A high-throughput distributed DNA sequence analysis and database system

by J. T. Inman
H. R. Flores
G. D. May
J. W. Weller
C. J. Bell

The National Center for Genome Resources (NCGR) has developed a high-throughput DNA (deoxyribonucleic acid) sequence analysis pipeline, which allows researchers at remote sites to submit biological sequence information for rapid analysis, the results of which can be queried through a Web interface. Behind the browser interface is a relational database used to manage both the raw data and the results of the different analyses performed, and a server, which performs those analyses. The system allows multiple contributors to submit data and also allows the data to be marked as "private" or as available to the general public. The CPU-intensive part of the processing is done on a 40-processor domain of a Sun Enterprise 10000 computer, which is represented by a distributed system of software objects, implemented in CORBA™ (Common Object Request Broker Architecture™). In this paper we discuss the architecture of the pipeline, the database support, types of DNA sequence analysis used, the distributed analysis system, and the capabilities of the Web interface. As a case study, we present data from an ongoing collaborative project in which expressed sequence tags (ESTs) from* Medicago truncatula *are being processed.* M. truncatula *is a plant that is used as a research model for crops in the legume family, an economically important group of food and forage plants.

The development of improved DNA (deoxyribonucleic acid) sequencing technologies in the 1980s and 1990s heralded the start of a new era in biology, in which it has become possible to examine organisms at the most fundamental level: the set of instructions that specify their development, form, function, and behavior. By analogy with computer programming, just as (typical) program code is dependent on run-time interactions with its environment and is executed in a specific order (often with variations in behavior depending on input data, or on asynchronous events), the "specification" of an organism depends on the orderly expression of genes in time and space, with the expression of certain genes being influenced by environmental factors.

Knowledge of which genes are expressed in a given organ or tissue, at a given developmental stage, under given conditions, can yield scientific insight into biological processes of all kinds. Surveys of this information, obtained by random sequencing of expressed genes from specific stages of development or as the result of influence of specific environmental factors has become a mainstay of modern experimentation.

As the ability of scientific investigation to produce large numbers of such sequences has become mainstream, the ability to process, store, and analyze them has generally not kept pace. The notion of an "analysis pipeline" for processing and analyzing large batches of sequences has risen independently more than once. The System for Easy Analysis of Lots of Sequences (SEALS)[1,2] and the Boulder data interchange format[3] are both solutions for controlling sequence analysis operations and using the output of

0018-8670/01/$5.00 © 2001 IBM

one operation as the input for another. The University of Washington tools for calling bases of raw sequencing data, masking repetitive elements, and clustering ESTs[4] into groups are further examples.[5–7] These are practical, widely used solutions that 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 laboratories that specialize in DNA sequencing.

The National Center for Genome Resources (NCGR) has implemented such a system in which researchers submit raw data by file transfer protocol (FTP), and essentially all computation is done by servers at NCGR. The users' subsequent interactions with their data are via the World Wide Web (WWW), and the results of data analysis are viewed using a browser. The system we describe here is called MGI (*Medicago* Genome Initiative), a collaboration between NCGR and the Samuel Roberts Noble Foundation. It is one aspect of bioinformatics support for the Noble Foundation Center for Medicago Genomics Research, which is an integrated program aimed at improving our understanding of plants in the family *Fabaceae*.[8] This group of plants comprises the pea and bean family, and includes economically important species such as soybean and alfalfa. As such it is a crucial protein source for humans and animals.

Although the system currently contains only cDNA (complementary DNA) sequences from *Medicago truncatula*, the species of origin for each sequence is stored in the database. This permits the system to be expanded to handle the analyses of multiple species, and can be scaled to contain many hundreds of thousands of sequences. With such numbers, the sequence analysis portions of the pipeline, such as similarity searching, could not keep up with the growth of the database if the analyses were performed on a single processor. We have solved this problem by developing the distributed analysis server, which efficiently distributes the analysis tasks over a 40-processor domain of a 64-CPU symmetric parallel processor. This development greatly increases the utility of the system, and is described at length. The analysis pipeline can logically be broken down into these components: DNA analysis methods, database support, the distributed analysis server, and the user interface. These are considered in turn.

Analysis pipeline

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. The execution of operations in the pipeline is partially controlled by data that are themselves stored in tables in the database. Object-oriented software has been written that interacts generically with these tables. Subclasses of the relevant software objects can be developed to implement many types of pipeline stages.

In the particular instance of pipeline described here, the data that serve as input to the initial stage arrive from the sequencing group at the Samuel Roberts Noble Foundation via FTP to a secure location at NCGR. Sequences are deposited in a popular format called FASTA, after a well-known algorithm of the same name. FASTA's input format has become one of the *de facto* standards for representing sequence data. For each sequence file, a quality-score file is optionally also deposited, consisting of a space-delimited array of numerical scores. Each score indicates a confidence rating for the identity of the corresponding nucleotide (i.e., an A [adenine], T [thymine], G [guanine], or C [cytosine] in the DNA string), and is assigned automatically by software that interprets the digital output of an automated DNA sequencer. Two widely used base-calling tools are PHRED[5] and TraceTuner (http://www.paracel.com/tracetuner/index.html). The sequence and quality files are both represented in ASCII (American National Standard Code for Information Interchange).

An earlier design and implementation of the pipeline has already been described,[9] using a simpler architecture than the one we are presenting here. Accordingly, the older features will be covered only in outline, allowing coverage of the new features in greater detail. The features that have been added to the earlier model include: tracking whether a sequence has been published to GenBank,[10] policing sequence quality using PHRED or TraceTuner quality scores, allowing users to query the database interactively using the BLAST[11] family of algorithms, allowing users to view sequences interactively based on the cDNA library of origin, and adding the distributed analysis system for increased throughput in computationally intensive tasks. The control for a

given pipeline is implemented as a shell script, which invokes the desired analysis stages in order, passing along relevant parameters. The scripts are then scheduled as jobs for "cron," which is a UNIX system utility allowing jobs to be invoked automatically at specified times. The MGI pipeline runs nightly, and consists of the five stages we now describe.

Gatherer. This program checks the FTP site for new sequence and quality-score files, and performs basic quality checks, such as making sure that the file names conform to a certain standard and that the sequence and quality-score files contain the same number of elements. Validated raw sequences in the form of text strings and quality scores in the form of space-delimited arrays of numbers are then entered into a Sybase** database. In addition, a filename convention is used (for the sequence and quality-score data files) that includes a code for the type of clone represented by the sequence data, and this is also recorded in the database.

Vector screen. A consequence of the sequencing methodology (see sidebar: cDNA Cloning and Sequencing) is that the nucleotide strings of interest may contain short stretches of contaminating vector sequence that need to be identified and removed. Additionally, an unavoidable problem with the sequencing technology is the occurrence of poor-quality sequence at the end of the sequence string (and sometimes at the beginning). Depending on the software that is used to call the bases, this may be manifested by the presence of nucleotides that cannot be reliably assigned A, G, T, or C, and may therefore be given the letter N (which represents an unknown nucleotide) by some versions of base calling software. Regardless of whether they are called as Ns or not, a certain quantity of these poorly identified nucleotides is tolerable, but it is desirable to identify and remove stretches of sequence that have fallen below acceptable levels. The removal of the trailing poor-quality stretch of sequence is achieved by the application of a simple heuristic, discarding nucleotides after the point where a span of 20 nucleotides is found with an average quality-score below a threshold (in the case of TraceTuner, we use a threshold of 15). In cases where no quality scores are available, the presence of three Ns within a span of 20 nucleotides is a fairly reliable indicator of the point at which the sequence quality can be expected to degrade rapidly, and this is used to make the cut. This tends not to be triggered by the acceptable occurrence of a few Ns at the start of a good-quality stretch of sequence.

In the vector-screen stage, sequences are first scanned for the presence of vector contamination using BLASTN, which is a member of the BLAST family of algorithms. To screen for vector contamination, the sequence of interest is compared to a library of sequences that represent the vector organism, and regions where the two match each other with high confidence are identified and removed. The library of vector sequences is constructed based on experimental details from the Noble Foundation collaborators, and must therefore be modified if a new vector is introduced. By virtue of the way the cloning technology works, the vector sequences are expected at the beginning or occasionally at the end of the sequence. Cases where vector sequence is found within nonvector sequence probably represent cloning artifacts, and these are set aside for further study, and not subjected to further processing in the pipeline.

The sequence is then scanned from left to right for the presence of one of the low-quality markers. If either quality-control pattern is found, then it and the remainder of the sequence are clipped off. If the good part of the sequence happens to be 50 nucleotides long, or less, this is considered too small to allow effective analysis; the processing of that particular sequence is terminated at this point, and a note to that effect can be seen via the Web interface.

Figure 1 shows a sample of typical sequence quality scores, in this case from the MGI cDNA library that was prepared from phosphate-starved leaves. This figure illustrates the need for trimming based on quality scores, but also shows that a naïve application might reject many sequences prematurely, based on low-quality scores at the beginning of the sequence.

The vector screen stage also identifies (within the sequence of interest) other synthetic sequences left over from the cloning process, namely the restriction-enzyme recognition sites and an adapter sequence that were used to insert the sequence into the vector. After being screened, sequences that continue in the pipeline are assured of having an acceptable standard of quality and are free of contaminating vector.

Similarity searching. One of the main functions of the system is to assist users in identifying the function of the gene products, a milestone on the road to gaining insight into biological processes in the tis-

cDNA CLONING AND SEQUENCING

The effectors of the instruction set specifying an organism (enzymes and other proteins, often called the gene products) are encoded in the deoxyribonucleic acid (DNA) of the chromosomes. The instructions are executed via the production of messenger ribonucleic acid (mRNA) molecules that can be considered the intermediates between the DNA code and the protein effectors. Although a considerable simplification, it is useful to think of genes as discrete segments of DNA, each of which encodes a single protein, with a cognate mRNA molecule as an intermediate. This directional flow of genetic information is known as the "central dogma" of molecular biology and the execution of the code (the production of both mRNA and protein from a given gene) is generally called "gene expression." Messenger RNA is one of several types of RNA found in a cell. The others mainly contribute to the structure of certain cellular components, and do not encode proteins. The mRNA molecules differ from other types of RNA in that they have a "tail"consisting of a string of adenines (one of the four letters in the RNA alphabet) at one end, and this feature is used to extract the mRNA away from the other RNA types. The mRNA is then copied into a cognate DNA molecule by the process of reverse transcription. The DNA thus produced is called complementary DNA, or cDNA. This process achieves two things. DNA is a much more stable molecule than RNA, and the cDNA is therefore easier to work with. The cDNA is also double stranded, meaning that the information-coding strand pairs by hydrogen bonding with another strand, in which the base adenine always lines up with thymine, and cytosine with guanine. The double strandedness of cDNA allows it to be treated as any other DNA molecule in a living system—when introduced into a bacterial cell it can be propagated as if it were bacterial DNA, and many millions of copies made from one initial starting molecule. This multiplication of foreign DNA molecules in a bacterial system is called molecular cloning, and is done in practice by inserting the cDNA into a specially modified circular molecule called a plasmid (generically known as a cloning vector), introducing the plasmid into a bacterial cell, and propagating the bacteria. What we call a cDNA library is a large collection of bacterial strains, each containing a plasmid derived from one cDNA molecule, in turn derived from one mRNA molecule. The cDNA is extracted from each strain, and its DNA sequence is determined. The process used to determine the sequence is too detailed to cover here, but it is usually done in an automated instrument, and the output is parsed by software that examines the data and determines the identity (A, T, C, or G) of each base. This latter process is known as "base calling," which is a process in which a quality score is assigned to each base call, where the score is proportional to the probability of the call being correct. Sequence runs of cDNA clones result in sequences of a few hundred to a thousand base pairs. These are known generically as expressed sequence tags (ESTs).[4]

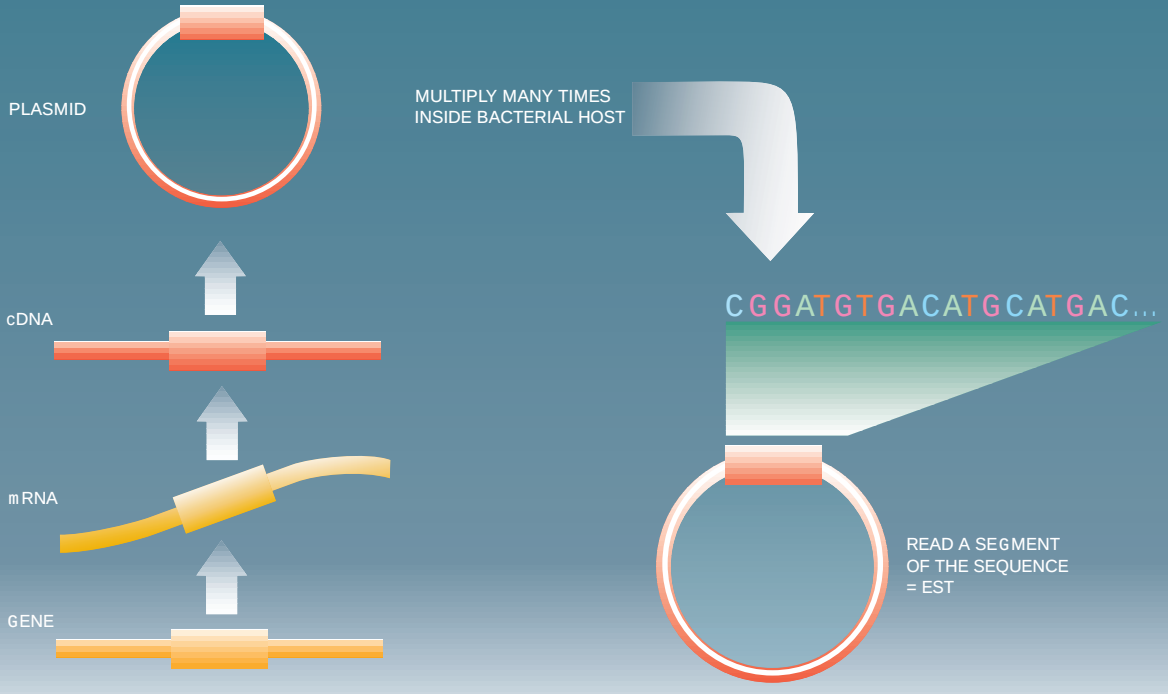
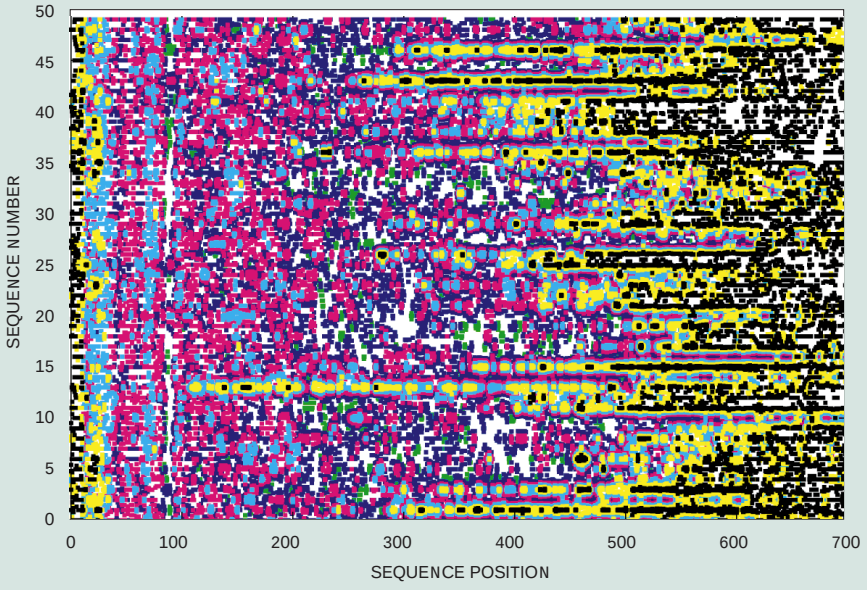


Figure 1 Typical sequence quality scores in the MGI database. Note that quality scores are often low near the beginning of the sequence, as well as toward the end. The data for each sequence have been truncated (or padded with zeros) to show a consistent length. (Black: <10, yellow: <15, sky: <20, magenta: <25, navy: <30, green: <35)



sue from which the cDNA library was made. Assigning a tentative function to gene products is generally called "functional annotation." There are several frequently used approaches to functional annotation, but the most heavily used one takes advantage of the massive, and expanding, number of sequences in public databases that have actual or predicted functions already assigned to them.

In the similarity searching stage(s), the system uses BLAST (the same algorithm used in the vector screen stage) to compare each vector-screened sequence to the nonredundant protein database maintained by the National Center for Biotechnology Information (NCBI).[12] The results of the database search are saved (as both raw text, and parsed into details) to the MGI database. The intuition underlying this search is that sequences that are sufficiently similar to each other are likely to have a similar function, but this simplifying intuition implies several caveats that need to be associated with the interpretation of the results: the annotation of the target sequence may be incorrect; an inappropriate threshold of similarity may have been chosen; very similar sequences may have different functions; and, due to the modular architecture of proteins, high similarity can be easily found in small regions of sequence, from which only limited information should be inferred. Because of these caveats, functional annotation is a complicated task that still requires human intelligence. Accordingly, the MGI database does not annotate sequences automatically, but assists the user in doing so by providing timely, well-organized information that can support skilled investigation and inference.

A second BLAST search is also carried out in order to detect the presence of *E. coli* genomic sequences (from the bacterium used to propagate the cDNA), and RNA molecules other than mRNA (messenger RNA) that can occur in cDNA libraries. The targets of this search are custom databases consisting of the *E. coli* genomic sequence, all known viruses, and all RNA molecules in GenBank that do not contain the string "mRNA" in their description.

Database support. The sequences, their meta-data, and the results of their analyses, as well as some tables that are used to administer the pipeline, are all stored in a Sybase relational database. The gatherer gets its input from a directory in the UNIX file system, whereas all subsequent operations get their inputs from, and deposit their outputs to, database tables. As noted earlier, the database is also used to support the pipeline by storing information, including status, about tasks that need to be done for each stage, as well as references to the data upon which they will operate. The relational schema used here is almost identical to that already described,[9] with added columns to hold the quality-score data, and GenBank accession numbers of published sequences. The GenBank accession numbers are entered *en masse*, after sequences have been made public by submission to GenBank, which has become the *de facto* official repository for all public nucleotide sequences in the United States. The ability to query sequences based on their cDNA library of origin was enabled by encoding the name of the library in the name of the sequence file, and having the user interface use the appropriate SQL (Structured Query Language) statement. Although less efficient than creating a database column for the cDNA library, it is a useful and flexible interim method of adding granularity. The cDNA libraries currently in the MGI database come from different plant tissues (e.g., leaf, stem, root) and different treatments (e.g., insect damage, phosphate starvation). We wish to ultimately import public data from sources such as GenBank into the MGI database, and to be able to formulate queries across data from different sources. In order to do this we will need a more sophisticated data model that incorporates a controlled vocabulary of tissues and treatments (see section on future work).

Distributed analysis server. In order to increase the throughput of this system, we have made simple modifications to the computationally intensive stages of the pipeline, so that they can function as clients of NCGR's distributed analysis server, allowing those pipeline stages to process multiple data elements concurrently on powerful parallel hardware. At present, the server only provides support for finding sequence homology using the BLAST algorithms,[11] and for removing vector sequence.

The distributed analysis server is a general facility that provides bioinformatics computation to a variety of clients. In addition to handling tasks associated with analysis pipelines (of which there are several), there are also interactive applications and Web-based clients, both internal and external. The system receives requests from clients and places them on a common queue, from which jobs are dispatched to be processed on a 40-processor domain of NCGR's Sun Enterprise 10000 computer, a large symmetric multiprocessor. The system can be reconfigured to conform to the size of the domain (i.e., the number of available processors), or can be run on other kinds of hardware platforms, such as a "farm" of networked workstations, or on a dedicated cluster, though these configurations have not been deeply explored. (In response to a reduction in the size of the original multiprocessor domain, another moderately sized multiprocessor machine was easily added to the configuration, so that the two multiprocessors formed a cluster. Aside from a somewhat increased administrative burden, no particular performance problems were noticed.) The jobs on the queue can be ordered according to any computable pair-wise ordering function, but the default behavior is to process jobs in first-in first-out (FIFO) order. There are tentative plans to extend the types of computations that are served by the system to include finding alignments of multiple sequences, clustering and assembly of contiguous overlapping segments of sequence, phylogenetic analysis, and other extensions.

CORBA

Middleware is the part of a software system that handles communications between segments of software such as a client and server, possibly running on separate computers. For the distributed analysis system, the communications between the server and its clients, and between the distributed elements that make up the server, are all supported with CORBA**. The Common Object Request Broker Architecture** is a specification for computer middleware support, and is defined through an open standard by the Object Management Group (OMG).[13] CORBA allows programs written in diverse computer languages to interact seamlessly, and this is one of the features that makes it attractive as a substrate for the distributed analysis system. This feature means that legacy server code written in C++ can be easily integrated with newer client software being written in the Java** language. The fact that CORBA is defined by a consortium of industry leaders tends to assure that its optimal use is not limited to some subset of computer systems or application domains. In addition, many CORBA implementations allow communications that take place between objects located in the same address space to be subsumed as interprocess

communications, which are much more efficient than network communications.

The server is designed to be simple, modular, flexible, and reconfigurable, in order to allow experiments with variations in implementation and deployment.

Design

The server system is composed of several distributed components, which interact with each other to accomplish the work of the server. There is the AnalysisServer object itself, a BasicQueue object, a LogManager object, and several ComputeServer objects, which do the actual computation on behalf of the server. All these objects are multithreaded, meaning that they may be performing multiple tasks concurrently (but in the case of any given ComputeServer object, all threads will be working on the same computation). In addition, when the system is busy, there will be numerous Request and Result objects, which represent the tasks submitted to the system by clients and the corresponding values being returned to clients. On a shared-memory multiprocessor, it makes sense to deploy all these components on the machine that is supporting the computation, for the sake of communication efficiency. However, it is only strictly necessary that the ComputeServer objects reside there. (In a cluster environment, for example, the AnalysisServer object and its queue might reside on a "master" node that has network connectivity outside the cluster, with each of the internal "servant" nodes supporting one or more ComputeServer objects.)

Reasoning from simple experiments that run identical similarity searches with different numbers of threads on a large multiprocessor, it is evident that the current NCBI multithreaded implementation of the BLAST algorithm has scaling limitations such that optimal performance is achieved with four 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, this scaling limitation is not particularly restricting. For example, instead of running two jobs sequentially, using eight processors for each, it makes more sense to run them concurrently, using four processors each. The time taken to process an individual request on four processors may (hypothetically) be greater than it would be with eight, but it is still faster overall to process two requests concurrently on four processors each, than to process them sequentially on eight processors. Consequently, the system spends less of its total resources to get both jobs done.

Given a 40-processor domain on a shared-memory machine, we currently deploy nine ComputeServer objects, each of which will run BLAST with four threads, accounting for a total of 36 processors, if all are busy. This leaves four processors available to run the server, queue, and other tasks.

The distributed analysis server is a general facility that provides bioinformatics computation to a variety of clients.

The BasicQueue object

A BasicQueue object is actually a subsystem that coordinates two internal queues. There is a request queue that handles requests for work to be done and a server queue that handles servers that are available to do work. Whenever both of these internal queues contain elements, the BasicQueue object removes the first element from each of the internal queues and dispatches the request element to the server element.

Each element on the server queue includes an associated parameter. When the BasicQueue object dispatches the request to the server, it includes this associated parameter. Thus, when a server is being enqueued on the server queue, this parameter can be used to associate the server with a ComputeServer object. Similarly, each element on the request queue includes an associated parameter that is used to hold a reference to the corresponding client. This client reference is also included in the dispatch to the server, whereby the server is able finally to perform a callback to the client, with the results.

Each of the internal queues can be maintained in any desired order, so long as that order can be defined in a comparator function that can determine the order of any two given elements. For example, an ordering function on the request queue might give higher priority to requests from interactive clients than it does to requests from "batch" clients like the analysis pipeline. And an ordering function on the

server queue might give priority to servers representing more powerful hardware. By default, both queues operate in FIFO order, which means that requests are processed in the order in which they are received, and are dispatched to the least recently used ComputeServer object.

This offers a simple way to organize the scheduling of tasks in the distributed analysis system. At initialization time, the server enqueues itself multiple times on the server queue, in association with each ComputeServer object that is to be represented on the hardware. Once initialization is complete, all incoming requests for work are simply passed on to the request queue. As the BasicQueue object finds work for the server to do, it will invoke the server via a callback mechanism, supplying a Request object for work, along with a ComputeServer object to use to perform that work, and the client that is to receive a call when the work is completed. The server then finds or spawns a worker thread, and the thread then invokes that ComputeServer object with the data from that Request object. The server resumes listening for more requests or dispatches, while the thread waits for the work to be done. When the thread receives results from the ComputeServer object, it makes a callback to the client, and, finally, re-enqueues the server on the server queue, in association with the same ComputeServer object. Thus, the behavior of the server is relatively simple, and scheduling is accomplished through the behavior of the BasicQueue object.

Additional request parameters

A Request object that is submitted to the distributed analysis server includes several control parameters. In addition to the data that define the work to be done, the request also identifies:

1. The desired mode of interaction.
   A. If the interaction is to be asynchronous, then the Request object must also contain a reference to the object that is to receive the eventual call with the completed result values.
   B. If the interaction is to be by e-mail, then the Request object must specify an e-mail address where the results are to be sent. When a Request object designating e-mail-mode interaction is received, the server creates a transient helper object containing the supplied e-mail address, and this transient object becomes the transaction client and will eventually receive the result values. When it does, it will format the results appropriately, and send them to the specified address.
   C. Finally, a Request object may specify synchronous interaction. This returns to the client a "future" object, which represents the result of the ongoing work. In this case, the returned future object also becomes the transaction client and will receive the results when they are eventually made available. Future objects are discussed in the next section.
2. The format of the result. Sequence similarity results are always returned as a data structure, parsed from the output of BLAST. Additionally, however, the client can control the format of a textual version of the output, which is also returned. This can be provided as (a) raw text, (b) HTML (HyperText Markup Language), or (c) as comma-delimited or (d) tab-delimited text, in which case the Request object may also specify which fields are to be included.

Futures

A "future" is an object that represents the result of a computation that is possibly not yet completed.[14] When a client submits a request to the distributed analysis server, the server immediately returns a result value. In the case of e-mail-mode or asynchronous-mode requests, this result can be a simple null value, because those modes both imply alternative channels for the communication of results from the server to the client. But in the case of synchronous-mode requests, the server can supply a future object, which represents the ongoing computation.

A future can be thought of as a reference to data that are not actually available at the moment. It is also useful to understand futures as a way to allow a server to communicate asynchronously with a synchronous client. A future can be stored into data structures, retrieved, passed as a parameter, returned as a result value, and so forth. As soon as a program attempts to extract the referenced data, the future object suspends until the data are actually available, at which point the data are returned.

From the server's perspective, each of the three modes of interaction with the client is handled relatively simply. In the case of an asynchronous-mode request, the server extracts the identity of the callback object from the Request object and enqueues that (as the client), along with the Request object, on the request queue. In the case of an e-mail-mode

request, the server creates the transient helper object, and enqueues that and the request on the request queue. And, in the case of a synchronous-mode request, the server creates a future object and enqueues that along with the request on the request queue (in this case, the server also returns that future object to the client).

From the worker thread's perspective, things are even simpler. All three modes of request interaction are handled in exactly the same way. By the time the server has handed off a dispatched request to one of its threads, the designated "client" is either (1) an object designated in the original request to be the recipient of an asynchronous callback, or (2) a transient e-mailer object waiting for values that it can format and send, or (3) a future object that was previously returned to the original client. All three object types are required to support the same callback interface, and so the thread is assured of being able to treat them identically. The thread simply creates a completed Result object and makes a callback to the designated client object, providing that Result object as a parameter. If the receiving object happens to be a future, the future object will now have its data, and if the client is waiting for those data, it can now proceed.

Garbage collection

Any system that continuously allocates memory dynamically must also be able to reclaim that memory somehow, if it is to avoid running out of space. This is called garbage collection. Some languages, like LISP and the Java language, provide built-in garbage collection subsystems, which enable them to reclaim chunks of allocated storage automatically by scanning through all the allocated storage. Any allocated storage not touched in such a scan can be assumed to have no references. Because such chunks cannot possibly be accessed by a program (written in one of those languages), they can safely be reused for other purposes. However, these systems typically reside in a single address space. Supporting garbage collection in systems that may potentially be distributed across multiple address spaces is significantly more challenging. Furthermore, CORBA is required to be mapped seamlessly onto programs written in diverse computer languages, including those such as C and C++ that do not have built-in garbage collection. Therefore, distributed systems written in CORBA do not have built-in garbage collection, and it is up to the distributed analysis server to reclaim objects in some appropriate way.

The server is not in a position to determine whether or not a client may retain a reference to some Request or Result object that has been allocated by the server. (Request objects may be allocated by the server, at the behest of a client, in order to allow convenient initialization and to promote efficient communication among the objects with which the server must deal.) A simple rule is added to the semantics of Result and Request objects in order to facilitate garbage collection: as soon as the data are extracted from a result, that result and its corresponding request are liable to garbage collection without further notice. This allows the client a reliable interface for extracting data from results, but also assures that the server can reclaim objects as soon as is reasonably possible. Since there may be a high volume of requests passing through the system, and since results can have significant size, this policy allows the server to keep its size as small as possible.

Pipeline client

In order for the analysis pipelines described earlier to be made clients of the distributed analysis system, it is necessary to make some minor modifications to the software classes that implement the stages of the pipelines. These software classes interact with the database to find the work that needs to be done for a given stage, to store the results of that work, and to set up work for any subsequent stages. The pipeline software is written using object-oriented software design, with a "base class" that defines the general behavior of a pipeline stage, which is then specialized by subclasses that define the particular behavior of specific pipeline stages.

The general behavior of a pipeline stage (as defined in the base class called TaskMaster) is to extract from the database a list of pipeline tasks (called "actions") that are pending for the given stage, suggested in Figure 2A. Then, for each of these actions in turn, the method ProcessPendingAction is called, which is an abstract method in the base class. This means that it is up to the specific implementation of any pipeline stage (i.e., in the definition of a subclass of TaskMaster) to define what it means to process a pending action. In the case of vector screening, for example, the pending action would be used to find the raw sequence that was imported by the previous Gatherer stage. This class structure is illustrated in Figure 2B. The raw sequence would be compared against the library of vector sequences to find contamination, and would have any trailing poor-quality region excised, and so forth as described previ-

ously, with the resulting purified sequence then being written back to the database in a different table by the TaskMaster object.

The modifications to this process are made both in the general (base class) and specific (subclass) aspects of the implementation, as suggested in Figures 3A and 3B. In the general case, instead of processing a task, saving the results, and moving on to the next task, every pipeline stage now has two phases of processing. First, after collecting the list of pending actions, a pipeline stage will "preprocess" the entire list. PreProcessPendingActions is defined in the base class as a "no-op," that is, as a "do nothing" operation. So, by default, preprocessing accomplishes nothing. However, in the specific implementation of each pipeline stage, the subclass can now determine whether the distributed analysis system is to be used. If so, then the preprocessing phase can be used to send off the entire list of pending tasks to the distributed analysis server, to be treated concurrently as a "batch." The requests all designate synchronous-mode interaction. For each request, the "future" result returned by the distributed analysis system (representing the ongoing work corresponding to that request) is collected into another list of "pending computations." The regular-processing (or "harvesting") phase in ProcessPendingAction methods can then associate the results extracted from each pending computation with the corresponding pending task, and can write the results to the database in the same fashion as before.

The "future" nature of the results returned by the distributed analysis system makes writing such a client fairly simple, if one can presume that the time cost of doing the computation is significantly higher than the cost of interacting with the database to store the results, and that the duration of the computation of each of the tasks is roughly similar. In the present case, these conditions are reasonable. The relevant database operations usually require milliseconds, and the sequence homology searches, using four threads on a symmetric multiprocessor, require from seconds up to a few minutes.

It may be that some particular result takes longer than many others to compute (either because it waits longer in the queue, or because it represents a more arduous computation), but this will generally not affect the throughput of the system, unless that result is one of the last elements in a batch, in which case many ComputeServer objects may fall idle while the client is waiting for the value of a last result to be computed. Normally, there will be other requests in the batch that are being handled by the distributed analysis system at the same time, and so while the client may be suspended for a while, waiting for one of its pending results to complete, there will probably be several other results that are completed in the meantime; and once the slow result is completed, the other completed results can be processed quickly. Because the preprocessing phase of the pipeline creates the list of pending results in the same order in which they were enqueued, and the queue of the distributed analysis system currently presents a "first come, first served" behavior, the harvesting or processing phase will generally be waiting for tasks that

Figure 2A Older design for operation of a pipeline stage. Each pending task is processed in sequential order.

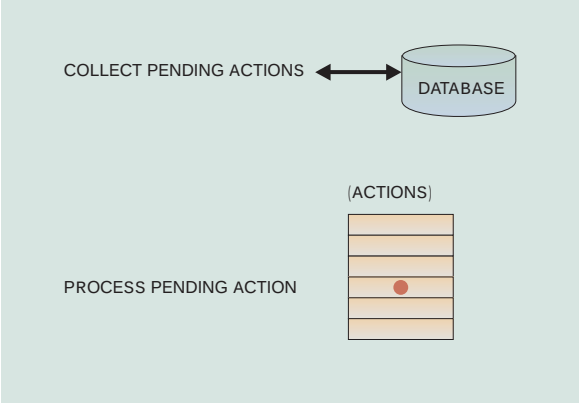


Figure 2B Class hierarchy for the older design of a pipeline stage. Each pending action is processed in sequential order.

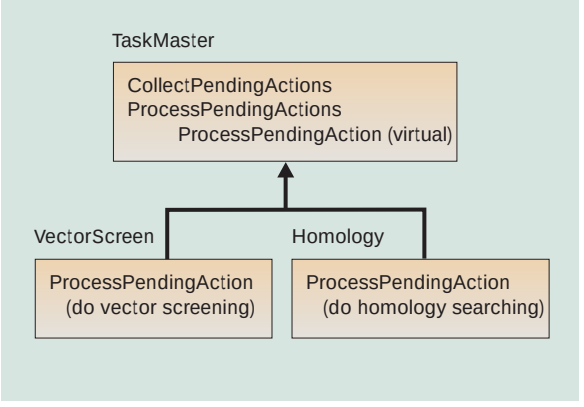
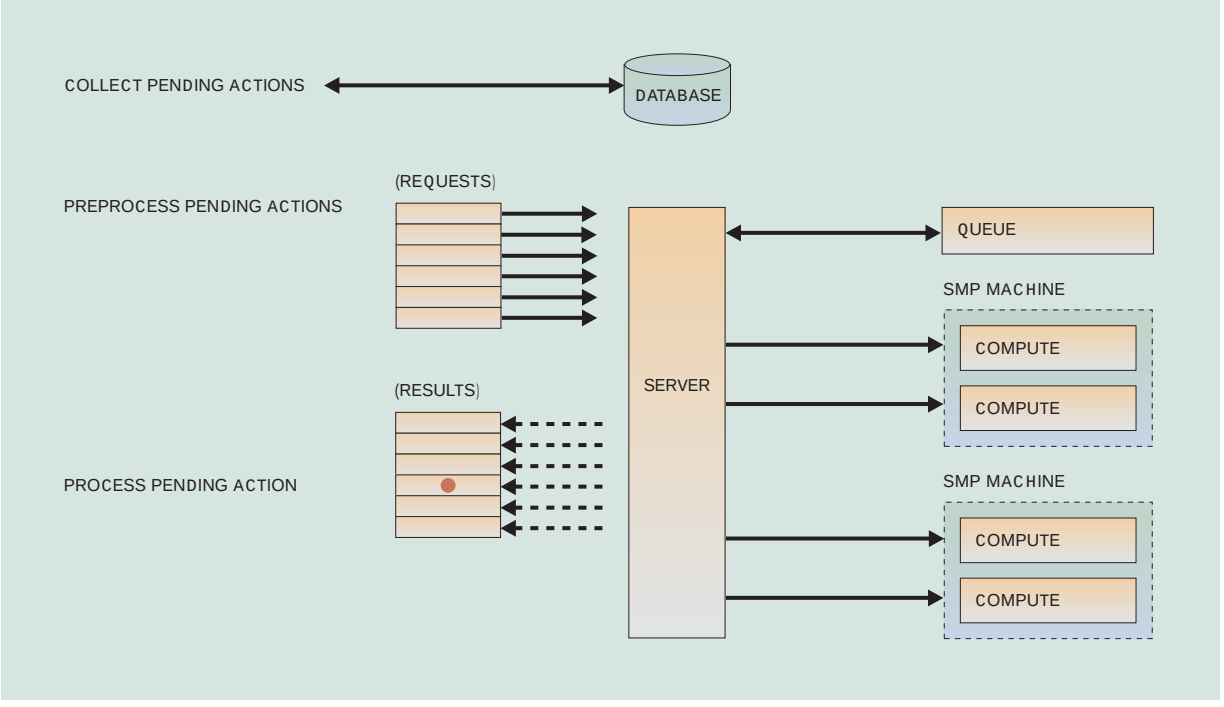


Figure 3A Revised operation of a pipeline stage. Tasks are now processed concurrently by the Distributed Analysis system, and only the results are gathered sequentially. The compute hardware is not required to be SMP machines, but unless the server queue is tuned to prioritize the available ComputeServer objects reasonably, these objects should all have similar resources.



are already being serviced. By making the batch size large relative to the number of compute servers, the throughput overhead incurred by the last few jobs in the batch (when many of the compute servers are likely to be idle) will be mitigated on average. The overhead mentioned earlier, in regard to the first elements in a batch, seems generally to apply only to the first batch.

Performance

Figure 4 shows some general throughput statistics. As described earlier, a set of requests that are submitted to the distributed analysis system during the preprocessing phase of a pipeline stage is considered to represent a batch. For all the requests in a batch, a common "start time" is recorded. When the result for each specific request is received, the "stop time" is recorded. Thus, when making a survey of the performance of the system, it is necessary to account for the fact that some requests will spend more time in the queue than others. This is accomplished by gathering the data back into the original batches and grouping actions for which requests were made at the same time. The latest stop time recorded for each batch will give the total time for processing all the requests of that batch, and this can be divided by the number of requests in the batch to yield the average time taken to process each request.

The scattered nature of the data is a consequence of operating in a production environment. Some fraction of the data will have been run when there was competition for resources between the ComputeServer objects and unrelated tasks. In addition, a significant fraction of the work has been done with a domain that was smaller than 40 processors, with a corresponding adjustment in the number of ComputeServers objects. Finally, the first few jobs in the first batch of a run will suffer the overhead of waiting for BLAST to page-in the reference libraries, as will be discussed shortly. But the conclusions from the data are fairly clear: overall, there is an average of 0.34 "jobs" (i.e., requests for similarity search us-

ing BLASTX against the "NR" reference library[15]) processed by the system, per second. This gives an average throughput of 20 homology searches per minute. If we limit the data to only those batches that contained 100 jobs, the figure is 23 searches per minute.

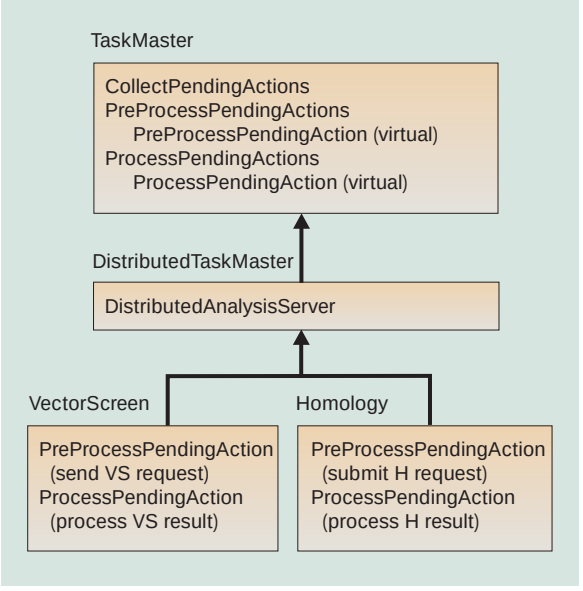
There tends to be significant overhead for some batches that contain only a few requests, because the operation of BLAST is significantly slower when run for the first time after a period of quiescence. This is apparently a result of the fact that the reference library (i.e., the library of sequence data against which the query sequence is being compared) is accessed using memory-mapped I/O. These libraries are fairly large (approximately 250 megabytes, for the library against which the data in Figure 4 were collected), and they are "loaded" by the BLAST algorithms via demand paging every time the BLAST programs are run. Once the first "generation" of jobs has passed through the queue (i.e., once each of the ComputeServer objects has been invoked), subsequent generations do not need to page-in the reference libraries again, and so they exhibit better performance. We are helped here, no doubt, by the amount of real memory that we have available on this large machine.

Figure 5 shows a simple analysis of the overhead of the communications in the distributed analysis system. To perform this test, a set of one or more concurrent client threads send repeated requests to the server, as rapidly as the server is able to return results. Each request specifies a BLASTX search against a small reference library (the library in fact contains the approximately 3000-nucleotide sequence representing the vector for the MGI pipeline). Thus, the entire overhead found in typical requests from the MGI analysis pipeline (in the BLASTX operation) is captured.

The experiment is run with four different configurations of the server, using one, two, three, or four ComputeServer objects. The first two ComputeServer objects are resident on the same machine as the server (and queue). The additional ComputeServer objects are added on a comparable SMP (symmetric multiprocessor) machine, where they can still expect to find four free processors each. Each line represents a client process running with a constant number of threads.

This simple experiment is intended to reveal a few basic properties of the current implementation. First,

Figure 3B Revised class hierarchy allowing concurrent processing of actions through the Distributed Analysis system. The MGI system can process actions concurrently while permitting legacy pipelines to continue to run serially.

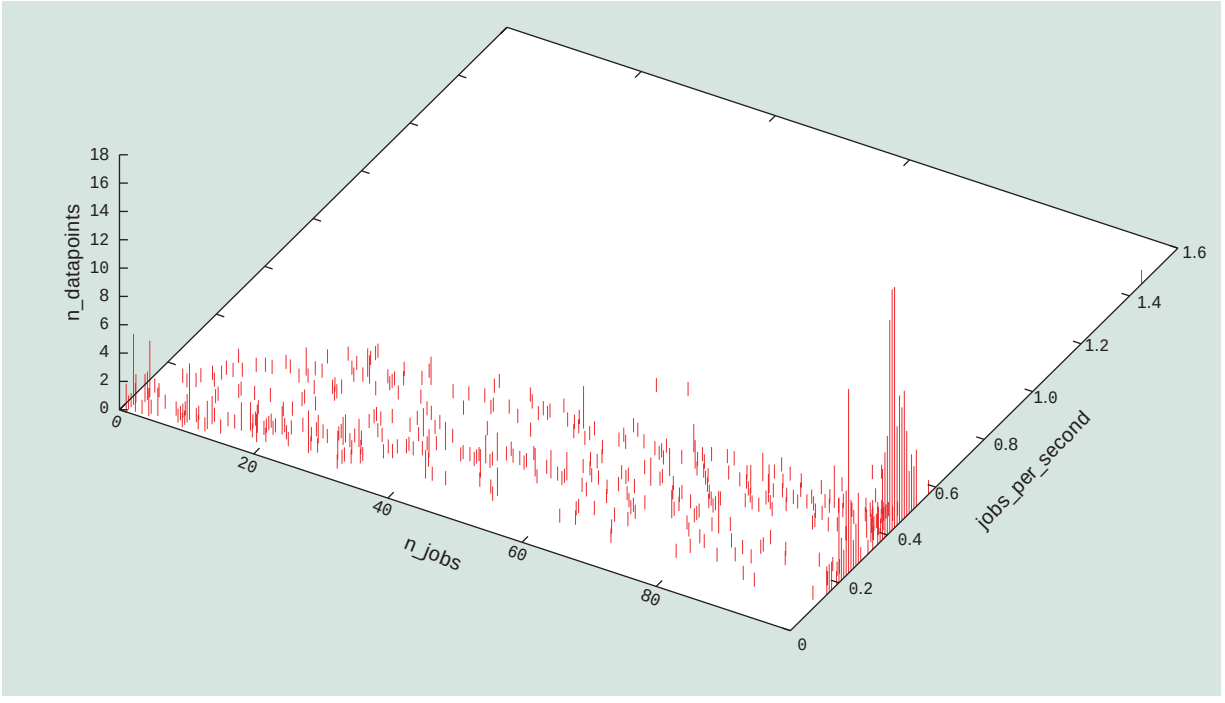


it is apparent that the limiting throughput for BLASTX requests through a single distributed analysis server lies somewhere between 500 and 600 requests per minute, when the ComputeServer objects are distributed across several machines. This makes it clear, in comparison with the production data from Figure 4, that the compute power of the hardware is currently the limiting factor. This suggests that compute power could be increased by a factor of approximately 25 before the communications through the distributed analysis system would become a bottleneck. It should be noted that the distributed analysis system maintains thorough logs of many internal events (through a separate server that supports logging), and that the Request and Result objects also record their state whenever it changes, so as to support object persistence. Performance would undoubtedly be improved by sacrificing these features.

User interface

The anticipated user community for this system has widely varying levels of computer aptitude and available technical support. For this reason we chose to

Figure 4 Performance over all batches of requests submitted by the BLASTX stage of the MGI pipeline. The *x* axis shows the number of requests in each batch. The *y* axis represents the average rate of throughput in a given batch. (Occasional faulty or extremely divergent data points have been eliminated for clarity.)



continue with the paradigm established by the prototype system, relying solely on Web browsers for the remote user interface. This also favors rapid prototyping and evaluation of new features, which can be quickly implemented using the Perl programming language. This is perceived as an advantage by the molecular genetics user community, which has an abundance of sequence analysis tools at its disposal. It also offers considerable flexibility at the back end, allowing developer choice in how HTML pages are generated. Currently, there are static HTML documents, CGI (Common Gateway Interface) Perl scripts, and PHP[16] scripts. While this adds to complexity and produces an increased maintenance burden, the results are seamless to the user, and the benefit is to simplify the prototyping of new features, which, if adopted, can be "hardened" later.

The system allows for two levels of data access, public and private. Because the researchers who are supplying the sequence information often do not know which sequences will prove to be interesting enough to merit further private scrutiny, we cause all the data produced by the pipeline to be placed into the database with an initial status of "private." These data can be viewed only by those collaborators who have password access to the database. Those sequences can then have their status changed to "public," at which point they become accessible to the general public. This is done through an interface that essentially mirrors the private one, but which has fewer features and lacks access to the private data. The public interface is located at www.ncgr.org/research/mgi. The shift from private to public is generally done at the same time that large sets of sequences are submitted to GenBank. This is done in a two-to-three month timeframe after initial submission of data to the system.

Users are welcomed by a static HTML welcome page, which has links to "help" pages, database statistics pages, the results of sequence redundancy checking, and an analysis page that offers several ways to access data.

Database statistics

The statistics page is generated in real time by a PHP script that queries a database table containing information on the number of sequences that are private and public, in several categories: number of sequences in the raw data table, number of sequences passing vector screen and quality control to enter the database proper, and in each of those categories, the number of sequences in each of a series of cDNA libraries. The table queried by the PHP script is itself generated nightly by a Perl program that performs a more complex series of queries on the database. Having a PHP script querying a simplified table greatly improves performance.

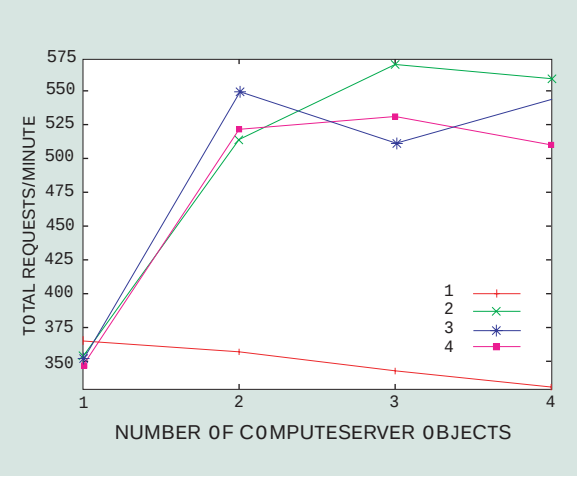
EST redundancy

Any collection of cDNA sequences generally represents a smaller number of mRNAs, a phenomenon called redundancy. To explore redundancy in a cDNA library, we employ a clustering method (see sidebar: DNA Sequence Clustering),[17–19] which groups EST sequences together based upon some measure of similarity, attempting to form sets of ESTs that are each representative of one mRNA. Clustering currently is not part of the pipeline, but is executed weekly and generates a static HMTL page indicating the number of ESTs and the number of nonredundant sequences in each cDNA library. Additionally, it has links to other static pages that show the ESTs that comprise each cluster. Clustering is important for two reasons. First, the system is used to support the design of cDNA microarrays, and because space on these arrays is limited, avoiding redundancy by using a single EST from each cluster is recommended. Second, individual sequence runs vary in length, so that it is desirable to create a "virtual consensus" sequence derived from looking at all of the constituent ESTs. Such a consensus is usually longer than the ESTs of which it is comprised and, as such, generally yields more information in analysis than the individual ESTs. Consensus sequences are therefore useful for further detailed study of sequences of interest. Incorporating clustering into the pipeline and using the consensus sequences as the input for analysis tasks are priorities for future development.

Data access

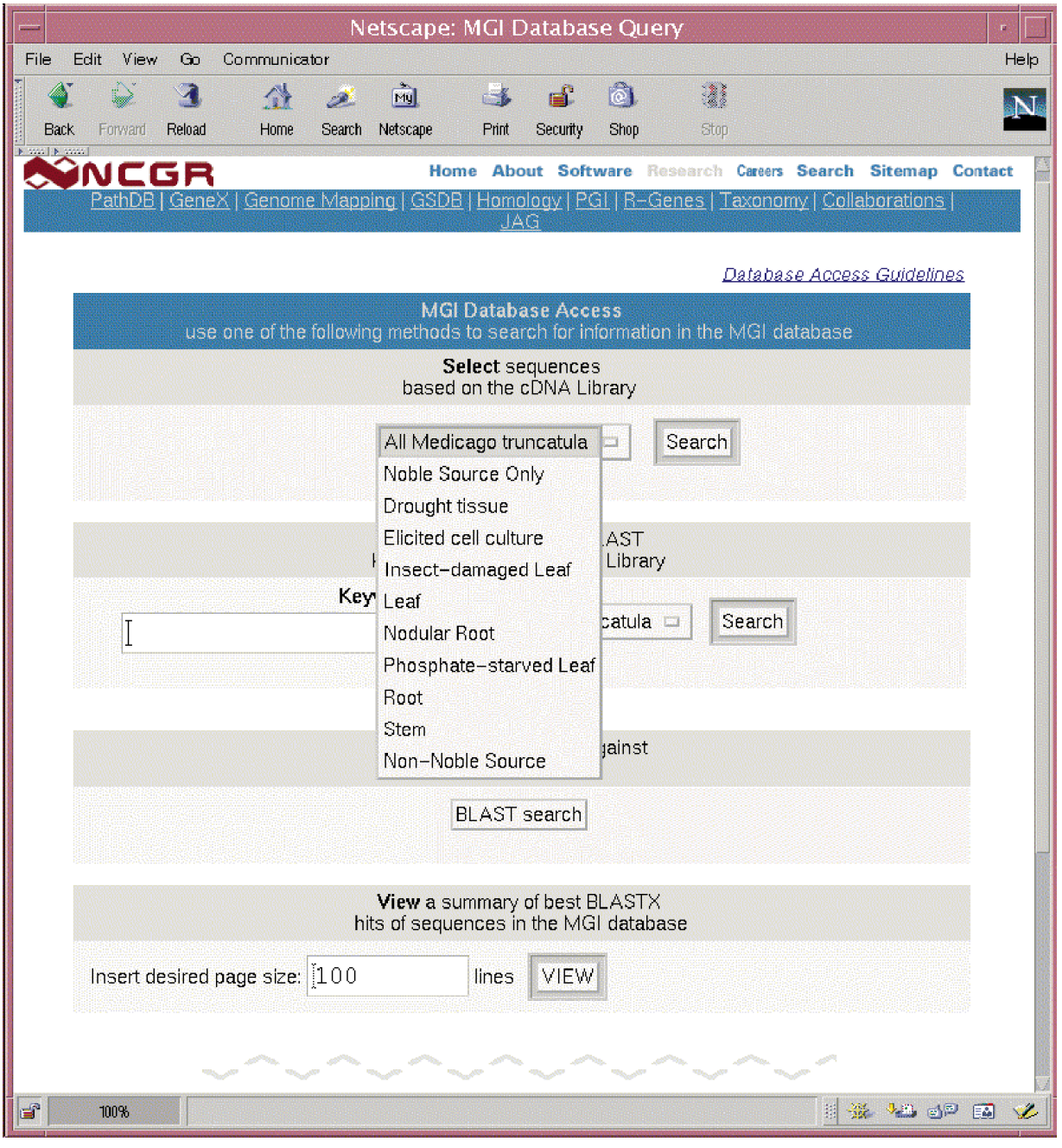
The welcome page has a link to a Perl CGI script that generates an HTML GUI (graphical user interface) with four choices of how to access data (Figure 6). Each of the subsequent choices connects to a Perl CGI script that queries the database appropriately via Sybperl or Perl DBI (database interface), returns data to the main program, and presents the data to the user in a familiar format in the browser. JavaScript** is employed where appropriate to manage the browser window environment, creating and formatting new windows as necessary.

Figure 5 Scalability in a cluster of SMP machines. The first two ComputeServer objects are resident on the same machine as the server and queue. The second two are on a different SMP. Each line represents experiments with some given number of client threads.



The first method offers a menu of the names of the cDNA libraries in the database. Upon selection, a scrolling list of the names of individual sequences is presented in the same browser window (Figure 7). Selecting one of these names causes a new window to appear, displaying the sequence of that entry, color-coded to indicate where the vector sequence and N-rich, poor quality sequence were identified (Figure 8). This window offers the user the option of saving the sequence to a local drive in either of two popular formats; and via the use of a menu, provides access to the results of the sequence analysis. There are three choices: vector screen results, BLASTX results vs the NCBI NR protein database, and BLASTN vs a custom set of libraries to check for various kinds of contaminating sequences. Clicking menu items leads to a hyperlinked display of the results of the analysis. Figure 9 shows the results of BLASTX for a particular sequence. This page is formatted in a way familiar to users with experience of BLAST, with a summary table of the best similarities in the NR da-

Figure 6 The main user interface page of the MGI system. Four methods of accessing sequences and their analysis results are offered, as described in the main text.



tabase, with hyperlinks to the sequence alignments on the same page, and to the appropriate sequence GenBank entries at NCBI.

The second method allows the user to again select a library (or all sequences in the database), and to perform a case-insensitive string search on the re-

DNA SEQUENCE CLUSTERING

The cDNA library that is prepared from mRNA in any given experiment typically represents 10 –10 independent mRNA molecules. Since an mRNA sequence for a given gene may be present many times in the same cell, the cDNA library may represent the same gene many times over. Because the clones are chosen randomly for sequencing, the sequence database may be expected to contain considerable redundancy. This is especially true of cDNA libraries made from tissues in which a single gene or small set of genes is expressed predominantly. Examples are green plant leaves, which contain an abundance of mRNA for the enzyme ribulose bisphosphate carboxylase, or the precursors of red blood cells, which express the globin genes at very high levels.

Estimation of the level of redundancy is done by a process called clustering, in which the goal is to group EST sequences such that those derived from the same gene are placed in the same group, and each group represents one gene. ESTs are usually grouped according to sequence identity, according to certain rules that govern the number of permissible mismatches and gaps between two sequences, and the minimum span of identity. An alternative method groups according to sequence composition by comparing the frequency of nucleotide "words" (of length N nucleotides where N is typically between and 10) in two sequences. There are several third-party programs available to do EST clustering, e.g., the TIGR assembler,[1] StackPack,[1,1] and PHRAP.

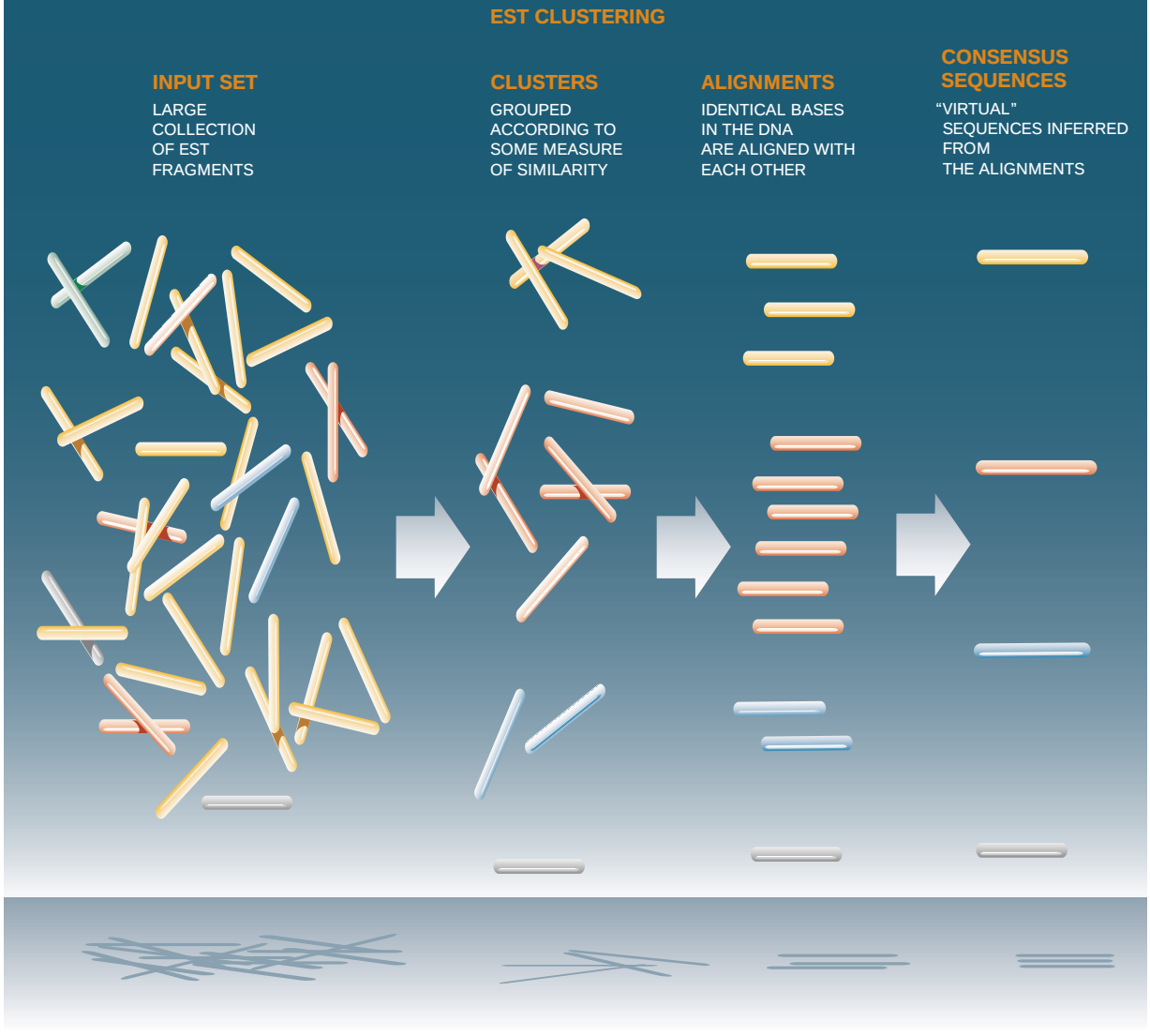
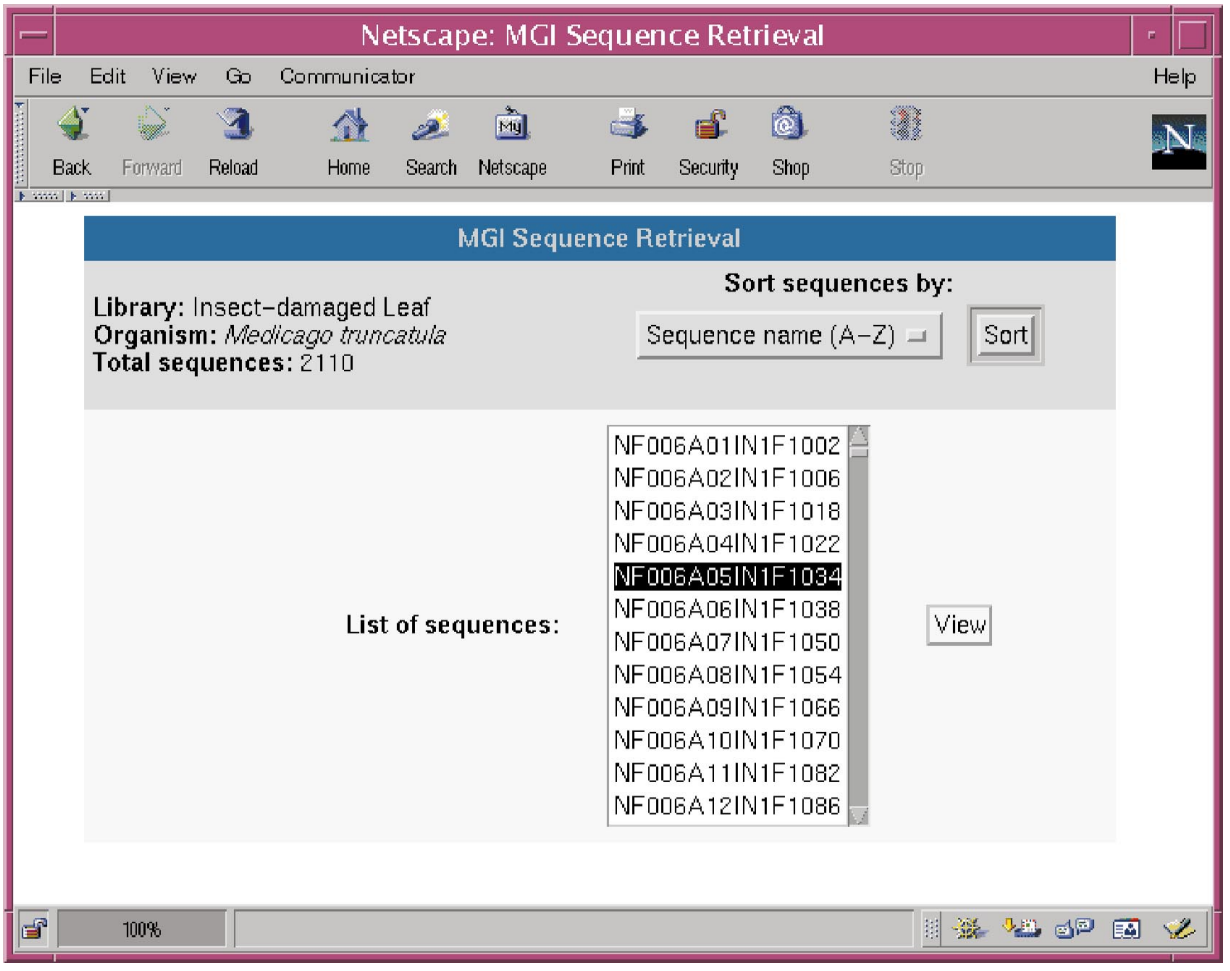


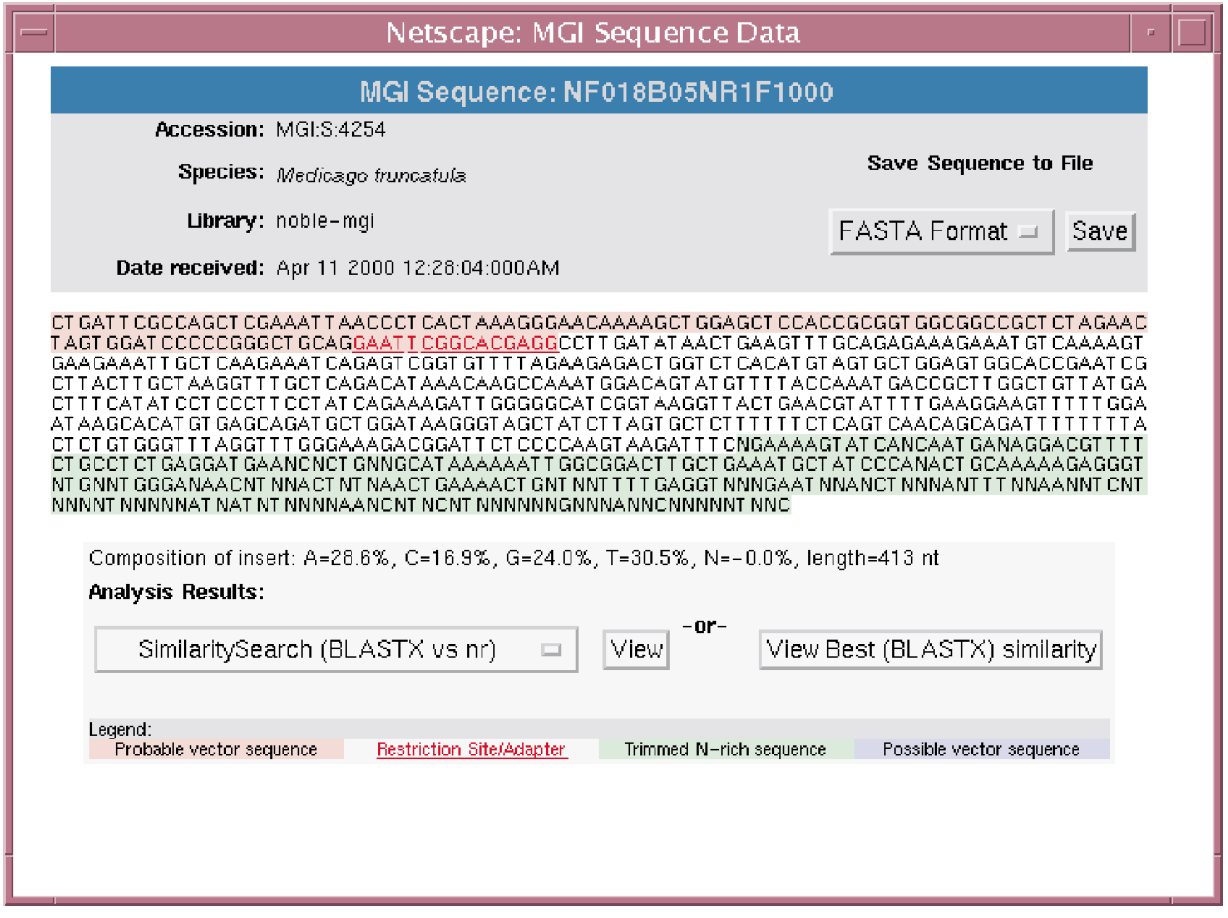
Figure 7 A scrolling list of all sequences in a given cDNA library can be sorted by date or by name. Selecting a sequence name and clicking the "view" button results in the new HTML window shown in Figure 8.



sults of the BLAST analyses of all sequences in the selected library. The string search is compared to the description line of all sequences that had a similarity above a certain statistical threshold. The description line is a short, human-readable summary of the sequence, usually including its function (where known) taken from its GenBank record. This method of exploring the database is the richest source of information for the user to search for genes with preconceived notions in mind. Submitting a string for search brings back either a message indicating that the string was not found, or a hyperlinked table showing the sequences for which BLAST hits contain the string, and links to the similar sequences at NCBI. The sequences with similarities are hyperlinks that invoke the window with the sequence display just described.

Just as similarity searching provides a rich source of information when comparing database sequences to the large public databases, it also can provide insight on the sequences contained in the MGI database itself. The third option for data exploration is a page where users can submit a sequence for BLAST search against all sequences in the database (Figure 10). This is valuable for identifying *Medicago* sequences similar to sequences from other species of particular interest, where the *Medicago* sequence is hitherto unknown. It is also useful for exploring the nature of gene families, using the TBLASTX feature that dynamically translates both the query sequence and the database sequences in all six reading frames, identifying similarities on the basis of protein, rather than nucleotide, similarity.

Figure 8 The entire raw sequence is displayed to the user, color-coded to indicate the vector sequence and N-rich, poor quality sequence. The sequence between these can be downloaded to the user's local disk by clicking the "save" button. By clicking the "view" button the user can see the results of sequence analysis.



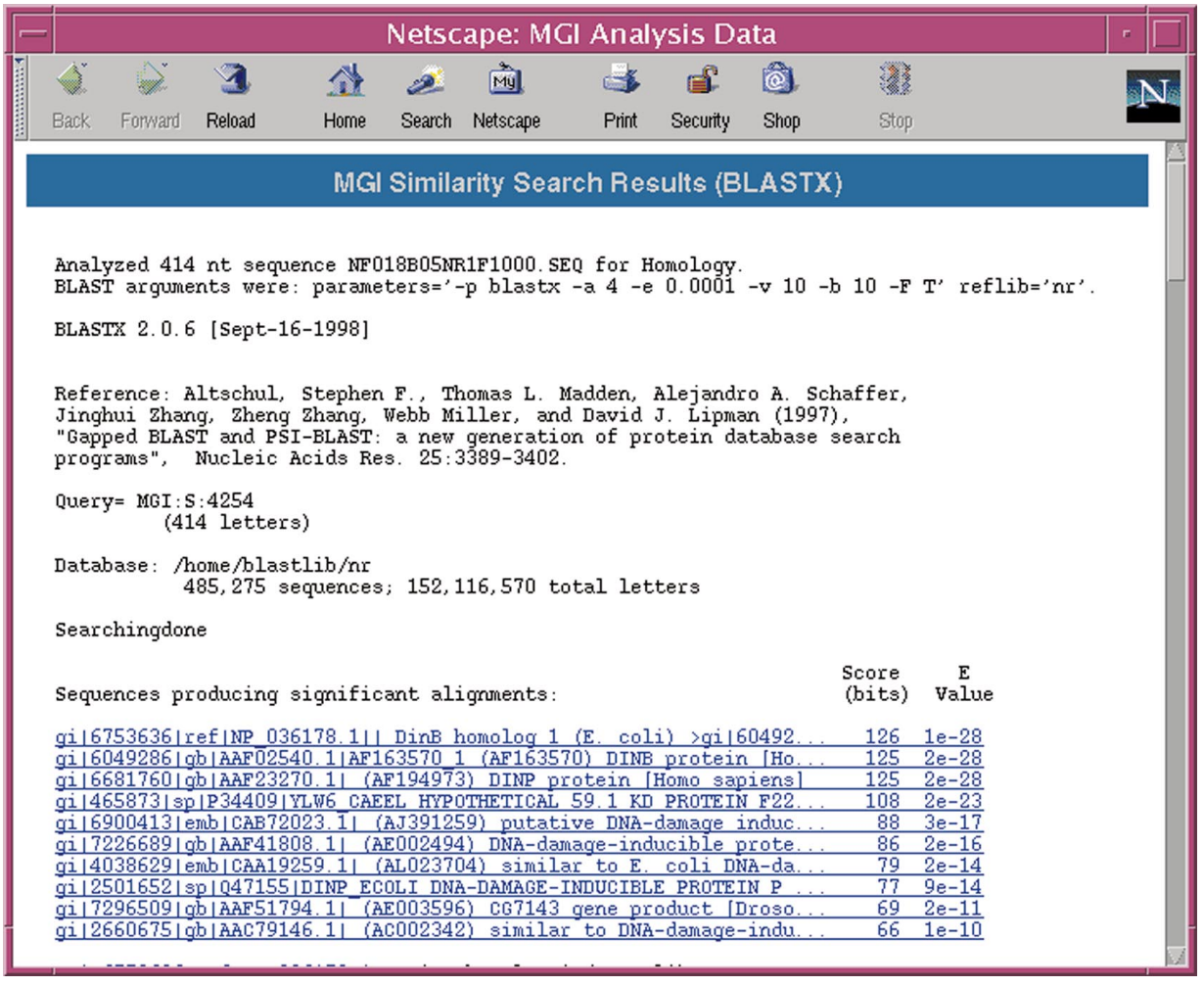
Whereas BLAST and string searches provide a specific approach to discovery, analogous to shopping with a list, the final method of exploration is closer to window-shopping. Clicking the button invokes a CGI Perl program that presents a list of every sequence in the database, along with its best similarity from the search against the NR database (Figure 11). The records are presented in pages, selected by the user. While it is laborious, feedback indicates that users identify genes of interest using this approach that would otherwise be overlooked.

Future work

We believe the current MGI architecture has reached the limits of its design, and rather than continue to add features in an *ad hoc* fashion, to support redundancy checking, for example, we are completely redesigning the system to retain the advantages of the current system with improvements that make it more modular, flexible, portable, and interactive. We are currently involved in developing the next generation of software for the new system. Some of the concerns we are addressing include:

1. The current database schema is EST-centric, meaning that individual ESTs are the units upon which the analysis operations work. Because a set of 40000 ESTs, for example, might only represent 10000 to 15000 genes, we do a lot of needless computation that could be eliminated by performing clustering first, and then doing the BLAST sim-

Figure 9 The results of a BLASTX search are shown. Each line in the table of hyperlinks at the bottom of the figure leads to a location further down the page where the user can examine the alignment between the query sequence and its "hit," and follow another link to the record of the hit at NCBI.



ilarity search on the consensus sequences from each cluster. This dependence on EST units is eliminated in a new schema.

2. The distinction between public and private data is useful, but is too simple for multiple groups of researchers who wish to share some data with other select groups, without making it globally public. A new technique is being developed to accommodate this and other requirements.
3. The distributed analysis system requires some sophistication to administer and is currently not portable. An alternative technology is being developed.
4. The software that provides an object interface to the database derives from a legacy library, which is complex and schema-specific. In order to accommodate advances in the schema, as well as in the semantics of interaction, new object interface tools are being introduced.
5. The only analysis operations that are currently supported are based on similarity searches, and there is a clear need to add others, for example a protein motif search, and secondary structure prediction methods. The process of adding analysis methods needs to be simplified.
6. There is a need for periodic automatic retrying

Figure 10 The interface where users can enter a sequence to be searched against the MGI database is shown. “Cut and paste” of text is supported, as is selection of a sequence from the user’s local drive. Results can be formatted in plain text or HTML and returned to the user in plain text or hyperlinked HTML.

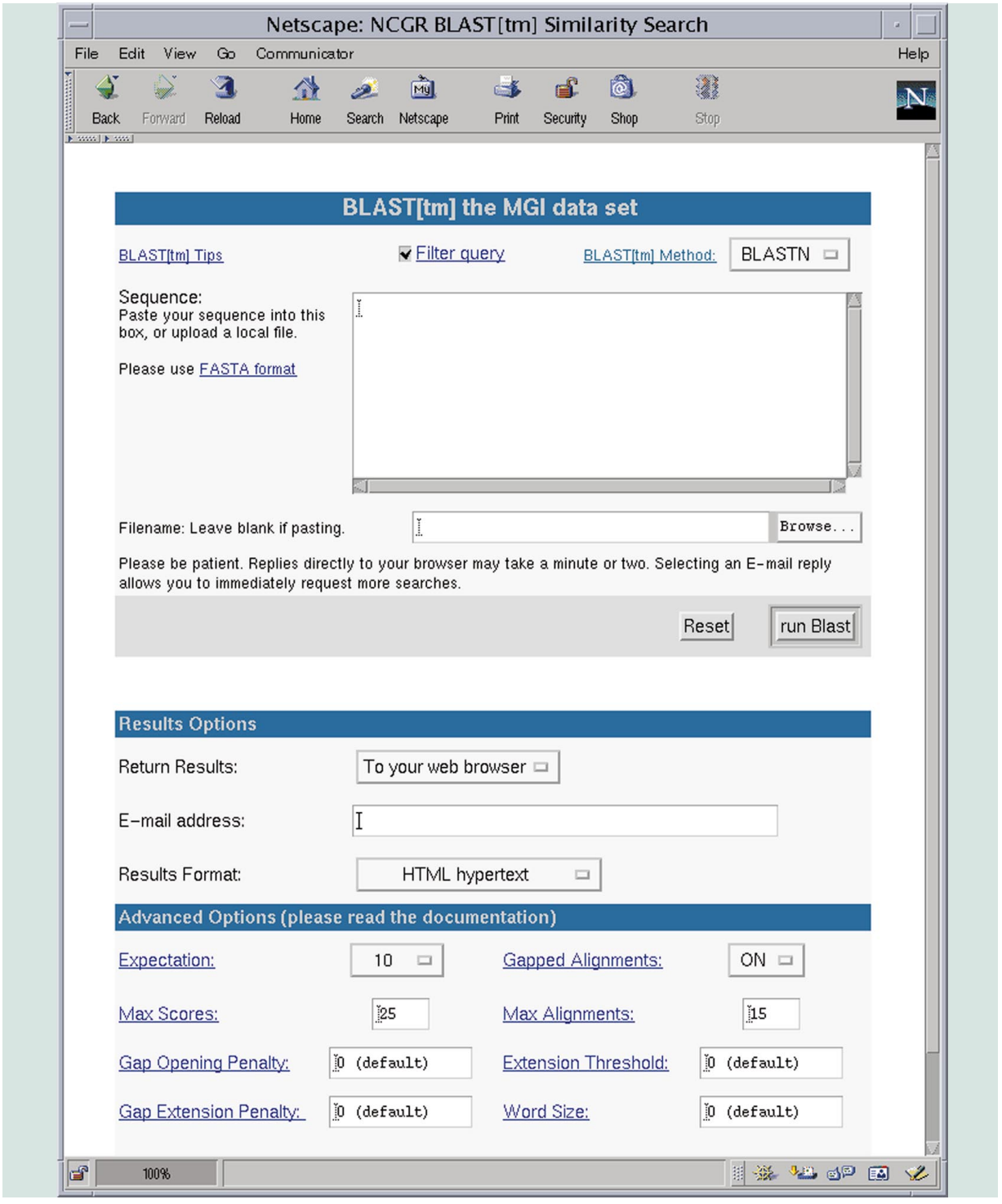
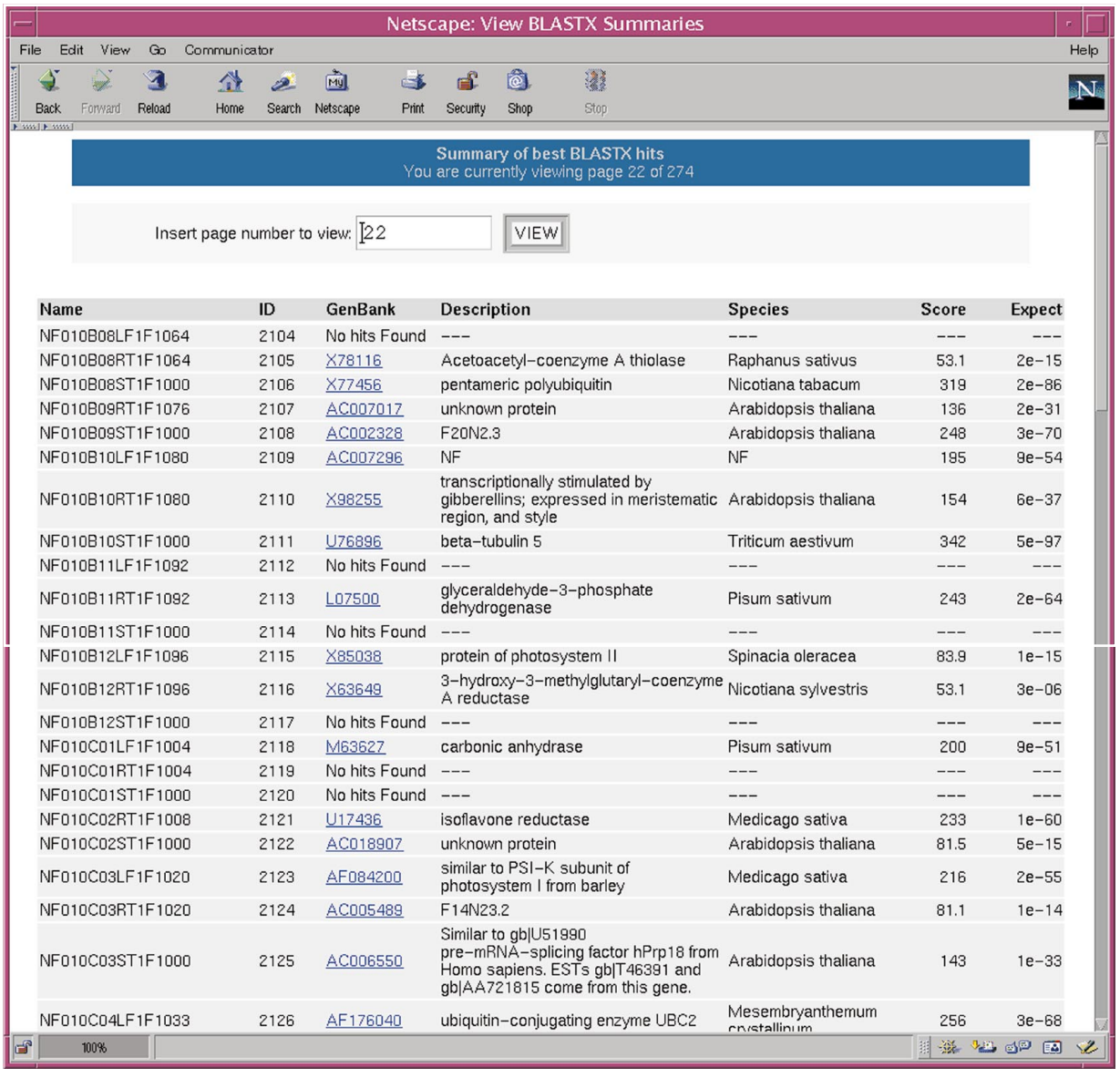


Figure 11 A page of best BLASTX similarities is shown. Important elements of the BLAST output such as the putative gene product function, species, and BLAST score are parsed from the output and formatted into columns in the table. The user can select from any of the available pages.



Name	ID	GenBank	Description	Species	Score	Expect
NF010B08LF1F1064	2104	No hits Found	---	---	---	---
NF010B08RT1F1064	2105	X78116	Acetoacetyl-coenzyme A thiolase	Raphanus sativus	53.1	2e-15
NF010B08ST1F1000	2106	X77456	pentameric polyubiquitin	Nicotiana tabacum	319	2e-86
NF010B09RT1F1076	2107	AC007017	unknown protein	Arabidopsis thaliana	136	2e-31
NF010B09ST1F1000	2108	AC002328	F20N2.3	Arabidopsis thaliana	248	3e-70
NF010B10LF1F1080	2109	AC007296	NF	NF	195	9e-54
NF010B10RT1F1080	2110	X98255	transcriptionally stimulated by gibberellins; expressed in meristematic region, and style	Arabidopsis thaliana	154	6e-37
NF010B10ST1F1000	2111	U76896	beta-tubulin 5	Triticum aestivum	342	5e-97
NF010B11LF1F1092	2112	No hits Found	---	---	---	---
NF010B11RT1F1092	2113	L07500	glyceraldehyde-3-phosphate dehydrogenase	Pisum sativum	243	2e-64
NF010B11ST1F1000	2114	No hits Found	---	---	---	---
NF010B12LF1F1096	2115	X85038	protein of photosystem II	Spinacia oleracea	83.9	1e-15
NF010B12RT1F1096	2116	X63649	3-hydroxy-3-methylglutaryl-coenzyme A reductase	Nicotiana sylvestris	53.1	3e-06
NF010B12ST1F1000	2117	No hits Found	---	---	---	---
NF010C01LF1F1004	2118	M63627	carbonic anhydrase	Pisum sativum	200	9e-51
NF010C01RT1F1004	2119	No hits Found	---	---	---	---
NF010C01ST1F1000	2120	No hits Found	---	---	---	---
NF010C02RT1F1008	2121	U17436	isoflavone reductase	Medicago sativa	233	1e-60
NF010C02ST1F1000	2122	AC018907	unknown protein	Arabidopsis thaliana	81.5	5e-15
NF010C03LF1F1020	2123	AF084200	similar to PSI-K subunit of photosystem I from barley	Medicago sativa	216	2e-55
NF010C03RT1F1020	2124	AC005489	F14N23.2	Arabidopsis thaliana	81.1	1e-14
NF010C03ST1F1000	2125	AC006550	Similar to gb|U51990 pre-mRNA-splicing factor hPrp18 from Homo sapiens. ESTs gb|T46391 and gb|AA721815 come from this gene.	Arabidopsis thaliana	143	1e-33
NF010C04LF1F1033	2126	AF176040	ubiquitin-conjugating enzyme UBC2	Mesembryanthemum crystallinum	256	3e-68

of the similarity search operation for those sequences that have no good database "hit," and for the users to introduce analysis operations themselves, in addition to those carried out by the pipeline.

7. The system only accommodates cDNA data. We intend to allow the storage and analysis of genomic DNA sequences in the same database as ESTs.

The revised data model is based on the concept of a generic analysis operation with input and output consisting of sets of data, which may contain zero, one, or more ESTs, genomic sequences, clusters, com-

puted features, and so forth. Thus, diverse analysis methods will be supported with the same relational schema, and the model will also allow for the flexible addition of new sequence analysis methods, without *a priori* knowledge of their input and output formats. We are also designing the new system with reusability and portability in mind, to simplify the process of creating new pipeline systems, and to allow remote deployment of the system without dependencies on NCGR computer infrastructure.

From the support perspective, once set up, the existing pipeline requires little maintenance, although some manual intervention is still required in the case of certain kinds of errors, which are brought to the administrator's attention by automated e-mail messages. For example, actions that were being processed at the time of a crash will have a status of "processing" or "error," and these must be reset to have a status of "pending." In addition, because the pipeline tasks are set up ahead of time by the preceding tasks, the analysis stages themselves have to be modified in order to add new types of analyses to the pipeline. In addition, setting up a new pipeline requires a fair amount of administrative work, mostly associated with customizing the user interface.

Conclusion

We have built a DNA sequence storage and analysis system for expressed sequence tag (EST) data that removes the bioinformatics responsibility from the DNA sequencing laboratory. Building on an earlier system with a simpler architecture, we have added new features of importance to the research community, along with an interface to a distributed analysis system that increases throughput by running tasks concurrently on multiprocessor machines or workstation clusters. The system offers the users well-organized views of their data and the results of analyses run on that data, and supports the discovery of biological knowledge embodied in the sequences by assisting in the functional annotation of genes. We are in the process of developing a more sophisticated system, based on experience we have gained by working with this system and its predecessor.

Acknowledgments

This project was supported by the Samuel Roberts Noble Foundation. We gratefully acknowledge Peter Hraber, Bruno Sobral, and the Phytophthora Consortium, whose database and analysis system, the Phytophthora Genome Initiative (PGI), served as a foundation for this work.

**Trademark or registered trademark of The Open Group, Sybase, Inc., the Object Management Group, or Sun Microsystems, Inc.

Cited references and notes

1. D. R. Walker and E. V. Koonin, "SEALS: A System for Easy Analysis of Lots of Sequences," *Intelligent Systems for Molecular Biology* **5**, 333–339 (1997).
2. See http://www.ncbi.nlm.nih.gov/Walker/SEALS/.
3. See http://stein.cshl.org/software/boulder/.
4. M. D. Adams, J. M. Kelley, J. D. Gocayne, M. Dubnick, M. H. Polymeropoulos, H. Xiao, C. R. Merril, A. Wu, B. Olde, R. F. Moreno, A. R. Kerlavage, W. R. McCombie, and J. C. Ventner, "Complementary DNA Sequencing: Expressed Sequence Tags and Human Genome Project," *Science* **252**, 1651–1656 (1991).
5. B. Ewing and P. Green, "Base-Calling of Automated Sequencer Traces Using PHRED: II. Error Probabilities," *Genome Research* **8**, No. 3, 186–194 (1998).
6. D. Gordon, C. Abajian, and P. Green, "Consed: A Graphical Tool for Sequence Finishing," *Genome Research* **8**, No. 3, 195–202 (1998).
7. See http://www.phrap.org.
8. See http://www.noble.org/medicago/ProgramLaunch/medicago_summary.htm.
9. M. Waugh, P. Hraber, J. Weller, Y. Wu, G. Chen, J. Inman, D. Kiphart, and B. Sobral, "The Phytophthora Genome Initiative Database: Informatics and Analysis for Distributed Pathogenomic Research," *Nucleic Acids Research* **28**, 87–90 (2000).
10. GenBank is the National Institutes of Health (NIH) genetic sequence database. It is the *de facto* official public repository of the United States for all of the nucleic acid sequences and is administered by the National Center for Biotechnology Information (NCBI), a branch of the National Library of Medicine.
11. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs," *Nucleic Acids Research* **25**, 3389–3402 (1997).
12. See http://www.ncbi.nlm.nih.gov:80/entrez/query.fcgi?db=Protein.
13. *The Common Object Request Broker: Architecture and Specification*, Revision 2.1, The Object Management Group, Inc., 492 Old Connecticut Path, Framingham, MA (1997).
14. R. H. Halstead, "MULTILISP: A Language for Concurrent Symbolic Computation," *ACM Transactions on Programming Languages and Systems* **7**, No. 4, 501–538 (1985).
15. The "NR" library is a nonredundant database of all known protein sequences. It is derived in part from GenBank and, like GenBank, is maintained by NCBI.
16. PHP is a server-side, cross-platform, HTML-embedded scripting language with database connectivity capabilities. Its use simplifies the creation of dynamic Web page content.
17. G. G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage, "TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects," *Genome Science and Technology* **1**, 9–19 (1995).
18. J. Burke, D. Davison, and W. Hide, "d2_cluster: A Validated

Method for Clustering EST and Full-Length cDNA Sequences," *Genome Research* **9**, No. 11, 1135–1142 (1999).
19. See http://www.sanbi.ac.za.

Accepted for publication January 18, 2001.

Jeff T. Inman *National Center for Genome Resources, 2935 Rodeo Park Drive East, Santa Fe, New Mexico 87505 (electronic mail: jti@ncgr.org).* Mr. Inman is a software developer at the National Center for Genome Resources in Santa Fe, New Mexico, where he pursues an interest in distributed systems. He received a B.S. degree in computer science from Boston University in 1986. He worked at Symbolics, Inc., in Cambridge, Massachusetts, and then at the Massachusetts Institute of Technology's Artificial Intelligence Laboratory, where he did research on fine-grained parallelism with the Message-Passing Semantics group. Prior to his work at NCGR, Mr. Inman spent time at the Santa Fe Institute, developing software that was used to study the inverse protein folding problem using genetic algorithms.

H. Raul Flores *Netvoice Technologies Corporation, 3201 West Royal Lane, Suite 160, Irving, Texas 75063 (electronic mail: rflores@netvoice.org).* Mr. Flores is currently working for Netvoice Technologies as a manager in the Web Development group where he is developing e-commerce applications using Java servlets, PHP, and SQL. Mr. Flores graduated from Texas A&M University with an undergraduate degree in geology. He received a master's degree in software engineering from Texas Christian University in 1993.

Gregory D. May *Samuel Roberts Noble Foundation, 2510 Sam Noble Parkway, Ardmore, Oklahoma 73402 (electronic mail: gdmay@noble.org).* Dr. May is an associate scientist in the Plant Biology Division at the Samuel Roberts Noble Foundation. His training is in plant physiology, molecular biology, and mechanisms of tissue-specific gene expression. Dr. May graduated with a B.S. degree in biology from Southeast Missouri State University and received his Ph.D. degree in plant physiology from the Department of Biochemistry and Biophysics at Texas A&M University. He did postdoctoral work at the Institute of Biosciences and Technology (IBT), Houston, Texas, where he studied plant tissue-specific gene expression. Following his postdoctoral studies, Dr. May was granted a Research Assistant Professor position at the IBT. In 1995, Dr. May joined the Boyce Thompson Institute for Plant Research at Cornell University as an assistant scientist. In 1997, he was granted adjunct assistant professor status in the Section of Plant Biology at Cornell University. Dr. May joined the Noble Foundation in May 1999 to establish the *Medicago* genomics program. His current research interests focus on DNA repair and gene targeting mechanisms in plants.

Jennifer W. Weller *Virginia Bioinformatics Institute. 1750 Kraft Drive, Suite 1400, Virginia Polytechnic Institute, Blacksburg, Virginia 24136 (electronic mail: jwweller@vt.edu).* Dr. Weller is a research assistant professor at the Virginia Bioinformatics Institute at Virginia Tech and was formerly NCGR's program leader for gene expression. She earned a Ph.D. degree in biochemistry from the University of Montana, Missoula. Her current research includes development of an EST and genomic DNA analysis pipeline and a "wet lab" investigation of the genes expressed by *Arabidopsis thaliana* in root development and during parasitic attack.

Callum J. Bell *National Center for Genome Resources, 2935 Rodeo Park Drive East, Santa Fe, New Mexico 87505 (electronic mail: cjb@ncgr.org).* Dr. Bell is a senior research scientist at the National Center for Genome Resources in Santa Fe, New Mexico. His training is in genetics, molecular biology, and genomics, with a long-standing interest in bioinformatics solutions to problems in genomics. Dr. Bell graduated with a B.S. degree in biological sciences from Edinburgh University and obtained his Ph.D. degree in the genetics of gravity responses in the model plant, *Arabidopsis thaliana*, from the same institution in 1988. He did postdoctoral work at the National Institute for Basic Biology in Okazaki, Japan, and at the University of Pennsylvania biology department. In his postdoctoral studies he worked in a group generating a physical map of *Arabidopsis*, and he pioneered the use of microsatellites as genetic markers in that organism. His work in genomics continued with an appointment at the Children's Hospital of Philadelphia where he led the physical mapping of human chromosome 22, and at Sequana Therapeutics, where he managed a multidisciplinary project team attempting to identify human susceptibility genes for Type 2 diabetes and obesity. Dr. Bell joined NCGR in September 1999. His main interest is bioinformatics systems that enable knowledge discovery in biological sequence databases.