CABIOS

Vol.10 no 1, 1994
Pages 31 – 34

Hierarchical Access System for Sequence Libraries in Europe (HASSLE): a tool to access sequence databases remotely

R.Doelz

Abstract

Sequence databases in biology are growing exponentially. Not only are large sites needed to keep the data, but the number of customers is continuously increasing. Network access plays a key role in utilizing remote resources. However, both synchronous and asynchronous access require tools that are currently non-standard in molecular biology computing. Additionally, information discovery of today frequently focuses on centers rather a hierarchically interconnected facilities. HASSLE (Hierarchical Access System for Sequence Libraries in Europe) is an implementation of an application-independent, user-transparent access tool in molecular biology. It features tools for both clients and information providers to permit accounting and/or prioritization on various levels. HASSLE focuses on the network aspect of the molecular biology computing and assumes that it is possible to have database applications available as remote 'services' (programs, program packages or utilities) which can be started by a simple command script after a suitable feed of datafiles. The current system provides these services for searching with programs like FASTA or BLAST which are compiled as obtained from vendors or servers.

Introduction

File servers in molecular biology are gaining increasing importance as the growth of sequence databases makes it necessary to update the database more frequently than the releases shipped on media like CD-ROM every 2–3 months. However, the servers as such need to get updates as frequently as possible in order to keep their knowledge as up-to-date as possible. In Europe, the EMBnet project (Doelz, 1992) has generated several national nodes which carry on-line duplications of the EMBL sequence database (Higgins *et al.*, 1992). However, serving the data to a computer network excludes end-users who are neither willing nor prepared to download an entire database, or updates thereof, before using the data with a retrieval system to find homologies, accession numbers or similar. End users employ electronic mail as a tool to query servers, like the one at EMBL (Higgins *et al.*, 1992) which will supply data, program code or results of sequence searches as desired. However, additional services are supplied at other servers, and many servers offer identical or similar services. In order to search for specific data, the user must know the addresses of different mail accounts, the particular syntax used, and the usage of electronic mail, including file inclusion and extraction.

The Hierarchical Access System for Sequence Libraries in Europe (HASSLE) is targeted on an improved integration of remote facilities. To use HASSLE, a workstation or larger computer connected to the internet is required. Remote tools and services appear to the researcher as if the service were local. Immediate response can be expected for quick jobs such as sequence retrieval or contents listing of the database. Exhaustive sequence searches will be run in the background after suitable interactive input. In contrast to MAIL-based services, the results appear directly in the researcher's directory or file space without interaction. HASSLE knows about known services, and learns from other HASSLE systems to query for alternative services if required.

System and Methods

HASSLE consists of three different components: local tools, the HASSLE kernel and the interpreter on the HASSLE server, which is a local script at the remote site.

The system was developed on Silicon Graphics hardware (Indigo). The C compiler employed (v. 3.10 on Silicon Graphics) is plain ANSI C. HASSLE can be compiled on IRIX 4.0.1 or higher, SunOS 4.3, ULTRIX 4.1 on either Vax or RISC processor, OSF/1, and Convex OS 9.1. The HASSLE program has also been successfully compiled on Vax and AXP VMS running either of the UCX 1.3 or 2.0, 3.0, TCPware, or the MULTINET implementations of the TCP/IP protocol.

The scripts used by the remote HASSLE server can be ported to any Unix variant as long as the (Unix) csh script language is available. The VMS version of the HASSLE server and associated scripts are also available.

The local tools required to address HASSLE services were written in either (Unix) csh or VMS (DCL) script languages. FORTRAN code (standard F77) was used to start remote sequence searches within a reformatting step (BLAST, see example below). In order to utilize the GCG software file format (Devereux *et al.*, 1984) the GCG procedure library v. 7.x was used, which requires a GCG software license at the computer used with HASSLE.

The HASSLE software uses Internet port 375 as assigned

Biocomputing, Biozentrum der Universitaet, CH 4056 Basel, Switzerland

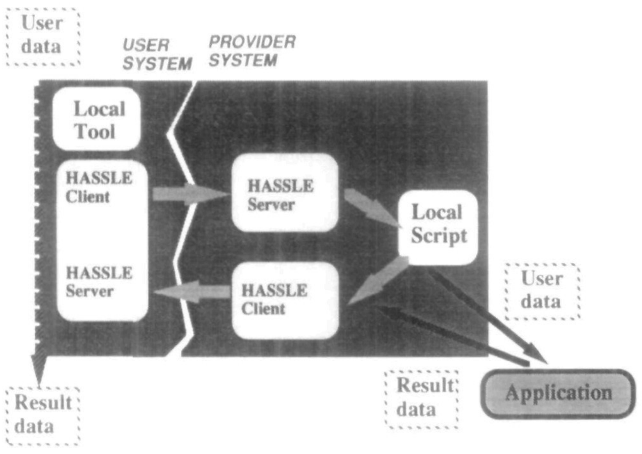
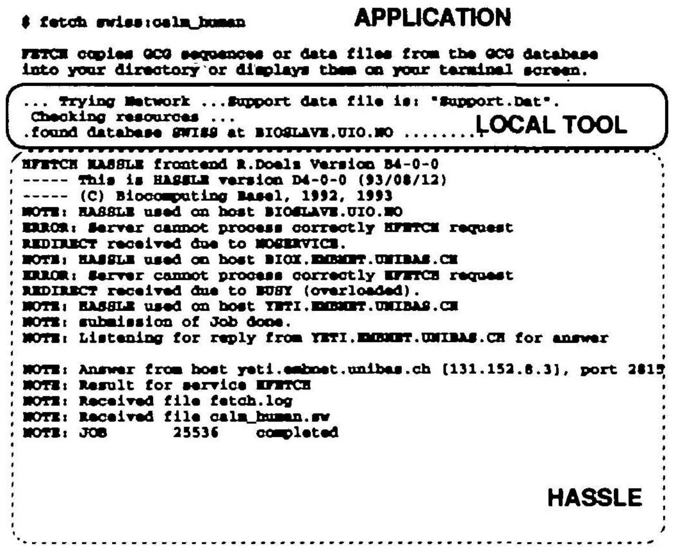


Fig. 1. Basic flow scheme of HASSLE. The grey arrows symbolize the program flow, whereas the dashed arrow represents the virtual flow of data seen by the user. The hashed area symbolizes the local user system and the remote provider system, separated by a network



```
$ fetch swiss:calm_human

FETCH copies GCG sequences or data files from the GCG database
into your directory or displays them on your terminal screen.

... Trying Network ...Support data file is: "Support.Dat".
 Checking resources ...
.found database SWISS at BIOSLAVE.UIO.NO ........
HFETCH HASSLE frontend R.Doelz Version D4-0-0
----- This is HASSLE version D4-0-0 (93/08/12)
----- (C) Biocomputing Basel, 1992, 1993
NOTE: HASSLE used on host BIOSLAVE.UIO.NO
ERROR: Server cannot process correctly HFETCH request
REDIRECT received due to NOSERVICE.
NOTE: HASSLE used on host BIOX.EMBNET.UNIBAS.CH
ERROR: Server cannot process correctly HFETCH request
REDIRECT received due to BUSY (overloaded).
NOTE: HASSLE used on host YETI.EMBNET.UNIBAS.CH
NOTE: submission of Job done.
NOTE: Listening for reply from YETI.EMBNET.UNIBAS.CH for answer

NOTE: Answer from host yeti.embnet.unibas.ch [131.152.8.3], port 2815
NOTE: Result for service HFETCH
NOTE: Received file fetch.log
NOTE: Received file calm_human.sw
NOTE: JOB         25536     completed
```

Fig. 2. HASSLE screen dump of a session with the FETCH program and the HFETCH tool. The entire procedure completes in a couple of seconds. The desired file (calm__human.sw) is directly placed in the default directory of the user.

by US agencies. This service must be added to the network configuration of the corresponding HASSLE server system. Additional ports are used on demand but assigned temporarily.

Operation principle (algorithm)

HASSLE uses several components to achieve transparent functionality which can be described as a remote procedure batch facility. In the following text, a 'client' denotes a program that seeks to connect to a 'server' program waiting for requests.

A HASSLE client starts a query directed to a well-known remote HASSLE server. After having obtained data from several configuration databases, the remote HASSLE server starts a dialog or redistributes the job to another remote HASSLE server. During the dialog, all datafiles needed, and a command file are transmitted to the remote HASSLE server. The final message is an execution command sent by the local HASSLE client. Then, the HASSLE client mutates to a local HASSLE server, i.e. it breaks the connection and waits for input. It accepts the connection of the remote HASSLE system, which, after the completion of the job, mutated into a HASSLE client, in order to receive the results of the job. A functional flow scheme is shown in Figure 1.

It should be emphasized that all data transmission throughout the HASSLE protocol are encrypted and compressed. The current implementation achieves a text file throughput that is ~80% of the performance if the files would have been compressed independently, and transferred via established programs like ftp.

Important intrinsic features of HASSLE are authorization and independency of location. Whereas a network access to a computer usually requires an account and a password at the remote system, HASSLE uses an independent authorization which makes it possible to probe the facility with a specific 'anonymous' user/authorization key. The normal HASSLE user and authorization data are independent of the operating system and therefore can cover 'generic' accounts a well as groups, institutes or individuals. Secondly, the location of the service must not necessarily be known to the user because the built-in database facilities allow for 'learning' of other HASSLE servers and automatic redirection.

Figure 2 shows a screen dump as the end user will see it if the FETCH program with the HFETCH tool is employed. This session was run on a Norwegian computer in order to obtain an entry in the SWISSPROT database. If the entry is not present locally, the HFETCH tool screens a configuration file for possible resources and will approach the service at the nearest location. As this fails, HASSLE learned from the host approached that another service is available in Switzerland. As the computer in Switzerland is busy, the Norwegian customer is again redirected to a host which finally does provide the requested data. Note that the customer site in Norway knew of only one service, and that the final destination was evaluated within the HASSLE dialog.

Implementation

HASSLE components

HASSLE in its current implementation permits only asynchronous job execution. It consists of several programs and scripts which run either locally (local or 'user' system) or on the service provider's machine (remote or 'provider' system). All HASSLE server and client executables are currently generated from one single source code.

Detailed information on the general features of the protocol are being published elsewhere (Doelz, 1993). The main components—Local Tool, HASSLE and Provider Local Script, as shown in Figure 1—need to be configured with straightforward site-specific data and will communicate via sockets and files.

The authorization database

The HASSLE dialog starts with an authorization which is used to identify, and account for, the user who requests services. In addition, a 'credit' is diminished each time a particular service is used. This implies that an 'anonymous' user will be entered in this database at the first time of access, and registered there with a default (configurable) credit. Once a special account is used, and agreements are made with the service provider, the credit in the authorization database will be significantly higher than for the non-committed anonymous access.

The service database

The database used by HASSLE to determine the system where the job finally executes contains data on the name of the requested service, the location, and a 'cost' as well as a 'priority' parameter. The cost parameter is a number of credit units subtracted from the user's account if the service is used. If two services have the same name and the same priority, the first name in the list will be used.

Discussion

Network aspects

Existing services offered on networks frequently suffer from being unknown, overloaded or unreachable. This is due to the fact that there is no automatic 'broadcasting' of services, or redirection in case of heavy load or other problems. HASSLE broadens the scope for remote system utilization by end users. This is achieved by three main features:

- *Ease of use*—The end user does no longer need to process his/her mailbox or similar communication devices in order to view results. The evaluation of remote data is no longer different from local data inspection.
- *Ambiguity*—Given the assumption that several types of services can be provided by more than one HASSLE server, resources are no longer vulnerable with respect to maintenance or local network problems at the remote sites.
- *Accounting*—Cost sharing can be implemented if funding structures change or request accounting documentation.

Another important feature of the HASSLE broadcasting of service data is the prioritization of services. In combination with the ambiguity discussed above, it is now possible to have a true hierarchy in accessing services. That is, a particular sequence database will be considered first at the site which is selected by a priority parameter rather than by the user. As long as the quality of service is identical, the priority parameter should be set in a way that the local (or nearest) source is queried before choosing long-distance service links.

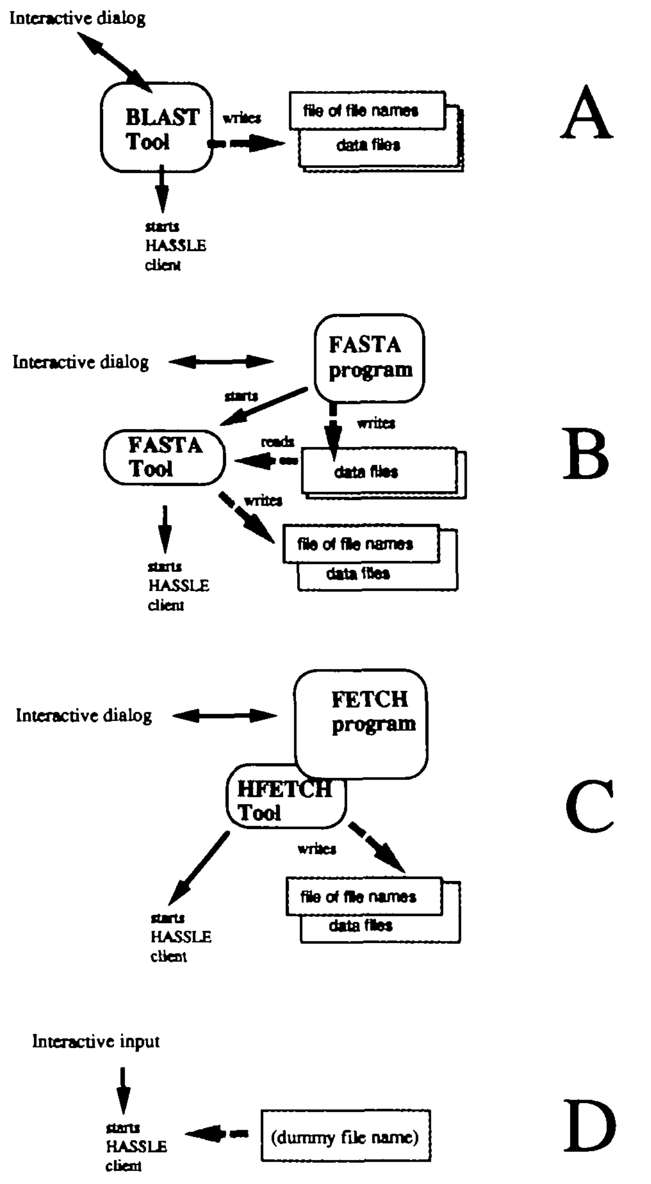


Fig. 3. Design examples of HASSLE local tools. (A) Standalone tool, (B) modifed command file for batch submission makes exisiting application a HASSLE frontend, (C) subroutine as part of an established program and (D) a symbol definition calls HASSLE from the command line. As there are no local data needed, a dummy file name is assumed in this case.

Service provider aspects

Network service providers frequently get swamped by many requests originating from few sources, which delay response time for services provided to those who access the service rarely. HASSLE allows for implementation of a mechanism to filter high-frequency requests, and process these accordingly in sequential fashion.

Table I. HASSLE tools available in the standard distribution of HASSLE

	Functionality		
Tool	Search BLAST	Search FASTA	Retrieve FETCH
Customer, Unix	X		X
Customer, VMS	X	X	X
Provider, Unix	X	X	X
Provider, VMS		X	X

The accounting feature of HASSLE offers powerful tools to the service provider to regulate the access to the HASSLE server. Whereas current MAIL server implementations can account for the origin of the query only if each address or class of address is entered manually, HASSLE permits threefold accounting procedures by manipulating the accounting database. Possible configurations include, but are not limited to, anonymous access for anybody, some sites, some user groups or any mixture of these.

If only a department or particular user group benefits from extended access, the local HASSLE client call will need to query the local group parameter from the operating system and call the remote HASSLE server with a authorization other than the 'anonymous' access.

HASSLE has also been successfully used for database synchronization following the 'polling' mechanism. Instead of transmitting a full database, a dedicated tool generates only a subset of the database containing the entries missing at the client side. An extensive set of tools has been created at our biocomputing laboratory and will be made available soon.

Implementation of new HASSLE services

Local tools for the implementation of new services can be designed and implemented easily. The important step in a new service definition is to define a data exchange formalism (e.g. definition of the sequence format). The local tool at the remote node can usually be derived from an existing script since the error-handling and parsing of command files is very similar. The local tool at the client side can be any combination of the possibilities as outlined in Figure 3. A summary of currently released tools is listed in Table I; further tools are available on request.

For example, for the BLAST tool a 'front end' to the BLAST software as provided by the NCBI (Altschul *et al.*, 1990) was composed. As the local installation in Basel uses the GCG software, the front end was written using the GCG procedure library (Devereux *et al.*, 1984).

The second option for creating new HASSLE services is to modify existing scripts which submit predefined data and procedures in the batch job. The GCG software provides such a mechanism which writes data into so-called 'initialization' files, which, in combination with suitable command files, can be processed in batch mode. The command to submit the job is defined as symbol on Vax/VMS and as alias on the Unix operating system. To make the system compatible with HASSLE, the symbol GCGSUBMITCOMMAND is replaced with a HASSLE tool, e.g. @HASSLESTUF:decide.com on Vax/VMS. This command procedure parses the data written by the GCG program and checks for the possibility of remote execution via HASSLE by reading a configuration table. If no local dependencies are found (e.g. no file of sequence names), the job is injected into HASSLE. Note that this example of running FASTA (Pearson and Lipmann, 1988) (example B in Figure 3) does not require any change to the program code but just adds a command file to the existing software.

Example C in Figure 3 indicates the possibility of extending a given code with one additional subroutine call, which will call HASSLE. While the user interface is retained, the FETCH program as provided by GCG is extended to call a subroutine 'tryhassle' in order to write all files needed, and call HASSLE subsequently.

The last example of HASSLE tools is applicable to applications that do not have local data files. For example, to get a description of the site which runs the HASSLE Provider suite, the command

```
% hassle -host ... -service generic (on Unix), or
$ HASSLE /HOST=.../SERVICE=GENERIC (on VMS)
```

is sufficient (example C in Figure 3). This implies that HASSLE can also be called interactively from the command line.

Availability

HASSLE and the HASSLETOOLS are available in source code from the author or from the bioftp.unibas.ch or nic.switch.ch FTP server.

Acknowledgements

The compression during HASSLE data transfer was contributed by L Rosenthaler, Biocomputing Basel. This work was supported by Basel University and a grant from the Swiss National Science Foundation (NF) to R.D.

References

Altschul,S.F., Gish,W. , Miller,W., Meysers,E.W. and Lipman,D.J. (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Devereux,J. Haeberli,P. and Smithies,O. (1984) A comprehensive set of sequence analysis programs for the VAX. *Nucleic Acids Res.*, **12**, 387–395.

Doelz,R. (1992) The EMBnet project. *Comput. Networks ISDN Syst.*, **25**, 464–468.

Doelz,R. (1993) HASSLE. A software to manage distributed processing on heterogenous systems. Published on the Internet on *http://beta.embnet.unibas.ch/*, further manuscript in preparation.

Higgins,D.G., Fuchs,R. Stoehr,P.J. and Cameron,G.N. (1992) The EMBL data library. *Nucleic Acids Res.*, **20**, 2071–2074.

Pearson,W.R. and Lipman,D.J. (1988) Improved tools for biological sequence analysis. *Proc. Natl. Acad. Sci. USA*, **85**, 2444–2448

Received on April 30, 1993; accepted on September 8, 1993