User's Guide for HMMER's Server and Remote Client Programs

High-performance biological sequence analysis using profile hidden Markov models

Nicholas P. Carter, Sean R. Eddy
and the HMMER development team

http://hmmer.org
Version 3.3.2; Nov 2020

HMMER development is supported in part by the National Human Genome Research Institute of the US National Institutes of Health under grant number R01HG009116. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Contents

Introduction

`Hmmserver` and `hmmclient` are replacements for the `hmmpgmd` and `hmmc2` programs provided by earlier versions of HMMER. Like `hmmpgmd`, `hmmserver` is a persistent, long-running, service that provides high-performance homology searches by caching sequence and HMM databases in RAM and distributing the work of each search across many computers and threads. `Hmmclient` is a full-featured command-line client application for `hmmserver` that submits searches to a running server and displays results in a format that is as close as possible to that of the `hmmsearch`, `hmmscan`, `phmmer`, and `jackhmmer` programs. It is thus a significant upgrade to the `hmmc2` program, which was more of a debugging tool for `hmmpgmd` than a full client program.

`Hmmserver` improves on `hmmpgmd` in several ways:

1. Parallelization has been significantly improved. `Hmmserver` uses a work-requesting protcol to distribute work across the machines that make up a server and a combination of work-requesting and work-stealing to balance load across the worker threads on each worker machine. Together, these parallelization improvements reduce search time by approximately 50% when the server is run on hundreds of cores.

2. `Hmmserver` reads standard FASTA and HMM files as its inputs, instead of the special-format files `hmmpgmd` requires. This allows it (at some cost in memory usage) to return the full results of a search. In contrast, `hmmpgmd` replaces the metadata (name, accession, taxonomy ID, etc.) associated with each sequence or HMM in a database with a unique ID, requiring database or file accesses to retrieve that metadata before results can be returned, which has proven to be a significant performance bottleneck.

3. `Hmmserver` can load multiple sequence and/or HMM databases into memory simultaneously and accept searches to any of them in any order. In contrast, `hmmpgmd` could only load a single sequence or HMM database. This required users to run separate instances of `hmmpgmd` for sequence and HMM data, potentially wasting resources if the

fraction of searches to each type of database was not predicted accurately. Similarly, if a user wanted to provide the ability to search multiple databases of a given data type, they had to either run a separate instance of `hmmpgmd` for each database or create a single data file containing all of the data in all of the databases and implement searches of individual databases as searches of sub-ranges of the unified database. Generating these unifiied databases any time one of the individual databases changed proved to be time-consuming for our partners at the European Bioinformatics Institute (EBI), making it more difficult for them to provide the latest version of each database on their `hmmpgmd` server.

4. `Hmmserver` implements a more-flexible sharding scheme than `hmmpgmd`. Originally, `hmmpgmd` required each worker node to load the entire database that `hmmpgmd` caches into RAM. As sequence database sizes increased, the amount of RAM this requried became a problem, so `hmmpgmd` was modified to shard its database by loading $\frac{1}{N}th$ of the database onto each worker node of a system with N workers.

5. `Hmmserver` is implemented as an MPI application, in contrast to `hmmclient`, which was a set of independent programs that communicated via UNIX sockets. This allows `hmmserver` to take advantage of high-bandwidth computer networks to improve performance, and simplifies the process of starting an `hmmserver`, as the entire server can be started as a single MPI application across multiple machines.

Installation

`Hmmserver` and `hmmclient` are included in the standard HMMER distribution package, but HMMER must be compiled with MPI support turned on for `hmmserver` to be useful, so it is likely you will have to compile HMMER from source to use the server. Builds of HMMER without MPI support enabled will create a `hmmserver` executable, which will print an error message and exit if run. `Hmmclient` does not require MPI support, so any installation of HMMER should have a working `hmmclient` application.

To build HMMER with MPI support, obtain a source-code copy of HMMER from hmmer.org[1] or elsewhere and go through the standard configuration/build process, with one change: you must pass the `-enable-mpi` flag to our configure script to cause HMMER to be built with MPI support:

[1] The HMMER User's Guidehas more detail on how to do this.

```
% ./configure --prefix=/your/install/path --enable-mpi
```

After configuring the build to enable MPI, running `make` as normal will build HMMER with MPI support and create a functional `hmmserver` program. One note, however, is that building HMMER in this way will make all of our programs dependent on the MPI libraries. If your system (like the one we develop on) requires the user to take steps to make the MPI libraries available, you may find it more convenient to build most of HMMER without MPI support, do a separate build with MPI support enabled, and copy the `hmmserver` executable from the MPI-enabled build to the non-MPI build so that it is the only application that loads the MPI libraries.

Once HMMER has been built, `hmmserver` and `hmmclient` will be available in the `src` directory of the distribution, and can either be run from there or made available system wide by running `make install`.

Usage

Hmmserver

`Hmmserver` must be run as an MPI process with at least one rank for the master node and one worker rank for each shard that the target database(s) will be divided into. The different implementations of MPI vary in how this is done, but a common invocation of `hmmserver` would be[2]:

```
% mpirun -n <number of ranks> hmmserver -num_dbs <n>
```

[2] On many systems, MPI jobs need to be sumbitted through a job control mechanism such as SLURM. See your system documentation for information on how to use the job control system on your cluster.

One challenge that comes from making `hmmserver` an MPI process is that the master node of the `hmmserver` will run as rank 0 of the MPI process, but the MPI runtime assigns ranks to hosts in an unpredictable manner. To address this, the master node of `hmmserver` determines the host name of its machine and outputs that to standard output at the beginning of execution. Alternately, the --host option to mpirun/mpiexec can be used to specify the set of computers that `hmmserver` will run on, and rank 0 will run on the first computer in that list[3]

[3] This may be an implementation-dependent behavior. The documentation for many MPI implementations states that this is what will happen, but we do not believe that this is a required behavior for all implementations of MPI.

One key decision when starting an `hmmserver` is the number of shards (pieces) that each database will be divided into. In an `hmmserver` with s shards, each worker node will load $frac1sth$ of each database into memory, but will then only be able to process the parts of each search that corresponds to the items it has in RAM. This creates a tradeoff between performance and memory requirements: increasing the number of shards a server uses will decrease the amount of memory used on each worker node, but will reduce performance by restricting the set of nodes that each shard's fraction of the search can be distributed across. While the only hard requirement is that the number of worker nodes must be at least equal to the number of shards, using a number of shards that is not an integer divisor of the number of worker nodes will reduce performance by having one or more shards with fewer nodes assigned to them than others. For example, a server with eight worker nodes could be set to divide its databases into three shards, but this would result in two of the shards having three worker nodes assigned to them, and one shard only having two, in which case the shard with two worker nodes would generally take longer to search than the other two, making it the bottleneck on performance.

Once started, `hmmserver` will continue running until it receives a shutdown command from a client, at which point it will terminate. As an (admittedly weak) protection against spurious/malicious shutdown requests, the `--password` option to `hmmserver` can be used to specify a password that must be sent along with a shutdown command for the command to take effect.

Hmmclient

The `hmmclient` application provides a command-line interface to send searches to a running `hmmserver` and display results in the same manner as the `hmmsearch`, `hmmscan`, `phmmer`, and `jackhmmer` programs. A typical use of `hmmclient` is:

```
    % hmmclient -s <servername> --db <d> <name of the file containing the sequence(s)
or HMM(s) to be searched>
```

`Hmmclient` accepts the same set of command-line options for controlling output, managing the acceleration pipeline, defining how HMMs are constructed from sequences, and setting reporting thresholds as `hmmsearch`, `phmmer`, `hmmscan`, and `jackhmmer`. See the documentation for those programs for more information on those options. `Hmmclient` does not support the `-cpu` option that sets the number of worker threads in our command-line applications, as that decision is made at the time the server is started.

In most cases, `hmmserver` infers the type of search to be performed from the types of the query and target objects. The exception is when the user wants to perform an iterative (`jackhmmer`-style) search. In that case, the `-jack <maxrounds>` option should be passed to `hmmclient`. Note that the `-jack` option is only valid when both the query object and target database are sequences.

For those converting from an hmmpgmd installation

For an administrator, the biggest change between `hmmpgmd` and `hmmserver` is the switch from independent programs to an MPI application. This allows `hmmserver` to be started with a single command, although the exact syntax of this command may vary between implementations of MPI. See your local system documentation for more information.

Making `hmmserver` an MPI application introduces the challenge of determining what machine the master rank (rank 0) of the server is running on, and thus which machine clients should send queries to. There are two ways to do this. The first is to examine the output of `hmmserver` shortly after the server starts, as the master rank will print its hostname on standard out. The second is to use options to the command that starts `hmmserver`, such as the `-host` flag supported by many MPI implementations, to control which machines `hmmserver` runs on and thus specify the machine that will run the master rank.

We attempted to keep the client interface to `hmmserver` as close as possible to that of `hmmpgmd`, but a few changes were necessary. The two most significant of these are:

- `Hmmserver` only allows a client to send one query to the server per socket connection, requiring that the client disconnect from the server after each search completes and request a new connection if it wants to send another request. In contrast, `hmmpgmd` allowed a client to open a socket connection and send multiple query requests before disconnecting, which, in some cases, led to clients monopolizing the server by sending large numbers of requests in one connection. Requiring a client to disconnect and reconnect between queries forces it to arbitrate with other clients for use of the server, making it harder for one poorly-behaved client to lock others out of the server.

- `Hmmpgmd` used two forward slash characters ("//") to indicate the end of a command, which worked because it only accepted text-formatted HMMs and sequences. `Hmmserver` also accepts HMMs as serialized `P7_HMM` structures, in order to support iterative search without the precision loss that converting an HMM to text format incurse. To support this, `hmmserver` requires two socket messages to transmit a command: one 32-bit message containing the length of the command being sent, followed by the command itself.

Hmmserver's Design

`Hmmserver` is an MPI application that uses a conventional master-worker topology, as shown in Figure 1. While the most-common use of `hmmserver` will distribute the MPI ranks across multiple machines, this is not required[4]. One MPI rank (rank 0) acts as the master rank, while the rest act as workers. The master rank handles communication with clients, receiving search requests from them and sending results back when each search completes. The master rank also co-ordinates the work of the workers, assigning them regions of the target database to search and collecting the hits that each worker finds into a single list. Worker ranks cache the server's target databases in memory and perform the work of each search, and can require significant memory, depending on the number and size of the target databases and the number of shards those databases are divided into. The master rank does not cache target databases, as it only needs to know the number of items in each database to distribute work across the worker ranks. Thus, it uses very little memory when the server is idle, but it may require significant amounts of memory to assemble the results of searches that return large numbers of hits. The decision to build `hmmserver` as an MPI application was a significant change from `hmmpgmd`, which was implemented as a set of independent programs that communicate with each other via UNIX sockets. The initial motivation for this change was to improve performance by taking advantage of high-bandwidth networks, which MPI typically does and sockets typically does not. During discussions with our collaborators at the European Bioinformatics Institute, they disclosed that managing `hmmpgmd`'s independent programs was a source of difficulty for them, so the switch to an MPI application provided a second benefit.

[4] If each MPI rank is not assigned to a separate computer, we strongly recommend using the `-cpu` flag to specify the number of worker threads on each worker rank. If this is not done, each worker rank will start as many worker threads as its machine has hardware thread slots, severely over-subscribing the hardware if multiple worker ranks run on the same machine.

Parallelization and Load Balancing

`Hmmpgmd` statically distributed the work of each search across the worker nodes, assigning $\frac{1}{Nth}$ of the items to be searched to each worker node in a system with N workers. This led to significant variance in the amount of time each worker took to perform its fraction of the search,

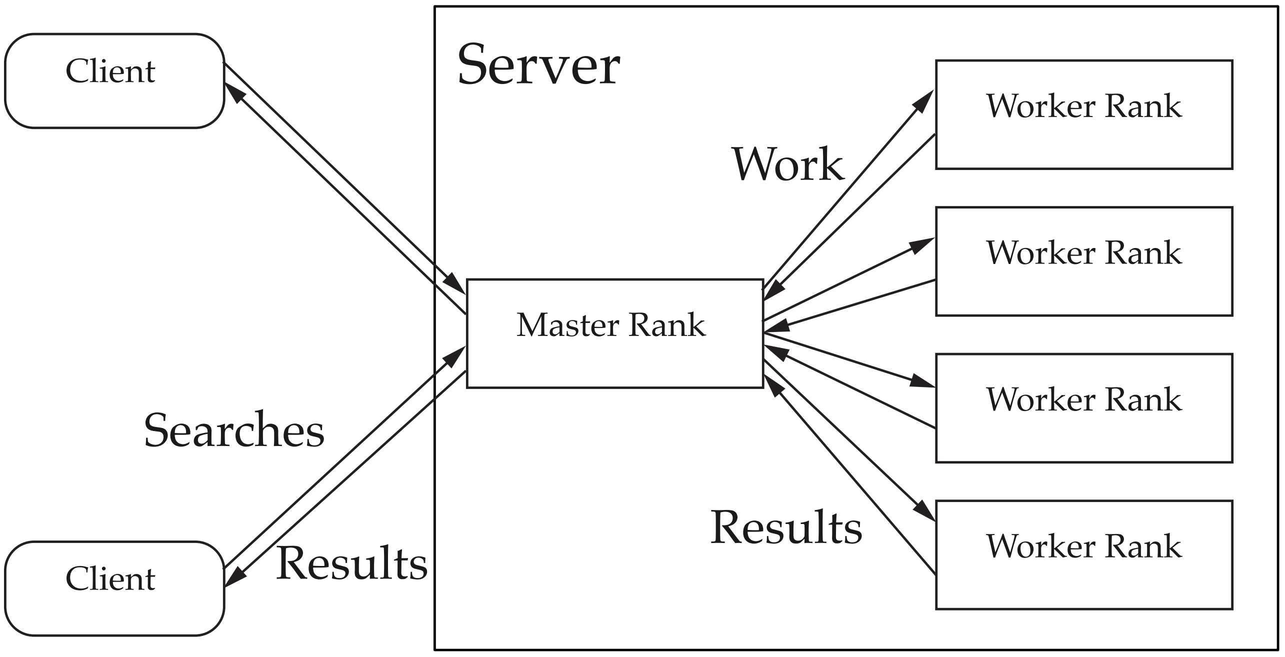


Figure 1: Hmmserver Architecture

reducing search performance. The root cause of this load imbalance is the fact that the amount of time to perform a sequence-HMM comparison varies greatly depending on whether the sequence and HMM are clearly not homologs, are close to being similar enough to be considered homologs, or are detected as homologs. This causes regions of the database that contain many homologs to take much longer to search than regions that contain few homologs. Thus, worker nodes that find many homologs take significantly longer to finish their fraction of a search than worker nodes that find few homologs. Worse yet, the distribution of homologs across the target database varies from search to search, making it impossible to find a static allocation of work to worker nodes that balances load well.

To address this, `hmmserver` dynamically distributes work across its worker node using a work-requesting system. The master node maintains a global list of the items in each shard that still need to be compared to complete the search. Worker nodes each maintain a local list of work (items to be searched) that they are responsible for. When the amount of work remaining on a worker node's local list drops below a threshold, it sends a request to the master node for more work. If the master node has any work remaining on its global list, it responds with a chunk of work (range of items) that the worker node is now responsible for. To trade-off load balancing against the amount of communication required, work chunks contain a large amount of work at the beginning of a search and decrease in size over the course of the search.

Within-Node Parallelization

Within each worker node, `hmmpgmd` uses a load-balancing system similar to how `hmmserver` distributes work across nodes. Each worker node maintains a list of the work assigned to it, and worker threads request chunks of work from the list as they complete their previously-assigned work. This creates a trade-off between load balancing and contention for the work list: if the work chunks are too small, the worker threads need to request more work frequently, and access to the work list becomes a bottleneck. On the other hand, if the work chunks are too large, the worker thread that grabs the last chunk of work from the list can wind up working for a significant amount of time after the other threads finish, increasing the time to complete each search.

`Hmmserver` improves load-balancing by adding work stealing to `hmmpgmd`'s within-node load balancing scheme. When a worker thread finds that the work list is empty, it examines the chunks of work that the other worker threads are working on and takes half of the work from the thread with the most to do. This spreads the work at the end of a search more-evenly across the worker threads, preventing one thread from significantly delaying the end of the search.

`Hmmserver` also adds split-phase processing of sequence-HMM comparisons to further improve performance. The final steps of hit and domain detection can take a significant amount of time, such that the threads processing the last few hits in a search can delay completion noticeably, particularly when running the server on a large enough number of machines to meet our one second average search time goal[5]. To address this, the server splits sequence-HMM comparisons into "front-end" (the filter pipeline) and "back-end" (the main stage) processing. Initially, all of the worker threads start processing the front end of comparisons. When a comparison passes all of the steps in the front end, it is placed in a queue for back-end processing, and one of the worker threads is switched to process comparisons out of the back-end queue. If the depth of the back-end queue exceeds a threshold, additional worker threads are switched to processing the back end of comparisons. Similarly, if the back-end queue drops below a depth threshold, worker threads are switched back to processing the front end of comparisons. This has the effect of prioritizing the back-end comparisons that take the longest to handle, decreasing the chance that a small number of long-running comparisons extend the overall search time.

[5] This was definitely true in the HMMER4 performance prototype of the server, as HMMER4's final stage trades run time for accuracy in some cases. We have not done a great deal of analysis on whether it is true in HMMER3.

Sharding

In the database community, *sharding* a database is the practice of dividing the data in a database across multiple computers, reducing the amount of storage required on each computer at the cost of requiring communication between computers when a database query needs to access data that is stored on multiple machines. In homology search, we use the term the same way, although, since each sequence-HMM comparison in a homology search is independent, sharding a database across multiple machines does not introduce extra communication. Instead, it restricts the server's ability to distribute work across multiple computers, since only some of the computers now have access to each sequence or HMM.

The initial version of `hmmpgmd` did not support sharding, instead requiring that the entire contents of the database file be loaded into memory on each worker node. As the size of sequence databases increased, the amount of memory this required on each node became a significant problem, so a simple version of sharding was added to `hmmpgmd`. This scheme always divided the database into as many shards as there were worker nodes, distributing the items in the database round-robin across the worker nodes. This minimized the amount of memory required per worker node, and, since `hmmpgmd` statically allocated the work of each search to worker nodes, did not reduce performance.

`Hmmserver` supports a more-flexible sharding scheme which, in combination with its improved load-balancing across nodes, allows the user to trade-off memory usage against performance. `Hmmserver` allows the user to specify the number of shards that the database will be divided into independently of the number of worker nodes as long as there is at least one worker node per shard. If the server has more worker nodes per shard, it loads each shard onto multiple shards and dynamically distributes the work of searching the shard across the nodes that have it loaded into memory. For example, a server with six worker nodes could be configured with one shard to deliver maximal performance, with six shards to minimize memory usage, or with two or three shards to provide greater performance than the six-shard case at lower memory usage than the one-shard case. Note that, while server will operate correctly if the number of shards is not an integer divisor of the number of worker nodes, this will lead to reduced performance because some shards will have fewer nodes assigned to them than others.

Client-Server Interface

Figure 2 shows the sequence of events involved in sending a search request to a `hmmserver` and receiving results back. A client initiates a search request by opening a socket connection to the server. It then sends a 32-bit message to the server containing the length of the command that will be sent, followed by the command itself. After the server completes the search, it sends a fixed-length serialized HMMD_SEARCH_STATUS object back to the client, followed by a HMMD_SEARCH_STATS object and one P7_HIT object for each hit the search found. The server then calls the shutdown function on its side of the socket to signal its intent to close the socket. Once the client has received all of the data, it closes the socket from its end, which, in turn, signals the server to close its end of the socket connection[6], at which point the server is ready to accept another search request.

[6] The server will also close its end of the socket if a timeout (default 30 seconds) expires after it executes the socket shutdown operation, to prevent clients from

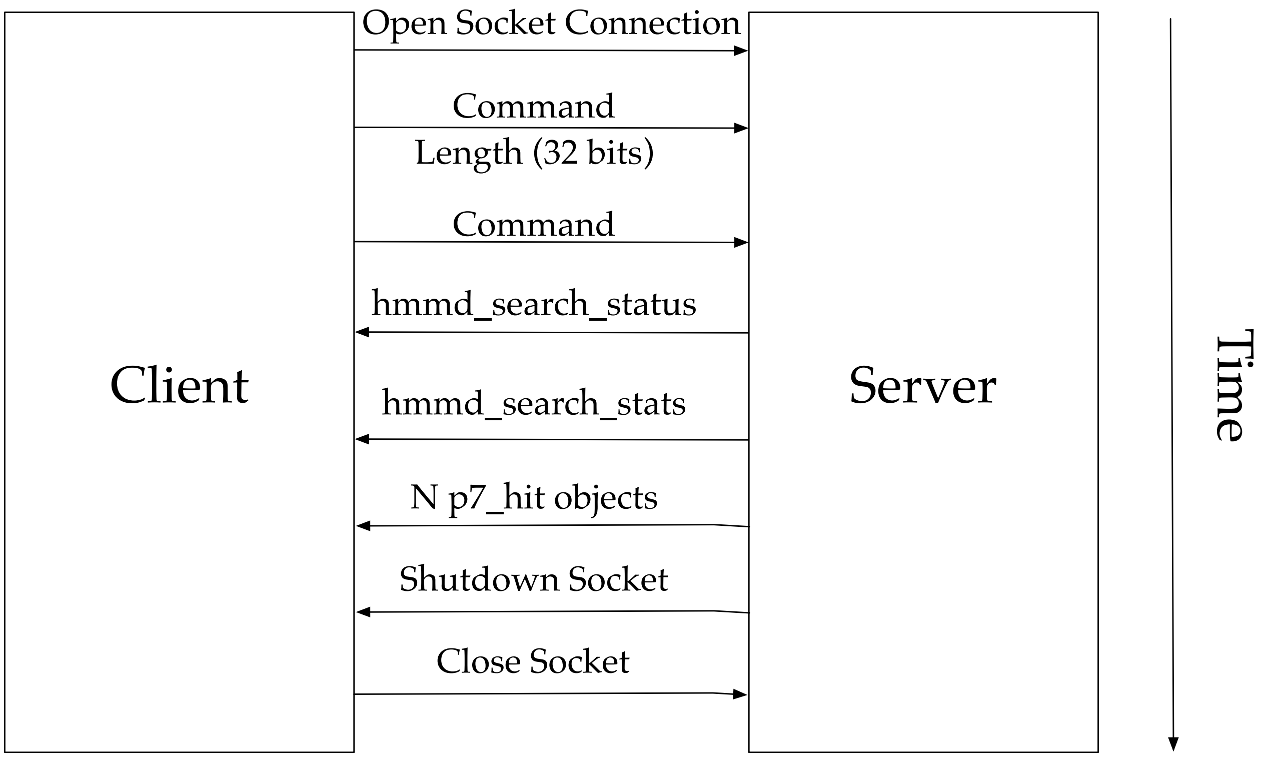


Figure 2: Client-Server Interface

The server can accept multiple client connections simultaneously, up to the limit set by the `--ccnts` flag passed to `hmmserver`, but queues the requests from clients and processes them sequentially. Limiting each client to one search per socket connection is new in `hmmserver`, as `hmmclient` allowed a client to keep a single socket connection open while it sent multiple search requests over the connection. This change was made to make it harder for clients to monopolize a server. In `hmmpgmd`, a client could potentially send thousands of requests over a single connection. With `hmmserver`, clients can still send many requests to a server, but having to drop and re-establish the socket connection between searches gives other clients a chance to access the server as well. Note that this "one search per connection" rule applies even to `jackhmmer`-style searches. `Hmmserver` does not implement multi-round searches internally. Instead a client must send a search request, use the results to create a query object for the next search, send that search, and so on.

Command Format

Commands are variable-length objects that must begin with a line of ASCII text. The server accepts two types of commands: search requests and server instructions.

The first line of a search request must begin with the sequence **`@--db <database #>`**[7], followed an options string containing the options to be passed by the search. The second and following lines of a search request contain the query object that will be searched against the specified database, which must either be a FASTA-format sequence, a text-format HMM, or an asterisk followed serialized `P7_HMM` object[8]

[7] For backwards compatibility, the server also accepts **`@--seqdb <database #>`** and **`@--hmmdb <database #>`** as the beginning of a search command, treating them as synonyms for **`@--db <database #>`**.

[8] The server examines the initial character of the query object to determine its format, and the asterisk indicates a serialized `P7_HMM`.

The following is an example of a search request command. The trailing slash characters are not strictly necessary, but are accepted for compatibility with `hmmpgmd`.

```
      @--db 1 --max -Z 42

>sp|Q6GZX3|002L_FRG3G Uncharacterized protein 002L OS=Frog virus 3 (isolate Goorha)
MSIIGATRLQNDKSDTYSAGPCYAGGCSAFTPRGTCGKDWDLGEQTCASGFCTSQPLCAR
IKKTQVCGLRYSSKGKDPLVSAEWDSRGAPYVRCTYDADLIDTQAQVDQFVSMFGESPSL
AERYCMRGVKNTAGELVSRVSSDADPAGGWCRKWYSAHRGPDQDAALGSFCIKNPGAADC
KCINRASDPVYQKVKTLHAYPDQCWYVPCAADVGELKMGTQRDTPTNCPTQVCQIVFNML
DDGSVTMDDVKNTINCDFSKYVPPPPPPKPTPPTPPTPPTPPTPPTPPTPPTPRPVHNRK
VMFFVAGAVLVAILISTVRW
//
```

`Hmmserver` determines the type of search to be performed from the types of the query object and target database, constructing an HMM from the query sequence if both the query object and target database are sequences. `Hmmserver` does not directly support iterative (`jackhmmer`-style) searches. Instead, the client must send the initial search to the server, construct an HMM from the results, and repeat as necessary. Support for serialized `P7_HMM` query objects is new in `hmmserver`, and was added to support iterative search. Converting a `P7_HMM` data structure into a text-mode HMM causes some loss of precision because the HMM file format uses a fixed number of digits for each floating-point value, which causes differences in output between iterative searches with `hmmserver` and `jackhmmer`. Sending the HMM for each search round as a serialized data structure eliminates this problem.

Server instructions begin with an exclamation point, followed by the instruction and any arguments. The only server instruction `hmmserver` currently supports is the shutdown command, which has the following format:

```
!shutdown <optional password>
```

Reply Format

When a `hmmserver` completes a search, it sends a serialized[9] HMMD_SEARCH_STATUS object, as shown in Figure 3, back to the client. The status field of the object contains an Easel result code, which is either ESLok if the search completed correctly, or an error code if something went wrong. The type field of the object is either HMMD_CMD_SEARCH or HMMD_CMD_SCAN depending on whether the target database contained sequence or HMM data. Finally, the msg_size field contains the length (in bytes) of the remaining data that will be sent to the client.

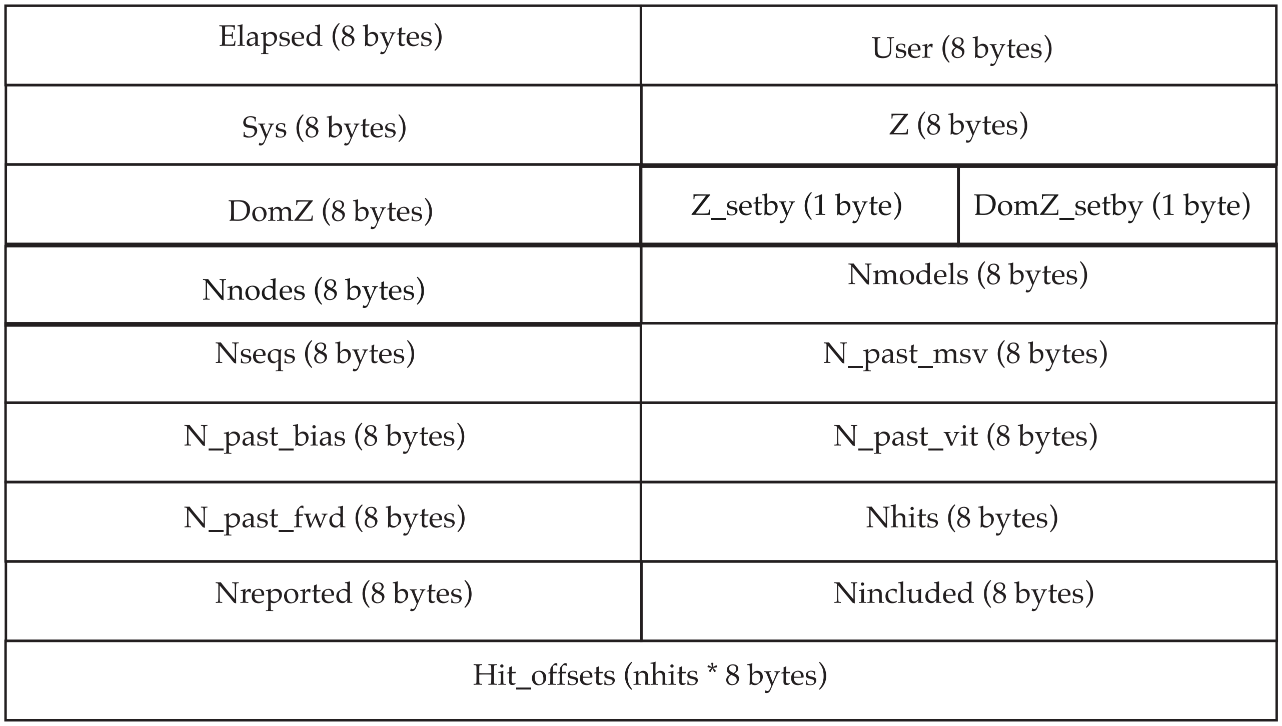
[9] Functions to serialize all relevant data structures into network-order streams of bytes and to de-serialize them into big- or little-endian values depending on the endianness of the hardware HMMER is running on are provided in the HMMER source code.

Status (4 bytes)	Type (4 bytes)
Msg_size (8 bytes)	

Figure 3: Serialized HMMD_SEARCH_STATUS Structure

If an error occurred during the search, the server then sends a string describing the error. If the search completed successfully, it sends a serialized HMMD_SEARCH_STATS structure (Figure 4) followed by *N* serialized P7_HIT structures, one for each hit found by the search. If an error occurred, the msg_size field of the HMMD_SEARCH_STATUS object contains the length of the error description string. Otherwise it contains the sum of the length of the HMMD_SEARCH_STATS structure and the *N* P7_HIT structures, which can be read into a single buffer and un-packed using the deserialization functions HMMER provides.

The HMMD_SEARCH_STATS structure contains information about the search itself. The first three fields (elapsed, user, and sys) contain the elapsed, user, and system time required for the search, in seconds, and are double-precision floating-point numbers. The next



Elapsed (8 bytes)	User (8 bytes)	
Sys (8 bytes)	Z (8 bytes)	
DomZ (8 bytes)	Z_setby (1 byte)	DomZ_setby (1 byte)
Nnodes (8 bytes)	Nmodels (8 bytes)	
Nseqs (8 bytes)	N_past_msv (8 bytes)	
N_past_bias (8 bytes)	N_past_vit (8 bytes)	
N_past_fwd (8 bytes)	Nhits (8 bytes)	
Nreported (8 bytes)	Nincluded (8 bytes)	
Hit_offsets (nhits * 8 bytes)		

Figure 4: Serialized HMMD_SEARCH_STATS Structure

four (Z, domZ, Z_setby, and domZ_setby) contain the number of targets searched and number of significant targets searched, which are used in calculating e-values, as well as enums that describe whether the number of targets was set by the database size or a command-line parameter[10]. The four fields after than (nmodels, nnodes, nseqs, and nres) contain the number of HMMs searched, number of HMM nodes searched, number of sequences searched, and total number of sequence residues searched. The next four fields (n_past_msv, n_past_bias, n_past_vit, n_past_fwd) contain the number of sequence-HMM comparisons that made it past each of the stages of HMMER's comparison pipeline during the search, and the three after that (nhits, nreported, nincluded) contain the number of hits found, the number of hits that met the reporting threshold for the search, and the number of hits that met the threshold to be considered significant. Finally, the hit_offsets array contains the offset (in bytes) from the start of the serialized P7_HIT objects to the start of each P7_HIT objects, if at least one hit was found.

[10] Because the mapping of enums to binary values is not guaranteed to be portable, our serialization and de-serialization routines explicitly convert enums to and from specific integers rather than serializing the actual enums.

P7_HIT objects, shown in Figure 5 are somewhat complex to serialize and de-serialize, as they are variable-length themselves, have optional fields, and contain P7_DOMAIN structures, which contain P7_ALIDISPLAY structures, both of which are variable-length. Because of the number of fields in these data structures, we refer the

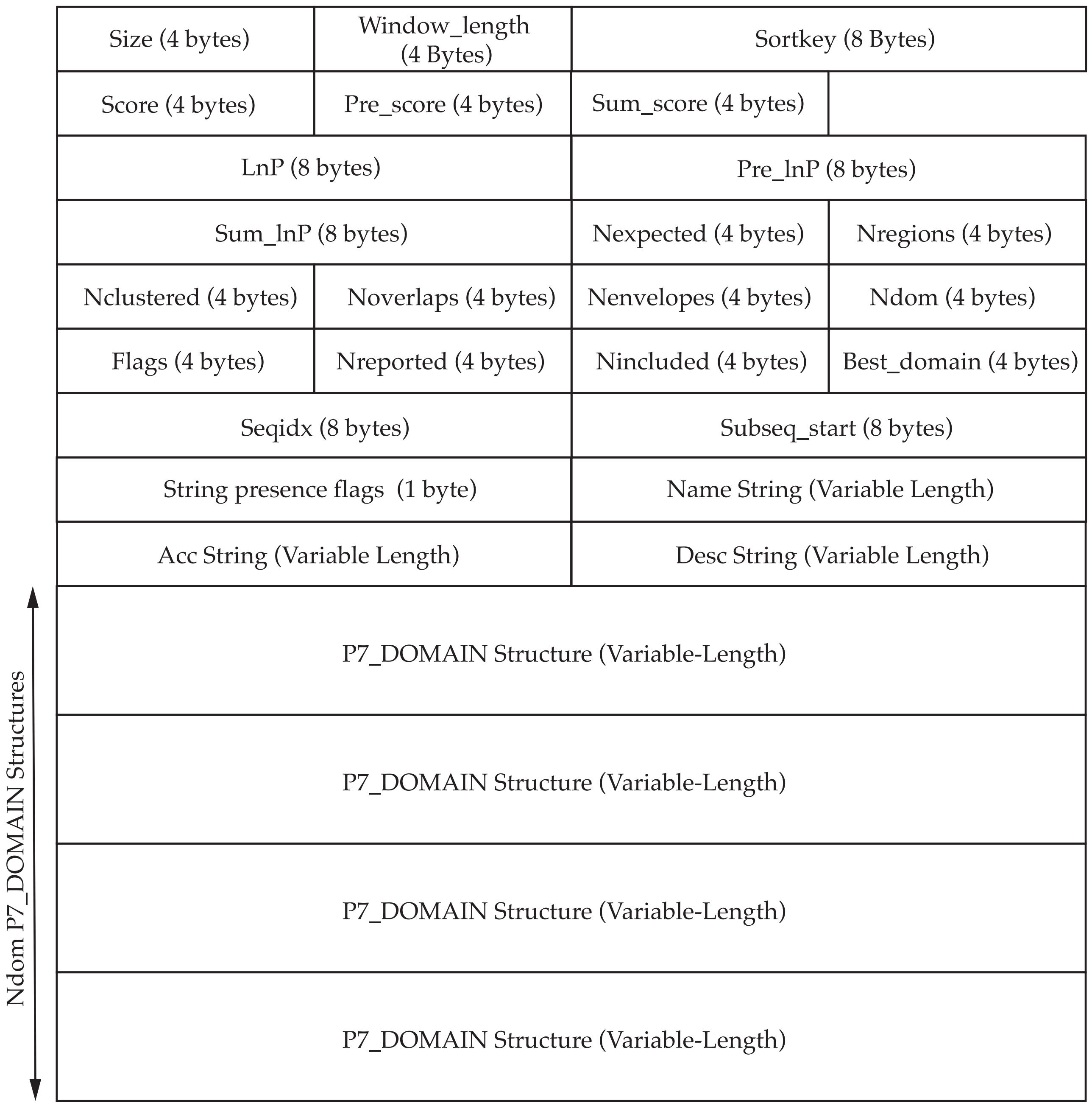


Figure 5: Serialized P7_HIT Structures

reader to the HMMER source code for information about the meaning of each data structure's fields, and only describe here how the serialization of these data structures differs from a straightforward concatenation of the fields in the structure into a network-ordered stream of bytes.

A serialized P7_HIT structure begins with a four-byte size field that is not part of the P7_HIT structure and contains the length (in bytes) of the serialized P7_HIT object. This is followed by all of the fixed-length fields in the data structure, a one-byte field of string presence flags that encode whether the P7_HIT object contains the optional acc and desc strings, and the variable-length name acc, and desc strings (if present) from the P7_HIT object. The serialized object then concludes with *Ndom* serialized P7_DOMAIN structures.

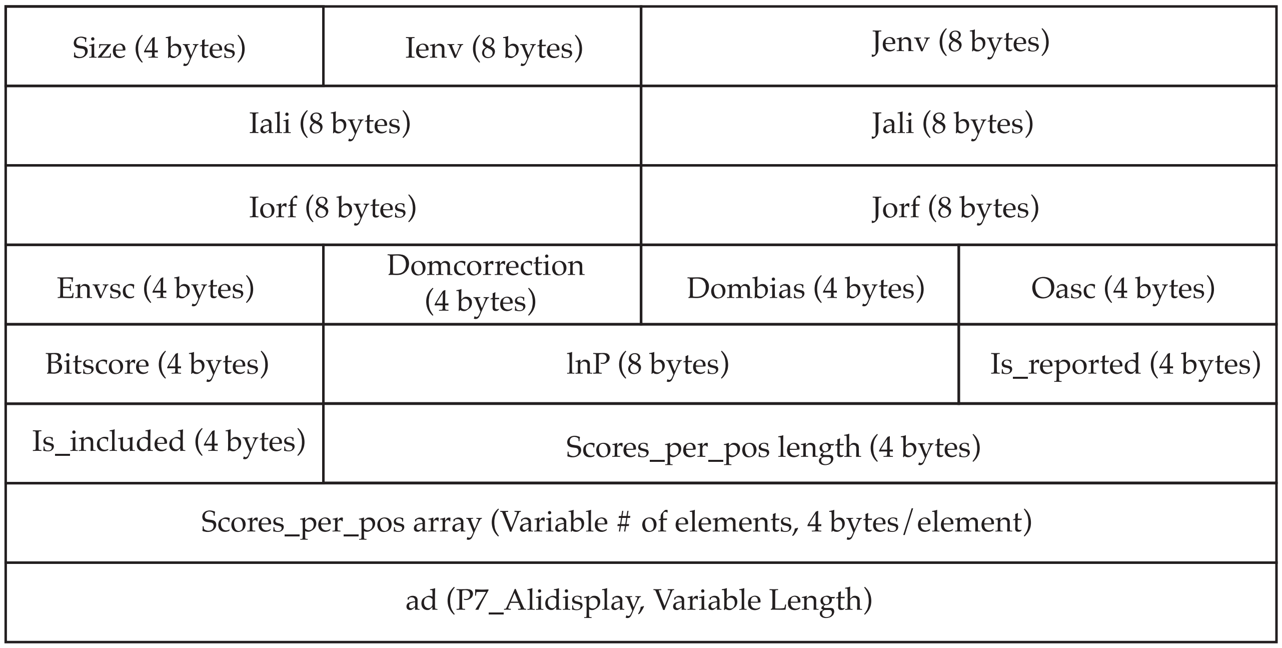


Figure 6: Serialized P7_DOMAIN Structure

The P7_DOMAIN structure (Figure 6) is relatively straightforward to serialize. The serialized structure begins with a size field that encodes its length in bytes, followed by the fixed-length fields of the data structure. The only complexity comes from the scores_per_pos array, which is a variable-length array of floating-point numbers. The scores_per_pos length field[11] gives the length of this array in elements, and is followed by the serialized scores_per_pos array and the domain's P7_ALIDISPLAY structure.

[11] This field is not strictly necessary, as it contains the same information as the n field of the domain's P7_ALIDISPLAY structure, but including it explicitly simplifies de-serialization.

The P7_ALIDISPLAY structure (Figure 7) contains a number of variable-length fields, several of which are optional. The serialized structure begins with a size field containing its length in bytes, followed by the

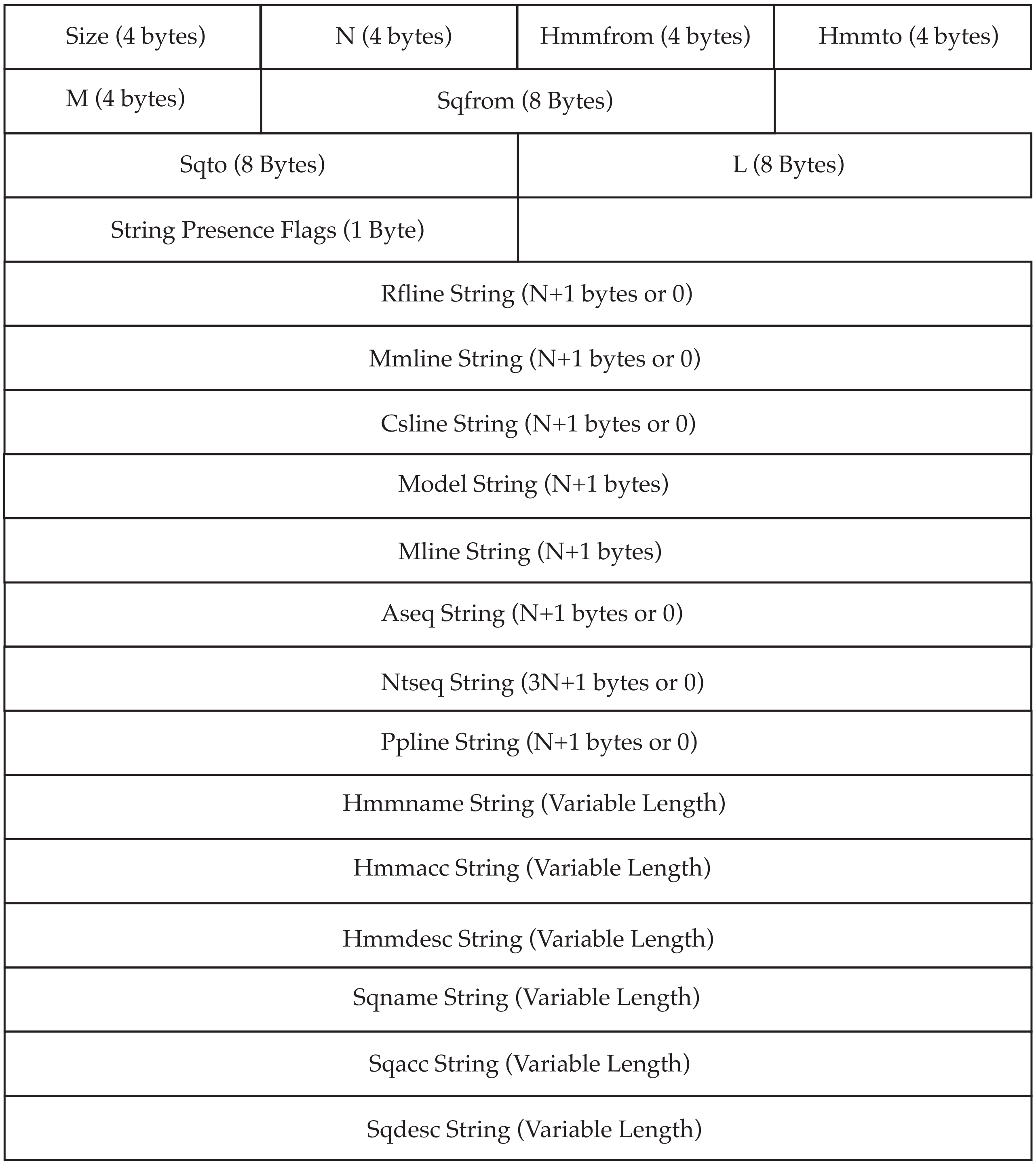

Size (4 bytes)
N (4 bytes)
Hmmfrom (4 bytes)
Hmmto (4 bytes)
M (4 bytes)
Sqfrom (8 Bytes)
Sqto (8 Bytes)
L (8 Bytes)
String Presence Flags (1 Byte)
Rfline String (N+1 bytes or 0)
Mmline String (N+1 bytes or 0)
Csline String (N+1 bytes or 0)
Model String (N+1 bytes)
Mline String (N+1 bytes)
Aseq String (N+1 bytes or 0)
Ntseq String (3N+1 bytes or 0)
Ppline String (N+1 bytes or 0)
Hmmname String (Variable Length)
Hmmacc String (Variable Length)
Hmmdesc String (Variable Length)
Sqname String (Variable Length)
Sqacc String (Variable Length)
Sqdesc String (Variable Length)

Figure 7: Serialized P7_ALIDISPLAY Structure

fixed-length fields of the data structure. A byte of string presence flags encodes the presence or absence of each of the optional field, using a one-hot format. Then, the variable-length fields of the data structure are serialized as byte arrays.

Manual Pages Related to the Server

hmmclient *- submit searches to a server for execution*

Synopsis

`hmmclient` [*`options`*] *`query hmmfile or seqfile`*

Description

`Hmmclient` is a command-line interface to submit searches to a running `hmmserver` It is designed to mimic the behavior and output formats of the `hmmsearch` , `phmmer` , `hmmscan` , and `jackhmmer` programs as closely as possible.

A typical usage of hmmclient is " `hmmclient` -s < *`name of the machine running the server's master node`* > --db < *`number of the database to be searched`* > < *`filename of the query sequence or hmm`* >, which would cause `hmmclient` to send the specified query to the server, wait for results to come back, and then display them in the same manner as our command-line programs. `Hmmclient` accepts the same set of options to control search parameters and format its output as our command-line programs, which are described below.

The query object may be read from standard input. To instruct `hmmclient` to do this, provide a dash ("-") as the last input to `hmmclient` instead of a file name.

Options

`-h` Help; print a brief reminder of command line usage and all available options.

Options that Control the Connection to the Server

`-s` <*servername*> Specifies the name of the machine running the server that `hmmclient` should send the search to.

`--cport` <*portnumber*> Specifies the port on the server that `hmmclient` should connect to. Defaults to 51371.

`--db` <*n*> Specifies the number of the database to be searched, defaults to one.

`--db_ranges` <*rangelist*> Instructs the server to search a subset of the items in the database instead of the entire database (default). *`rangelist`* must be a list of one or more numerical ranges separated by commas, i.e.: start1..end1,start2..end2,etc.

--jack *<maxrounds>* Instructs the server to perform an iterative, jackhmmer-style search, with at most *maxrounds* of iterative search. If this option is specified, both the query object and the target database must be sequences, not HMMs, or an error will occur.

--shutdown Send a shutdown command to the server instead of a search request.

--password *<password>* Specifies a password to be sent along with the shutdown command. May only be used if `--shutdown` is also used.

Options for Controlling Output

-o *<f>* Direct the main human-readable output to a file *<f>* instead of the default stdout.

-A *<f>* Save a multiple alignment of all significant hits (those satisfying *inclusion thresholds*) to the file *<f>*.

--tblout *<f>* Save a simple tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target sequence found.

--domtblout *<f>* Save a simple tabular (space-delimited) file summarizing the per-domain output, with one data line per homologous domain detected in a query sequence for each homologous model.

--pfamtblout *<f>* Save a table of hits and domains to the specified file in Pfam format.

--chkhmm *prefix* When doing an iterative search, at the start of each iteration, checkpoint the query HMM, saving it to a file named *prefix-n*`.hmm` where *n* is the iteration number (from 1..N). Is only valid when --jack is specfied.

--chkali *prefix* When doing an iterative search, at the end of each iteration, checkpoint an alignment of all domains satisfying inclusion thresholds (e.g. what will become the query HMM for the next iteration), saving it to a file named *prefix-n*`.sto` in Stockholm format, where *n* is the iteration number (from 1..N). Is only valid when --jack is specfied.

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but can truncate target profile description lines.

--textw *<n>* Set the main output's line length limit to *<n>* characters per line. The default is 120.

Options Controlling Single Sequence Scoring

These options control how an HMM is generated from a single sequence, either when both the query object and target database are sequences, or in the first round of an iterative search.

--popen *<x>* Set the gap open probability for a single sequence query model to *<x>*. The default is 0.02. *<x>* must be >= 0 and < 0.5.

--pextend *<x>* Set the gap extend probability for a single sequence query model to *<x>*. The default is 0.4. *<x>* must be >= 0 and < 1.0.

--mx *<s>* Obtain residue alignment probabilities from the built-in substitution matrix named *<s>*. Several standard matrices are built-in, and do not need to be read from files. The matrix name *<s>* can be PAM30, PAM70, PAM120, PAM240, BLOSUM45, BLOSUM50, BLOSUM62, BLOSUM80, or BLOSUM90. Only one of the `--mx` and `--mxfile` options may be used.

--mxfile *mxfile* Obtain residue alignment probabilities from the substitution matrix in file *mxfile* on the machine running the server program. The default score matrix is BLOSUM62 (this matrix is internal to HMMER and does not have to be available as a file). The format of a substitution matrix *mxfile* is the standard format accepted by BLAST, FASTA, and other sequence analysis software. See `ftp.ncbi.nlm.nih.gov/blast/matrices/` for example files. (The only exception: we require matrices to be square, so for DNA, use files like NCBI's NUC.4.4, not NUC.4.2.)

Options Controlling Reporting Thresholds

Reporting thresholds control which hits are reported in output files (the main output, `--tblout`, and `--domtblout`). Sequence hits and domain hits are ranked by statistical significance (E-value) and output is generated in two sections called per-target and per-domain output. In per-target output, by default, all sequence hits with an E-value <= 10 are reported. In the per-domain output, for each target that has passed per-target reporting thresholds, all domains satisfying per-domain reporting thresholds are reported. By default, these are domains with conditional E-values of <= 10. The following options allow you to change the default E-value reporting thresholds, or to use bit score thresholds instead.

-E *<x>* In the per-target output, report target sequences with an E-value of <= *<x>*. The default is 10.0, meaning that on average, about 10 false positives will be reported per query, so you can see the top of the noise and decide for yourself if it's really noise.

-T *<x>* Instead of thresholding per-profile output on E-value, instead report target sequences with a bit score of >= *<x>*.

--domE *<x>* In the per-domain output, for target sequences that have already satisfied the per-profile reporting threshold, report individual domains with a conditional E-value of <= *<x>*. The default is 10.0. A conditional E-value means the expected number of additional false positive domains in the smaller search space of those comparisons that already satisfied the per-target reporting threshold (and thus must have at least one homologous domain already).

--domT *<x>* Instead of thresholding per-domain output on E-value, instead report domains with a bit score of >= *<x>*.

Options for Inclusion Thresholds

Inclusion thresholds are stricter than reporting thresholds. Inclusion thresholds control which hits are considered to be reliable enough to be included in an output alignment or a subsequent search round, or marked as significant ("!") as opposed to questionable ("?") in domain output.

--incE *<x>* Use an E-value of <= *<x>* as the per-target inclusion threshold. The default is 0.01, meaning that on average, about 1 false positive would be expected in every 100 searches with different query sequences.

--incT *<x>* Instead of using E-values for setting the inclusion threshold, instead use a bit score of >= *<x>* as the per-target inclusion threshold. By default this option is unset.

--incdomE *<x>* Use a conditional E-value of <= *<x>* as the per-domain inclusion threshold, in targets that have already satisfied the overall per-target inclusion threshold. The default is 0.01.

--incdomT *<x>* Instead of using E-values, use a bit score of >= *<x>* as the per-domain inclusion threshold.

Options for Model-specific Score Thresholding

Curated profile databases may define specific bit score thresholds for each profile, superseding any thresholding based on statistical significance alone. To use these options, the profile must contain the appropriate (GA, TC, and/or NC) optional score threshold annotation; this is picked up by `hmmbuild` from Stockholm format alignment files.

Each thresholding option has two scores: the per-sequence threshold <x1> and the per-domain threshold <x2> These act as if `-T` *<x1>* `--incT` *<x1>* `--domT` *<x2>* `--incdomT` *<x2>* has been applied specifically using each model's curated thresholds.

--cut_ga Use the GA (gathering) bit scores in the model to set per-sequence (GA1) and per-domain (GA2) reporting and inclusion thresholds. GA thresholds are generally considered to be the reliable curated thresholds defining family membership; for example, in Pfam, these thresholds define what gets included in Pfam Full alignments based on searches with Pfam Seed models.

--cut_nc Use the NC (noise cutoff) bit score thresholds in the model to set per-sequence (NC1) and per-domain (NC2) reporting and inclusion thresholds. NC thresholds are generally considered to be the score of the highest-scoring known false positive.

--cut_tc Use the TC (trusted cutoff) bit score thresholds in the model to set per-sequence (TC1) and per-domain (TC2) reporting and inclusion thresholds. TC thresholds are generally considered to be the score of the lowest-scoring known true positive that is above all known false positives.

Options Controlling the Acceleration Pipeline

HMMER3 searches are accelerated in a three-step filter pipeline: the MSV filter, the Viterbi filter, and the Forward filter. The first filter is the fastest and most approximate; the last is the full Forward scoring algorithm. There is also a bias filter step between MSV and Viterbi. Targets that pass all the steps in the acceleration pipeline are then subjected to postprocessing -- domain identification and scoring using the Forward/Backward algorithm. Changing filter thresholds only removes or includes targets from consideration; changing filter thresholds does not alter bit scores, E-values, or alignments, all of which are determined solely in postprocessing.

--max Turn off all filters, including the bias filter, and run full Forward/Backward postprocessing on every target. This increases sensitivity somewhat, at a large cost in speed.

--F1 *<x>* Set the P-value threshold for the MSV filter step. The default is 0.02, meaning that roughly 2% of the highest scoring nonhomologous targets are expected to pass the filter.

--F2 *<x>* Set the P-value threshold for the Viterbi filter step. The default is 0.001.

--F3 *<x>* Set the P-value threshold for the Forward filter step. The default is 1e-5.

--nobias Turn off the bias filter. This increases sensitivity somewhat, but can come at a high cost in speed, especially if the query has biased residue composition (such as a repetitive sequence region, or if it is a membrane protein with large regions of hydrophobicity). Without the bias filter, too many sequences may pass the filter with biased queries, leading to slower than expected performance as the computationally intensive Forward/Backward algorithms shoulder an abnormally heavy load.

Options Controlling Profile Construction

This option is only valid when performing an iterative search.

`--fragthresh` <*x*> We only want to count terminal gaps as deletions if the aligned sequence is known to be full-length, not if it is a fragment (for instance, because only part of it was sequenced). HMMER uses a simple rule to infer fragments: if the sequence length L is less than or equal to a fraction <*x*> times the alignment length in columns, then the sequence is handled as a fragment. The default is 0.5. Setting `--fragthresh 0` will define no (nonempty) sequence as a fragment; you might want to do this if you know you've got a carefully curated alignment of full-length sequences. Setting `--fragthresh 1` will define all sequences as fragments; you might want to do this if you know your alignment is entirely composed of fragments, such as translated short reads in metagenomic shotgun data.

Options Controlling Relative Weights

These options are only valid when performing an iterative search. Whenever a profile is built from a multiple alignment, HMMER uses an ad hoc sequence weighting algorithm to downweight closely related sequences and upweight distantly related ones. This has the effect of making models less biased by uneven phylogenetic representation. These options control which algorithm gets used.

`--wpb` Use the Henikoff position-based sequence weighting scheme [Henikoff and Henikoff, J. Mol. Biol. 243:574, 1994]. This is the default.

`--wgsc` Use the Gerstein/Sonnhammer/Chothia weighting algorithm [Gerstein et al, J. Mol. Biol. 235:1067, 1994].

`--wblosum` Use the same clustering scheme that was used to weight data in calculating BLOSUM substitution matrices [Henikoff and Henikoff, Proc. Natl. Acad. Sci 89:10915, 1992]. Sequences are single-linkage clustered at an identity threshold (default 0.62; see `--wid`) and within each cluster of c sequences, each sequence gets relative weight 1/c.

`--wnone` No relative weights. All sequences are assigned uniform weight.

--wid <*x*> Sets the identity threshold used by single-linkage clustering when using `--wblosum`. Invalid with any other weighting scheme. Default is 0.62.

Options Controlling Effective Sequence Number

After relative weights are determined, they are normalized to sum to a total effective sequence number, *eff_nseq*. This number may be the actual number of sequences in the alignment, but it is almost always smaller than that. The default entropy weighting method (`--eent`) reduces the effective sequence number to reduce the information content (relative entropy, or average expected score on true homologs) per consensus position. The target relative entropy is controlled by a two-parameter function, where the two parameters are settable with `--ere` and `--esigma`.

--eent Adjust effective sequence number to achieve a specific relative entropy per position (see `--ere`). This is the default.

--eentexp Adjust the effective sequence number to reach the relative entropy target using exponential scaling.

--eclust Set effective sequence number to the number of single-linkage clusters at a specific identity threshold (see `--eid`). This option is not recommended; it's for experiments evaluating how much better `--eent` is.

--enone Turn off effective sequence number determination and just use the actual number of sequences. One reason you might want to do this is to try to maximize the relative entropy/position of your model, which may be useful for short models.

--eset <*x*> Explicitly set the effective sequence number for all models to <*x*>.

--ere <*x*> Set the minimum relative entropy/position target to <*x*>. Requires `--eent`. Default depends on the sequence alphabet; for protein sequences, it is 0.59 bits/position.

--esigma <*x*> Sets the minimum relative entropy contributed by an entire model alignment, over its whole length. This has the effect of making short models have higher relative entropy per position than `--ere` alone would give. The default is 45.0 bits.

--eid <*x*> Sets the fractional pairwise identity cutoff used by single linkage clustering with the `--eclust` option. The default is 0.62.

Options Controlling Priors

These options are only valid when performing an iterative search or when both the query object and target database are sequences. In profile construction, by default, weighted counts are converted to mean posterior probability parameter estimates using mixture Dirichlet priors. Default mixture Dirichlet prior parameters for protein models and for nucleic acid (RNA and DNA) models are built in. The following options allow you to override the default priors.

`--pnone` Don't use any priors. Probability parameters will simply be the observed frequencies, after relative sequence weighting.

`--plaplace` Use a Laplace +1 prior in place of the default mixture Dirichlet prior.

Options Controlling E-value Calibration

These options are only valid when performing an iterative search. Estimating the location parameters for the expected score distributions for MSV filter scores, Viterbi filter scores, and Forward scores requires three short random sequence simulations.

`--EmL` *<n>* Sets the sequence length in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

`--EmN` *<n>* Sets the number of sequences in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

`--EvL` *<n>* Sets the sequence length in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

`--EvN` *<n>* Sets the number of sequences in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

`--EfL` *<n>* Sets the sequence length in simulation that estimates the location parameter tau for Forward E-values. Default is 100.

`--EfN` *<n>* Sets the number of sequences in simulation that estimates the location parameter tau for Forward E-values. Default is 200.

`--Eft` *<x>* Sets the tail mass fraction to fit in the simulation that estimates the location parameter tau for Forward evalues. Default is 0.04.

Other Options

`--nonull2` Turn off the null2 score corrections for biased composition.

`-Z` *<x>* Assert that the total number of targets in your searches is *<x>*, for the purposes of per-sequence E-value calculations, rather than the actual number of targets seen.

`--domZ` *<x>* Assert that the total number of targets in your searches is *<x>*, for the purposes of per-domain conditional E-value calculations, rather than the number of targets that passed the reporting thresholds.

`--seed` *<n>* Set the random number seed to *<n>*. Some steps in post-processing require Monte Carlo simulation. The default is to use a fixed seed (42), so that results are exactly reproducible. Any other positive integer will give different (but also reproducible) results. A choice of 0 uses a randomly chosen seed.

`hmmserver` - *server that accelerates homology searches*

Synopsis

`hmmserver` [*`options`*] *`list of data files to be read into memory`*

Description

The `hmmserver` command starts a service that accepts search requests from clients and distributes the work of searches across one or more computers to improve performance. Once started, the server listens on the specified port for search requests, processes them as they arrive, and returns the results to clients. The server userguide (documentation/userguide/Server_Userguide.pdf) describes the format that `hmmserver` expects for search requests and replies.

`Hmmserver` must be run as an MPI (message-passing interface) process with at least two ranks, one for the master process and one or more worker processes. See your local system's documentation for details on how to start an MPI process, as the details vary depending on cluster configuration and the implementation of MPI used.

Options

`-h` Help; print a brief reminder of command line usage and all available options.

`--cport` <*n*> Port to use for communication between clients and the master node of the server. The default is 51371.

`--ccncts` <*n*> Maximum number of client connections to accept. The default is 16.

`--num_dbs` <*n*> Number of database files to read into RAM. Must match the number of data files passed as arguments to `hmmserver` , and defaults to one.

`--num_shards` <*n*> Number of shards (pieces) to divide each database into to reduce memory usage. Defaults to one, and must be less than the number of ranks in the MPI process to allow at least one worker rank per shard, plus one rank for the master node. In general, having fewer shards will increase performance by allowing better load-balancing at the cost of making the server use more memory on each worker node. Specifying a number of shards that is not an integer divisor of the number of worker ranks will lead to suboptimal performance as some shards will have fewer worker nodes operating on them than others.

`--cpu` <*n*> Number of worker threads to start on each worker node. If 0 (the default), queries the system's hardware to determine how many threads it can support simultaneously, and starts that many minus one worker threads, to leave one thread for the worker node's master thread. The master node always starts a fixed (small) number of threads, independent of the value passed via this flag.

`--stall` Instructs the server to stall immediately after startup so that a debugger can be attached to each node. When this option is set, the server sets the variable "stalling" to TRUE and enters a loop that iterates until stalling is set to false by the debugger.

`--password` *<password>* Specifies a password that a client must send along with the shutdown command in order to shut down the server.

Acknowledgments

Simon Potter of the European Bioinformatics Institute was of great help in understanding the daemon's interactions with the EBI's web servers. We would also like to thank all of the organizations that have supported the development of HMMER, as well as all of the individuals who have contributed to it. In particular, Washington University, the National Institutes of Health, Monsanto, the Howard Hughes Medical Institute, and Harvard University have been major supporters of this work. For a more thorough set of acknowledgments that includes a discussion of HMMER's history, please see the HMMER User's Guide.