sarcophaga bullata prophenoloxidase       48   0.014
gb|AF161260_1|AF161260  Sarcophaga bullata prophenoloxidase       48   0.014
gb|AC004640_1|AC004640  Drosophila melanogaster DNA sequence..    48   0.014
emb|X16652_1|ECHEMED3  Tarantula exon 3 for hemocyanin subun.     48   0.014
dbj|D45835_1|DROORA  Drosophila melanogaster pro-phenol oxid.     48   0.014
gb|AC007357_2|F3F19  Arabidopsis thaliana chromosome 1 BAC F      44   0.22
emb|X16653_1|ECHEMED4  Tarantula exon 4 for hemocyanin subun.     44   0.22
gb|AF155223_1|AF155223  Porphyromonas gingivalis tonB-linked.     42   0.86
emb|Z93929_1|HS272E8  Human DNA sequence from clone 272E8 on.     42   0.86
emb|Y07618_1|PGTLAGEN  P gingivalis tla gene                      42   0.86
dbj|D49370_1|BMDPS1A  Bombyx mori mRNA for prophenoloxidase       42   0.86
gb|AC008080_1|AC008080  Homo sapiens clone RP11-89N17 from 7.     40   3.4
gb|AC024848_1|AC024848  Caenorhabditis elegans clone Y67D8A,.     40   3.4
ref|NM_009347_1|  Mus musculus tectorin alpha (Tecta), mRNA       40   3.4
gb|AF179375_1|AF179375  Mycoplasma fermentans orfD1 gene, In      40   3.4
gb|AC007450_1|AC007450  Homo sapiens 12p BAC RPCI11-434C1 (R      40   3.4
gb|AC005255_1|AC005255  Homo sapiens chromosome 19, CIT-HSP-.     40   3.4
gb|AC002988_1|AC002988  Human DNA from chromosome 19-specifi.     40   3.4
gb|AC002113_1|HSAC002113  Human Cosmid g1862x083 from 7q31.3.     40   3.4
emb|X99805_1|MMALPHTEC  Mus musculus mRNA for alpha tectorin      40   3.4
emb|AL158111_2|CNS01RGL  Human chromosome 14 DNA sequence **.     40   3.4
emb|AL136132_15|AL136132  Human DNA sequence from clone RP11.     40   3.4
```

Figure 47: The Database Entries in BLAST Result



```
>emb|X16893_1|ECHEMSUA Tarantula mRNA for hemocyanin subunit a
          Length = 2110

 Score = 4183 bits (2110), Expect = 0.0
 Identities = 2110/2110 (100%)
 Strand = Plus / Plus

Query  1     gaatcggagagtgttggtcacttagcgcgggggaacatcgagcaattccaagatgaccatt 60
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  1     gaatcggagagtgttggtcacttagcgcgggggaacatcgagcaattccaagatgaccatt 60

Query  61    ttgcacgacaagcaggttcaggcactgaagttgttcgagaagctcagcgtagccgccact 120
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  61    ttgcacgacaagcaggttcaggcactgaagttgttcgagaagctcagcgtagccgccact 120

Query  121   ggtgagccagttcctgcagaccagatcgacgaaaggcttagaaacatcacaaccttaggt 180
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  121   ggtgagccagttcctgcagaccagatcgacgaaaggcttagaaacatcacaaccttaggt 180

Query  181   cccaatgaattcttctcttgcttttacccagaccacttggaacaagccaagagagtctac 240
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  181   cccaatgaattcttctcttgcttttacccagaccacttggaacaagccaagagagtctac 240

Query  241   gaagttttctgccatgctgctaacttcgatgacttcgtcagcttggcaaagcaagcgcga 300
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  241   gaagttttctgccatgctgctaacttcgatgacttcgtcagcttggcaaagcaagcgcga 300

Query  301   agcttcatgaactccactctgtttgccttctctgcagaagttgccctccttcatcgggaa 360
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  301   agcttcatgaactccactctgtttgccttctctgcagaagttgccctccttcatcgggaa 360

Query  361   gactgccgaggcgtcatcgtacctcccgtccaagaagttttcgctgacagattcatcccc 420
             ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  361   gactgccgaggcgtcatcgtacctcccgtccaagaagttttcgctgacagattcatcccc 420
```

Figure 48: The Detailed Alignments

Chapter 6

Summary and Future Work

Here, a distributed system is designed for BLAST service with Jini network technology. Currently designing is a simple system, which can only process the requests form clients synchronously. The future work to improve the system will include followings:

1) Supporting the nation of an analysis pipeline As the ability of scientific investigation to produce large numbers of such sequences has become mainstream, the ability to process, store, and analyze them becomes very inportant. The notion of an analysis pipeline for processing and analyzing large batches of sequences has risen recently. The sequence analysis pipeline consists of an ordered set of processing stages, each of which does some well-understood operation on its input data, and produces output data that may be used by some subsequent stage. With the idea, a sequence analysis pipeline can logically be broken down into these components: DNA analysis methods, database support, the distributed analysis server, the user interface, and so on.

2) Handling the requests from clients asynchronously To handle multiple requests from clients, the system receives these requests from clients and places them on a request queue, from which requests are dispatched to be processed.

3) Using distributed processors to increase the throughput To increase the utility of multiple processors system, parallel processing the computationally intensive tasks in the sequence analysis portions of the pipeline, such as similarity searching, we can distributed the analysis tasks over multiple processors. Since it is more valuable for the distributed analysis system to maximize throughput, rather than to provide the minimal turnaround time for a given job, scheduling multiple requests concurrently should be considered.

Bibliography

[1] Bruno A. Gaeta, **Internet On-Ramp.** *BioTechniques*, 28(3) (2000) 436–438.

[2] Altschul, SF., Gish, W., Miller, W., Myers, EW. and Lipman, DJ., **Basic local alignment search tool.** *J. of Mol. Biol.* 215(1990) 403–10.

[3] **Jini network technology.** URL http://www.sun.com/jini/

[4] **What is Jini Network Technology?**
URL http://www.jini.org/whatisjini.html

[5] Waugh, M., Hraber, P., Weller, J., Wu, Y., Chen, G., Inman, J., Kiphart, D., and Sobral, B., **The Phytophthora Genome Initiative Database: Informatics and Analysis for Distributed Pathogenomic Research.** *Nucleic Acids Research*, 28 (2000) 87–90.

[6] Inman, J. T., Flores, H. R., May, G. D., Weller, J. W., and Bell, C. J., **A high-throughput distributed DNA sequence analysis and database system.** *IBM System Journal*, 40(3) (2001) 464–486.

[7] Newmarch, J., **Jan Newmarch's Guide to Jini Technology.** URL http://jan.netcomp.monash.edu.au/java/jini/tutorial/Jini.xml

[8] Allison, L., Wallace, C. S. and Yee, C. N., **When is a String like a String?** In *AI Maths 1990(a)* .

[9] Needleman, S. B. and Wunsch, C. D., **A general method applicable to the search for similarities in the amino acid sequences of two proteins.** *Journal of Molecular Biology*, 48(1970) 443–453

[10] Sellers, P. H., **On the Theory and computation of evolutionary distances.** *SIAM J. Appl. Math*, 26(1974) 787–793.

[11] Smith, T. F. and Waterman, M. S., **Identification of common molecular subsequences.** *Journal of Molecular Biology*,147(1981) 195–197.

[12] Pearson, W. R. and Lipman, D. J., **Improved tools for Biological Sequence Comparison.** *Proc. Natl. Acad. Sci. USA*, 85(1998) 2444–2448.

[13] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J., **Basic local alignment search tool.** *Journal of Molecular Biology*, 215(1990) 403–410.

[14] Karlin, S. and Altschul, S. F., **Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes.** *Proc. Natl. Acad. Sci. USA*, 87(1990) 2264–2268.

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

[16] Zhang, Z., Schaffer, AA., Miller, W., Madden, TL., Lipman, DJ., Koonin, EV., and Altschul, SF., **Protein sequence similarity searches using patterns as seeds.** *Nucleic Acids Res*, 26(1998) 3986–90.

[17] **WU-BLAST 2.0 Topics.** URL http://blast.wustl.edu/blast/README.html

[18] Hughey, R., **Parallel hardware for sequence comparison and alignment.** *CABIOS*, 26(17)(1998) 3986–90.

[19] Intel Corp., **Intel Architecture Software developer's manual.** *Instruction Set Reference*, 12(1996) 473-479.

[20] Rognes, T., **ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches.** *Nucleic Acids Res.* 29(2001) 1647-52

[21] **TurboBLAST**
URL http://www.turbogenomics.com/products/turboblast-overview.html

[22] Benson, DA., Karsch-Mizrachi, I., Lipman, DJ., Ostell, J., Rapp, BA., and Wheeler, DL., **GenBank.** *Nucleic Acids Res*, 28(2000) 15-18.

[23] Stoesser, G., Baker, W., Broek, A., Camon, E., Garcia-Pastor, M., et al., **The EMBL nucleotide sequence database.** *Nucleic Acids Res.* 29(2000) 17-21.

[24] Tateno, Y., Miyazaki, S., Ota, M., Sugawara, H., and Gojobori, T., **DNA Data Bank of Japan (DDBJ) in collaboration with mass sequencing teams.** *Nucleic Acids Res.* 28(2000) 24-26.

[25] **Boulder Data Interchange Format**
URL http://stein.cshl.org/software/boulder/

[26] **Bioperl** URL http://www.bioperl.org/Core/POD/Bio/Tools/BPlite.html