EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE LAUSANNE
POLITECNICO FEDERALE DI LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY LAUSANNE

Computer Science Department
Operating Systems Laboratory

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Blast server on a cluster of Dec Alpha workstations at EPFL
Reference Manual

Alexandros Pappas

December 22, 1994

Abstract

A server for blast requests was implemented on a cluster of Dec Alpha workstations. We will discuss the parallelization of the Blast (Basic Local Alignment Search Tool) programs with the PVM library, the load balancing with RPC Mechanisms, and the network and mail server using Unix sockets and mail.

Contents

1 Introduction

This manual is intended for people who want to maintain the implemented blast server, and for anybody who wants to install a blast server on a distributed environment. The reader should know what the blast programs are and how they are called, should have an idea about the PVM library and the RPC mechanisms, and should have given a look to E. Masson's Diploma Thesis "Portage d'une application de biologie moléculaire d'un environnement multiprocesseurs vers un réseau de stations" [Masson94], and to the report of C. Javet and X. Défago "Distribution de la charge pour BLAST" [Jav Def 94]. The reader might also want to look at the page dedicated to this project on the WWW, at `http://diwww.epfl.ch/~blastdat`.

2 Blast accounts

There are two accounts for the project blast. One with username `blast` which is used for program development, and the other with username `blastdat` which is used only to store the databases. There are accounts as well on the didecs (didec1 - 30), as well on the disuns (disun1 - 50). Everything is done on the didecs except the mail server.
The didecs run for the moment OSF/1 V2.1 and the PVM library version 3.3.4 is installed in `~pvm3`. Of the blast programs, we have installed the serial versions 1.3 and 1.4, and our parallelization of version 1.4. Version 1.4 was released on Oct 94.

3 The serial blast programs

3.1 Description

The blast programs compare a given sequence (of amino acids or nucleotides) with all the sequences of a database using the statistical methods of Karlin and Altschul [Kar Al 90].
There are six blast applications: blastp, blastn, tblastn, blastx, tblastx and blast3 which were implemented at NCBI[1]:

- blastp: compares an amino acid query sequence against a protein sequence database.
- blastn: compares a nucleotide query sequence against a nucleotide sequence database.
- blastx: compares the six-frame conceptual translation products of a nucleotide query sequence against a protein sequence database.
- tblastn: compares a protein query sequence against a nucleotide sequence database dynamically translated in all six reading frames.
- tblastx: compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database.
- blast3: compares an amino acid query sequence against a protein sequence database and identifies statistically significant three-way sequence alignments in which the component pairwise alignments are statistically insignificant.

What is a six-frame translation? For our purpose it is enough, if we say, that it is the mapping of nucleotide sequences to amino acid sequences in 6 possible ways.
Blast3 exists in version 1.3, but not in 1.4, so we did not parallelize blast3.

[1] National Center for Biotechnology Information, USA

3.2 Installing and running

The blast programs are in the public domain, and can be obtained by ftp from ncbi.nlm.nih.gov/blast. Currently there are two versions installed at our site:

- version 1.3 under `~/blast1.3` and
- version 1.4 under `~/blast1.4`.

A priori, version 1.3 is not used any more and can be deleted.
To compile, you just have to follow the instructions in the `INSTALL` files in the directories `~/blast1.3/blast` and `~/blast1.4/blastapp` respectively.
The blast programs use the following environment variables, which when not defined, the values are taken from the equally named constants defined in `blastapp.h`.

- BLASTDB: directory of databases.
- BLASTMAT: directory of matrices.
- BLASTFILTER: directory of filters.
- BLASTCDI: directory of genetic codes.

Be sure to change these environment variables, if the installation does not correspond to the default values defined in `blastapp.h`.
To run a blast program, you have to specify at least the database to use and the query sequence to search for. For example:
`% blastp swiss ~/queries/protpart1.seq`
searches the query contained in the file `~/queries/protpart1.seq` in the database `swiss`, which is located in the directory given by BLASTDB.

4 The parallel blast programs

4.1 Parallelization

All blast serial programs have a very similar structure as shown bellow:

```
parse arguments;
while (chunks left) {
  get next chunk;
  compare given sequence with sequences of chunk;
  add results to the total results;
}
print the results;
```

Since we have many processors, we can search on many chunks at the same time. We implemented a master-slave configuration. The master starts its slaves and sends them the chunk numbers to work on. When a slave has finished with a chunk, it requests the next chunk from the master. At the end, every slave sends its results to the master, who adds them together.

The slaves are almost the same as the original program, only the way they get the next chunk changes, and instead of printing the results, they send them to the master. The master is like the original program, except that it does no searching at all, but delegates the searching to the slaves. In addition, the master distributes the chunks.
The pseudoprograms for the master and the slave are as follows:

<table>
<tr><th>master</th><th>slave</th></tr>
<tr><td><pre>send arguments;
parse arguments;
while (chunks left) {
  send next chunk;
}

while (not all slaves responded) {
  get results from slave;
  add results to the total results;
}
print the results;</pre></td><td><pre>get arguments;
parse arguments;
while (get next chunk) {
  compare given sequence with sequences of chunk;
  add results to the total results;
}
send results to master;</pre></td></tr>
</table>

From the above figures, it can be seen, that it is quite easy to modify the original programs, to get the master and the slave. For the master, the original blast main program, must be cut in two, one for parsing the arguments, and one for printing the results. The searching must be left out (procedure RunWild) and the distribution of chunk numbers and adding of the results must be added. For the slave, one just has to replace the procedure for getting the next chunk (TaskNext) with one, which will ask the master for the next chunk (net_next_seq), and the printing of the results must be left out. As a refinement, whenever a slave exits with an error, it sends a message to the master that it failed. The original programs call the function `exit()` when an error occurs. A function which is always called on a process exit can be installed with the atexit() system call. So we installed a function that sends a PVM message to the master before the slave exits.
In version 1.4 of blast programs, the main programs are so similar, that the transformation from the serial to the parallel version is the same for all of them and is implemented with a single Perl [2] [Swarz 93] script. (This was not the case for version 1.3, where one Perl script per blast program was needed). The script looks for some characteristic statements in the original blast programs, and inserts mainly some preprocessor statements.
Figure 1 shows for blastp how the master and the slave is generated. The `blastp.c` program is modified with the Perl script `net_patch.pe` to `net_blastp.c`. With conditional compilation statements, the master version `net_master_blastp.o` and the slave version `net_slave_blastp.o` are obtained. To these are linked two other modules, `resultsp.o` which contains functions mainly for mapping and unmapping of results, and `m_blastp.o` and `s_blastp.o` which contain the PVM communications.

4.2 Communication with PVM

The master and the slave communicate through PVM. The master spawns the slaves and sends them the message SEND_WORK which contains the command line arguments, and the first chunk number to treat. When a slave has finished with a chunk it requests the next chunk with SLAVE_REQUEST_WORK, to which the master answers ADD_SLAVE_WORK and gives the next chunk number. When all chunks are treated, the slaves send their results to the master with SLAVE_RESPONSE. If a slave fails, it sends a SLAVE_BFATAL to the master. The master kills the other slaves and sends an END_REQUEST to the deamon and a GET_RESULT to the client.

[2] Practical Extraction and Report Language

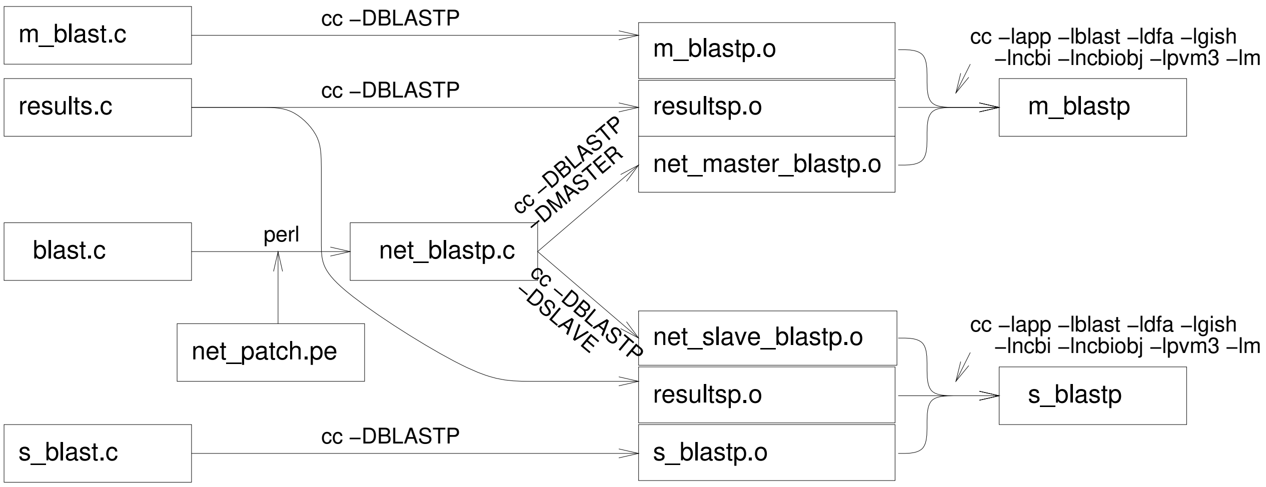

Figure 1: Generation of master and slave from the original programs

For reasons we will explain later, we introduced a deamon and a client program. The client program sends a CLIENT_REQUEST with the command line arguments to the deamon. If the deamon can not accept the request, it replies with a REJECT_REQUEST, otherwise it spawns the master and sends it the command line arguments with MASTER_WORK. When the master finishes it sends an END_REQUEST to the deamon, and a GET_RESULT to the client which means, that the client can now access the file, into which the results were written. The involved PVM messages for blastp are displayed in figure 2, but they are the same for all blast programs.

4.3 Deamon and client program

Because we have 30 stations, and normally 8 stations are used per request, we want to be able to treat more than one request at the same time. Also, we have to determine the load of the machines and run the requests on the least loaded machines. For that, we wrote a deamon (`demon_blast`). `demon_blast` waites for requests. When a request arrives, and if enough not very loaded machines are available, the request together with the machines to use are passed to the master. If not enough machines are available, then if the request is a batch job, it will wait in a queue, until it can be serviced, otherwise it will be rejected. The load determination of the machines is very well explained in [Jav Def 94]. There is a deamon (`monitor_svc`) running on every machine. `demon_blast` asks the load of each machine through remote procedure calls (RPC) to `monitor_svc`. `demon_blast` looks from time to time, if `pvmd` (the pvm deamon) has stopped running on some hosts, and if so, delegates its child `demon_child` to restart pvmd where appropriate. To do this, a list of all the hosts must be given to `demon_blast` in a file. `demon_child` runs all the time and is assigned to do this job, because it is very time consuming. The client program replaces the original blast programs. It is named as the original blast program with a `.pvm` suffix (ex.: `blastp.pvm` and takes the same command line arguments. They all are links to the executable `blast.pvm`. The command line argument `-batch` was added, which indicates that this is a batch job. This argument, if specified, must be the third, i.e. right after the query sequence. The client has to search the tid (task id in PVM) of `demon_blast`. The problem is, that we can get only a list of tasks wich were started through PVM, but `demon_blast` is not necessarily started from PVM. So, the client searches in the list for the child of `demon_blast, demon_child,` and through the child's parent tid, it gets the tid of `demon_blast`.

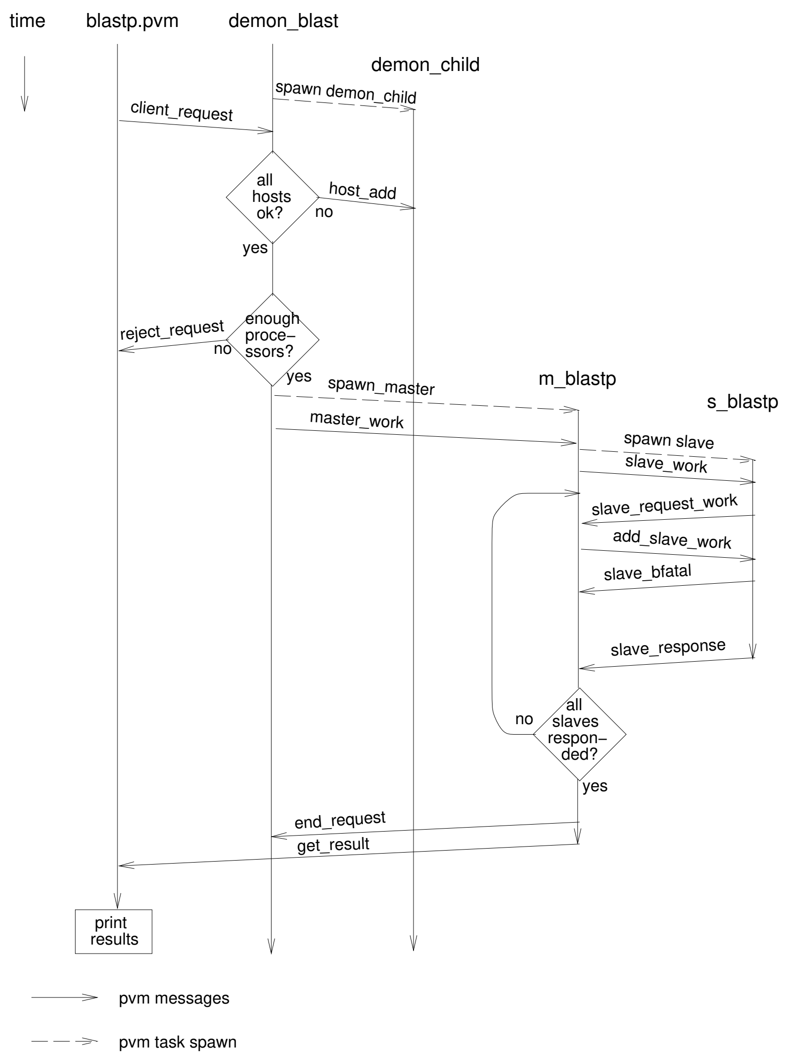

Figure 2: PVM Messages

Once the request is sent to `demon_blast` the client waits till it is notified by the master that results are ready in a file. The client reads this file and displays the results on stdout. The client also sets a timer. If the timer ends before results are returned, an error message is displayed. For the moment the timer is set to 5 minutes, which from experience, is enough to search even large databases. Figure 3 summarizes the whole program structure of the parallel blast programs.

4.4 Installing and running

The parallel blast programs are located in the directory `~/blast1.4/blastapp/net`. In that directory execute:

- `make clean` to delete everything except the necessary sources.
- `make tags` to make a tags table in file `TAGS` readable by emacs.
- `make` to make the executables.

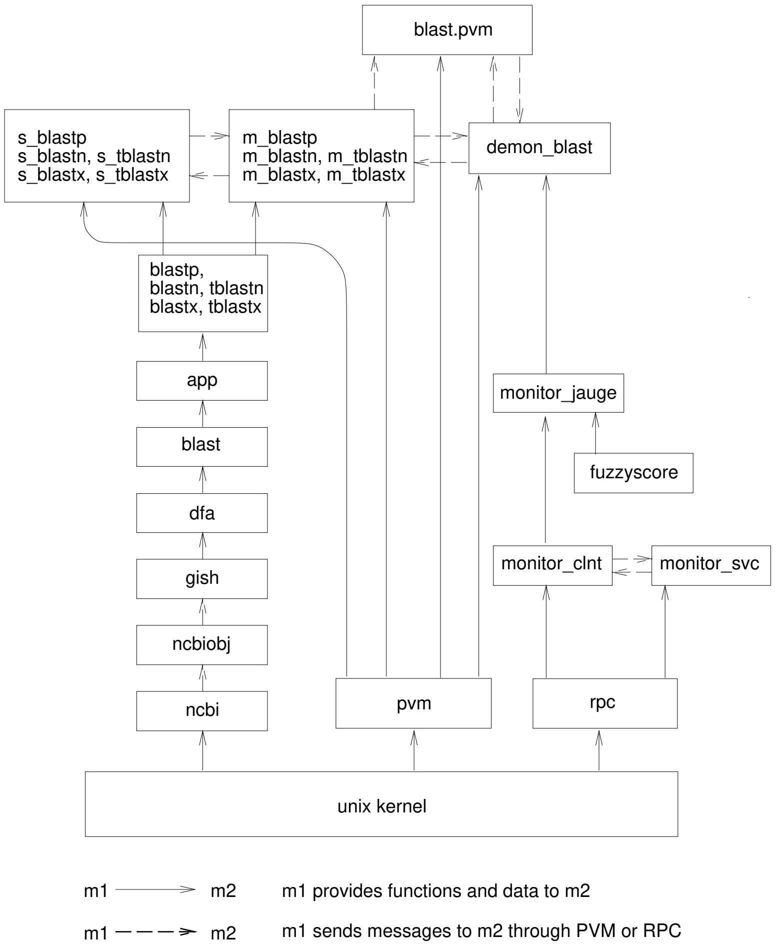

Figure 3: Program structure of the parallel blast programs.

Two additional environment variables were introduced:

- BLASTNUMSLAVES: number of machines to use.
- BLASTNUMCHUNKS: number of chunks to cut the database into.

If these envrionment variables are not defined, the values are taken from the equally named constants defined in `blast.pvm.c`.
The problem is, when a new version of blast programs arrives. Blast programs will probably always consist of the parts:

- parse arguments
- do searching
- print results

So, try to follow our method: Let the master and the slaves do the parsing of arguments, let the slaves do the searching, and let the master print the results. If the new version is not very different, then, by modifying a little bit the Perl script `net_patch.pe` you might be able to generate the parallel blast

programs (`net_blastp.c`, etc.). But probably you will have to change also something in the module `results.[ch]`, because the mapping, unmapping and adding of the results is done there.
Minor updates will have perhaps to be done, when a new version of PVM or of the operating system comes out. The blast programs depend on the very portable ncbi and gish libraries, so most probably, recompilation of the programs will be enough.
To run the parallel blast programs, you have to start the PVM deamon on as many machines as possible, start the blast deamon `demon_blast` and execute the client `blast.pvm`. A simple session looks like this:
In one xterm start `pvm` and `demon_blast`. These are started just once.

```
% pvmd ~/pvm3/bin/some_hosts.pvm &
% demon_blast ~/pvm3/bin/some_hosts.pvm
```

And in another start the parallel blast application with the same arguments as in the serial version:

```
% blastp.pvm swiss ~/queries/protpart1.seq -batch
```

5 The servers

There are three possible ways to submit requests to our server: By direct connection through a port, by filling out a mosaic page, and by e-mail. Figure 4 shows the program structure of our network and mail server.

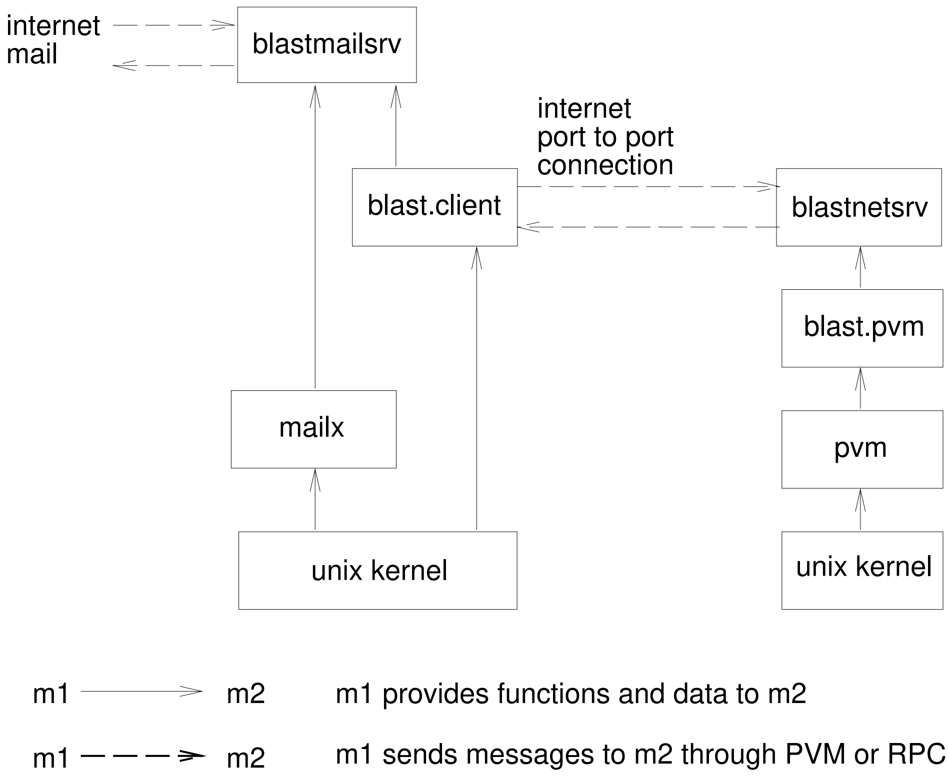

Figure 4: Program structure of the network software

5.1 Network client and server

5.1.1 Description

The network client and server, establish a connection on a port and communicate with each other through unix sockets. The network server `blastnetsrv` listens on a port for connection requests from remote clients. When the connection is established, the server forks itself, gets the command line arguments and the query sequence, executes the appropriate parallel blast program and redirects the results back to the client. When the child process finishes, it sends the signal SIGCHLD to the parent. The parent catches this signal with the `wait()` system call and gets the exit status for the child. We could ignore this signal, but it is usefull for statistical reasons, to know how many requests succeeded. By starting for each request a new process, many requests can be treated simultanaeously. Our network server uses the same protocol as the network client program from NCBI.

5.1.2 Installation and running

The network client program can be obtained from ncbi.nlm.nih.gov/blast/network/experimental/unix. On the didecs, as well on the disuns, it is installed in `~/blast1.4/netclient`. To get more information about the network service, get the file README from ncbi.nlm.nih.gov/blast/network/experimental. A local copy is kept in `~/man/blastnet`. Instructions for installing are given in the provided INSTALL file. Before installing the network client program, the libraries `ncbi, gish` and `dfa` must have been installed. For `ncbi` the object files are not necessary, just the include files in the directory `ncbi/include`. The port to use is defined by the constant `BLASTPORT` in `blastsrv.h` and is currently set to 5555. In that directory execute:

- `make clean` to delete the executables.
- `make` to make the executables.

The network client for a blast program has the same name as the blast program with a .remote suffix (ex: `blastp.remote`). It takes the same parameters as the original blast program. By default it calls the hosts at NCBI. There are some additional parameters, which must be specified before the database argument:

- `-b` To submit request as batch job.
- `-s hostname` To try the blast server on host `hostname` first.
- `-S hostname` To try only the blast server on host `hostname`.

For example, to submit a request on `didec28.epfl.ch` as a batch job, use the command:
`% blastp.remote -b -s didec28.epfl.ch swiss ~queries/protpart1.seq`
The network server program is called `blastnetsrv` and is located only on the didecs in `~/blast1.4/netserver`. In that directory use:

- `make clean` to delete the executables
- `make` to make the executables

To start the network server use the command: `% blastnetsrv`

`blastnetsrv` writes a message to `stdout` every time a client is connected. It also writes to two log files:

- /tmp/blastnetsrv.daylog: log file of the day. Contains the requests (command line and query sequence) submitted on one day. It is reset every new day.
- /tmp/blastnetsrv.yearlog: log file of the year. Contains the number of total and succesfull requests for each day of the year. It is reset every new year.

blastnetsrv.c will have to be adapted when a new version of a network client program comes out from NCBI.
The mosaic page located at http://ulrec3.unil.ch/software/EPFLBLAST_form.html is a nice interface to the network client program, which facilitates the entering of command line options. For the moment it calls the blast network server on didec28.epfl.ch.

5.2 Mail server

5.2.1 Description

The mail server functions as follows: when a mail message arrives, it is piped to the program blastmailsrv in the .forward file. blastmailsrv parses the mail message, makes a script which calls the network client program, and pipes the results to mailx which sends the results back to the sender of the request. The source is blastmailsrv.l and is written in lex. This may not be the best way to parse a text file, but since the format of the message is very simple, lex is strong enough to do that.
To learn more about the format of the mail message send a mail to blast@ncbi.nlm.nih.gov with the word HELP in the body of the message. A local copy is kept in ~/man/blastmail. Here is an example of a mail query:

```
PROGRAM blastp
DATALIB protpart
FILTER seg+xnu
MATRIX blosum62
BEGIN
>sp|P11684|10KS_HUMAN CLARA CELLS 10 KD SECRETORY PROTEIN PRECURSOR (CC10).
MKLAVTLTLVTLALCCSSASAEICPSFQRVIETLLMDTPSSYEAAMELFSPDQDMREAGA
QLKKLVDTLPQKPRESIIKLMEKIAQSSLCN
```

5.2.2 Installing and running

The mail server is located on the disuns because mail does not work correctly on the didecs. Our mail server is called blastmailsrv and is under ~/blast1.4/mailserver.
To install the mailserver you have to install the network client programs. Then, just run make in that directory to make the executable and the .forward file.
It will be necessary to modify blastmailsrv.l when NCBI changes the mail message format or adds new options, in order to provide a compatible service with NCBI. The mail server runs on the machines disun1 through disun30 which run Solaris 5.3.
In order to submit a mail request, you have to send a message to blast@disun10.epfl.ch for example. Examples of queries for mail requests are in the directory ~/queries/*.mq, 'mq' for mail query.

6 Databases

We created the account blastdat for the databases. This account exists on the didecs as well on the disuns, but all databases are stored on the didecs. Currently following databases are available:

Nucleotide databases:

- embl: from the the European Molecular Biology Laboratory.
- est: expressed sequence tags (a subset of embl).
- nr: non redundant embl+emblupdate+genbank.
- dnapart: small database for testing.

Protein databases:

- swiss: SwissProt.
- gp: GenPept.
- gpu: cumulative weekly updates of GenPept.
- nr: non-redundant SwissProt+GenPept+GenPeptupdates.
- protpart: small database for testing.

The databases are updated by ISREC[3].

7 History and persons related to this Project

- Warren Gish (NCBI), gish@ncbi.nih.gov, writes the serial blast programs.
- Prof. André Schiper (EPFL/LSE), schiper@di.epfl.ch, starts a joint project with ISREC to parallelize the blast programs. Contact persons on ISREC side are Philip Bucher, pbucher@isrec-sun1.unil.ch and Victor Jongeneel, vjongene@isrec-sun1.unil.ch.
- Eric Masson (EPFL/LSE), masson@di.epfl.ch, parallelizes blastp during his diploma thesis in the wintersemester 93-94.
- Xavier Défago, defago@di.epfl.ch, and Christian Javet (EPFL/LSE), javet@di.epfl.ch, add the load distribution by asking the load of each machine through RPC, summer 94.
- Alexandros Pappas (EPFL/LSE), pappas@di.epfl.ch, parallelizes the rest of the blast applications, upgrades from version 1.3 to 1.4, and writes a network and mail server, autumn 94.
- Kay Hoffmann (ISREC), khofmann@isrec-sun1.unil.ch, tests the parallel blast server, writes a mosaic page to interface to our blast network server, installs and updates the necessary databases at our site.

References

[Jav Def 94] C. Javet, X. Defago, Distribution de la charge pour BLAST, internal report, Jul 1994, EPFL/LSE

[Kar Al 90] Karlin, Samuel and Stephen F. Altschul (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. Proc. Natl. Acad. Sci. USA 87:2264-68.

[3] Swiss Institute for Experimental Cancer Research in Epalinges s/Lausanne

[Masson94] Eric Masson, Portage d'une application de biologie moléculaire d'un environnement multiprocesseurs vers un réseau de stations", diploma thesis, Feb 1994, EPFL/LSE

[PVM 93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam. PVM 3 User's Guide and Reference Manual. Oak Ridge National Laboratory, May 1993.

[Stevens 90] R. Stevens, Unix network programming, Prentice Hall, 1990

[Swarz 93] Randl L. Swarz, Learning Perl, O'Reilly, 1993.