User's Guide for the HMMER Daemon

High-performance biological sequence analysis using profile hidden Markov models

Sean R. Eddy, Nicholas P. Carter
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

The performance of HMMER's command-line tools is limited by two factors: the number of cores that are available to work on each search, and the amount of time required to read sequence or HMM databases from disk in order to search them. When HMMER is run on small (2- or 4-core) desktop or laptop CPUs, CPU performance tends to be the limiting factor on search time (although this depends on the speed of the system's hard disk). However, on systems with more cores, such as most servers, the time required to read databases from disk becomes a bottleneck that prevents performance from improving as processor performance increases.

To address this, HMMER provides a daemon[1], known as `hmmpgmd` that runs as a persistent service across multiple computers. As shown in Figure 1, one machine running the daemon is designated the master node, while the others become worker nodes. Client systems send search requests to the master node via internet sockets. When the master node receives a search, it distributes the work of the search across the worker nodes, merges the partial results from each worker node, and returns the search results to the client[2].

[1] The term "daemon" here is taken from the UNIX term for an application or service that runs without being connected to a user login, which is derived from a Greek term that means roughly "spirit" or "supernatural being".

[2] In the European Bioinformatics Institute's (EBI) instantiation of the daemon, the client machines shown in the figure are EBI's web servers. End users interact with the web servers, which provide a human-friendly interface for submitting searches and inspecting results.

The daemon improves search performance over HMMER's command-line tools in two ways. First, when the daemon starts up, it reads its input databases from disk once and caches them in RAM, eliminating the file read bottleneck on performance. Second, it distributes the work of each search over many machines, each of which may contain tens of cores, significantly reducing the amount of work each core must perform to complete the search. In combination, these techniques can reduce database search times to only a few seconds when the daemon is run on a large-enough set of computers.

`Hmmpgmd` caches all of its database(s) in RAM on each of its worker nodes, in spite of the fact that each worker node only accesses part of a database on each search. This allows `hmmpgmd` to tolerate worker node failures by simply changing how work is assigned to the worker nodes as the number of worker nodes changes, but causes `hmmpgmd` to use significantly more memory than would be strictly necessary, particularly when the daemon has many worker nodes. To address this, HMMER

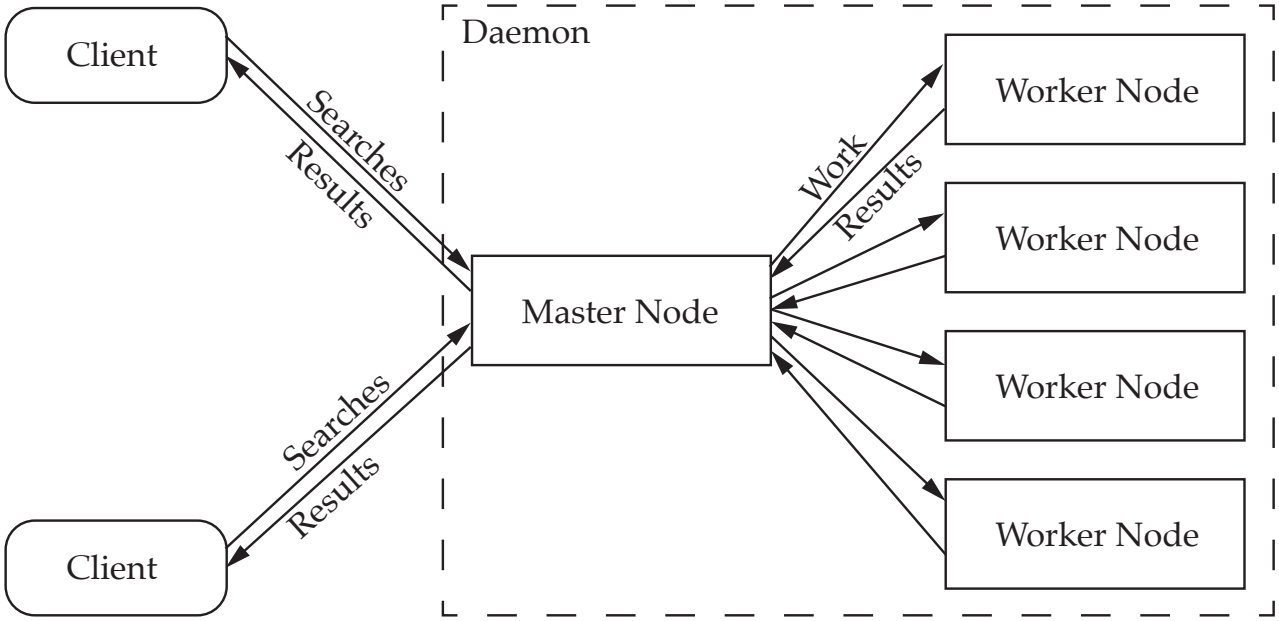

Figure 1: The HMMER Daemon

now provides a "sharded" version of `hmmpgmd`, `hmmpgmd_shard`, which distributes the sequences in a sequence database[3] round-robin across the worker nodes so that each worker node only caches $\frac{1}{Nth}$ of each sequence database. This allows systems with limited RAM to support larger sequence databases, at the cost of making the daemon unable to execute searches unless all of its worker nodes are available.

`Hmmpgmd` was designed to be integrated with the European Bioinformatics Institute's computing and database infrastructure. To reduce RAM and bandwidth usage, sequence names, accession information, and descriptions are not included in `hmmpgmd's` databases and are not cached in RAM. Instead, each sequence or HMM is assigned an ID number based on its position in the database file, and `hmmpgmd` reports that ID number when a hit occurs. EBI's web servers then look that ID number up in their own databases to retrieve the sequence/HMM's name, description, etc., and combine that with the alignment information the daemon returns when displaying search results to users.

This manual outlines `hmmpgmd`'s usage and interface to client applications, and assumes that the reader is familiar with the concepts covered in the main HMMER User's Guide. It is intended for individuals who are interested in either running an `hmmpgmd` server of their own or in writing clients that communicate with an existing `hmmpgmd` server. Biologists who wish to use an `hmmpgmd` server in their research without the complexity of configuring one themselves should consider using the European Bioinformatics Institute's HMMER server[4], which pro-

[3] We do not support sharding of HMM databases because current HMM databases are so much smaller than current sequence databases that the amount of RAM they occupy has not been an issue.

[4] www.ebi.ac.uk/Tools/hmmer/

vides a web interface to `hmmpgmd` servers that load a number of common genetic databases.

Usage

To perform searches using `hmmpgmd`, a user must first start a daemon on one or more machines. At least two `hmmpgmd` processes, one master and one worker, must be started, although these processes may be run on the same computer if only one machine is available[5]. Once the daemon is running, users can submit searches via a client program such as the `hmc2` example client HMMER provides.

Running Hmmpgmd

To start a daemon process, execute the "`hmmpgmd`" command. `Hmmpgmd` requires that the user provide either the `--master` flag, which indicates that the process should run as the master node of a daemon, or the `--worker <IP of master node>`, which indicates that `hmmpgmd` should run as a worker node with its master node at the specified IP address[6]. Note that this IP address must be provided as a numeric value, for example "127.0.0.1", and not as a text machine name, as `hmmpgmd` does not perform domain name service lookup on the provided IP address.

When run as a master node, `hmmpgmd` requires that the user provide at least one of the `--seqdb <filename>`, or `--hmmdb <filename>` options to specify a sequence and/or HMM database for the daemon to cache in RAM. The user may specify both a sequence and an HMM database file that the daemon should cache. The master node reads the database(s) itself and also sends the specified filename(s) to each of the worker nodes so that they can cache their own copies of the database(s). Thus, the database file(s) must be available on all of the nodes in the daemon and the filename(s) provided must be valid paths to the database file(s) on all of the nodes, either because `hmmpgmd` is invoked from the same directory on every node or because the filename(s) are specified as full paths from the root node of the filesystem.

When run as a master node, `hmmpgmd` first reads its input databases and then listens for socket connections from worker nodes. When a worker node connects to the master, the master outputs "Handling worker <IP address (socket number)"" to its display or logfile. After the worker has read its databases and is ready to handle work, the

[5] When running `hmmpgmd` on a single machine, be aware that the master and worker processes will load the database(s) into RAM independently, so the memory required will be approximately double what is required when running the worker and master processes on separate machines. `Hmmpgmd_shard` greatly reduces the amount of memory used by the master node, so using it with only one shard may be a better choice for searching sequence databases on a single machine.

[6] If the master node's IP is omitted, `hmmpgmd` defaults to connecting to a master process on the same node it is running on.

master outputs "Pending worker <IP address> (socket number)". The master can begin accepting search requests from clients as soon as one worker node has reached the pending state, and will distribute each search across all of the worker nodes that are pending when the search begins.

Running `hmmpgmd_shard`

Starting a sharded daemon using the `hmmpgmd_shard` program follows the same procedure as starting an undsharded daemon, with two exceptions. First, when run on the master node `hmmpgmd_shard` takes a mandatory `-num_shards <n>` argument, which specifies the number of shards that the sequence database file should be broken into. This must be equal to the number of worker nodes that will connect to the master. Second, because each worker node only contains a portion of its sequence database(s), `hmmpgmd_shard` can only process search requests when the number of worker nodes connected to it is equal to the number of shards. Attempting to run searches before that many worker nodes have connected will return an error. Attempting to connect more worker nodes than the specified number of shards causes the daemon to exit.

Note that `hmmpgmd_shard` only shards sequence databases. HMM databases are not sharded, due to their small size, so there is no reason to use `hmmpgmd_shard` on HMM databases.

Sending Searches to a Daemon

The HMMER package includes *hmmc2*, an example client that can communicate with an `hmmpgmd` or `hmmpgmd_shard` daemon. Hmmc2 accepts four command-line arguments[7]:

`-i <IP address>` specifies the IP address of the daemon to connect to. Defaults to 127.0.0.1 if not provided.

`-p <port>` specifies the port number that the daemon is listening to for client connections. Defaults to 51371 (the `hmmpgmd` default) if not provided.

`-S` print the scores of any hits found by searches

`-A` print the alignments of any hits found by searches. The data printed when this option is provided is a superset of the data printed when "-S" is provided, so the "-S" option is redundant if -A is provided.

Once it starts up, `hmmc2` enters an interactive loop by printing the prompt "Enter next sequence:". Any text the user enters between that

[7] Hmmc2's code breaks Easel conventions by manually parsing its command-line arguments instead of using Easel's `getopts` functions. It also defines a separate Easel `getopts` structure, `searchOpts`, which it uses to parse search requests, which can easily be mistaken for a definition of `hmmc2's` command-line options.

prompt and the end-of-command string "`//`" is interpreted as a command and sent to the daemon.

Usage Example

After starting `hmmc2` with the command

```
hmmc2 -i <IP of daemon's master node>
```

it will print the prompt

```
Enter next sequence:
```

To run a simple search, enter the text

```
@--seqdb 1
>sp|Q6GZW9|length_10
YLGPWVQAEY
//
```

The text **`@--seqdb 1`** on the first line instructs the daemon to perform a search against sequence database 1. The next two lines (**`>sp|Q6GZW9|length_10`** and **`YLGPWVQAEY`**) are the FASTA-format specification of a (short) amino acid sequence that the daemon will translate into an HMM and search against the database. Finally, the `//` by itself on the third line ends the command.

Once it sees the end-of-command sequence, `hmmc2` will send the command to the server, which will perform the search and generate output similar to

```
Sending data 46:
Internal pipeline statistics summary:
-------------------------------------
Query model(s):                            1  (0 nodes)
Target sequences:                     556825  (0 residues searched)
Passed MSV filter:                     10903  (0.0195807); expected 11136.5 (0.02)
Passed bias filter:                     8073  (0.0144983); expected 11136.5 (0.02)
Passed Vit filter:                       220  (0.000395097); expected 556.8 (0.001)
Passed Fwd filter:                         1  (1.7959e-06); expected 5.6 (1e-05)
Initial search space (Z):             556825  [actual number of targets]
Domain search space  (domZ):               1  [number of targets reported over threshold]
# CPU time: 0.01u 0.00s 00:00:00.01 Elapsed: 00:00:00.54
# Mc/sec: 0.00
//
Total bytes received 968


Enter next sequence:
```

although the exact output generated will vary depending on the database that the daemon has cached. This output is a simple summary that shows the number of comparisons performed by the daemon, the number of hits found, and the time it took to perform the search. If the `-S` or `-A` options had been used when starting `hmmc2`, substantially more information about the hits found by the server would have been displayed.

The next chapter provides significantly more detail about the format of the commands the daemon accepts and of the output it sends back to the client.

Daemon-Client Interface

Client machines use internet sockets to send commands to and receive results from a daemon's master node. When a client opens a connection to the master node's client communication port (port 51371 by default), the master node forks a thread to manage the connection with the client. This thread configures a socket to communicate with the client and then repeatedly calls the `clientside_loop`[8] function, which monitors the socket for commands from the client, until either the client detaches from the port or the daemon shuts down. This approach allows multiple clients to connect to a daemon simultaneously without interfering with each other, although requests from one client may impact the amount of time it takes for the daemon to respond to requests from other clients.

[8] This function name is a bit of a misnomer, in that it does not contain a loop. Instead, it is repeatedly called from within an outer loop.

Daemon Command Format

Daemon commands are variable-length sequences of ASCII text. The first line of a command must contain the command itself and any options or parameters. For search commands, this is followed by one or more lines that contain the sequence or HMM to be searched. All commands end with a line that contains only two forward slashes ("`//`"). When a command arrives from a client, the daemon reads bytes from the appropriate socket into a buffer until it sees the end-of-command sequence, growing the buffer as necessary[9], and then parses the contents of the buffer in order to execute the command.

[9] This is a security vulnerability that should be addressed in HMMER4, as it allows an adversarial or erroneous client to consume arbitrary amounts of RAM, potentially exceeding the capacity of the master node.

The daemon supports three commands:

`@--hmmdb <database #>` Initiates a search of a protein sequence against the HMM database cached by the daemon. The protein sequence to be searched must be provided on the lines following the `@--hmmdb` command. Note that the user is required to provide a database number argument to `@--hmmdb`, but `hmmpgmd` can only load one HMM database at a time and ignores the value provided. This is a known idiosyncrasy that has been left unchanged to avoid breaking EBI's tools and web interface code.

`@--seqdb <database #> [--seqdb_ranges <rangelist>]` Initiates a search of a protein sequence or HMM against the specified protein sequence database[10]. The daemon determines whether a sequence or HMM has been submitted by examining the contents of the lines that follow the command, and, if a sequence has been submitted, converts it to an HMM before performing the search.

If the `--seqdb_ranges` option is not provided, the entire target database is searched[11]. If the `--seqdb_ranges` option is provided, it must be followed by a range list describing the set of sequences to be searched. Each range in the range list should be formatted in the form "start..end", where "start" and "end" are the sequence IDs of the start and end of the range, and ranges in the list should be separated by commas. One note here is that the sequences in a sequence file are indexed as a single contiguous list, even if the file contains multiple databases, and each database can contain an arbitrary subset of the sequences in the file. Thus, the sequence IDs specified in a range list refer to positions within the database file, and a range list search searches the sequences in the specified database whose IDs fall into the specified range(s), not the specified positions in the set of sequences contained in the database. For example: the command **`@--seqdb 2 --seqdb_ranges 1..100, 201..300`** searches the sequences in database 2 whose sequence ID's range from 1 to 100 or 201 to 300, not sequences 1-100 and 201-300 of the database.

`!shutdown` Shuts the daemon down in an orderly fashion by first sending shutdown messages to all of its worker nodes and then exiting the master node's processes[12].

When the daemon receives a search command, any text on the command line after the `@--seqdb <database #>` or `@--hmmdb <database #>` specifies options to the search, using the same format as the `hmmsearch` or `hmmscan` commands. Thus sending the command **`@--seqdb 1 -E 20`** to the daemon instructs it to perform a search of sequence database 1, reporting all results with an e-value of less than 20 instead of the default 10.

[10] Note that there is an inconsistency in how databases are numbered in search commands as compared to how they are numbered in the database file itself. The database file uses 0-indexed numbering, (databases are numbered from 0 to N-1), while the search commands use 1-indexed numbering (databases are numbered from 1 to N)

[11] Currently, there is no way to search only a portion of an HMM database. This is probably because existing HMM databases are small enough that the time to search them is rarely an issue.

[12] There's another security vulnerability here, in that any machine that can connect to the master node can shut it down. This needs to be addressed in H4, as we intend to allow arbitrary clients to send searches to a server

Search Results Format

The results from each search are split across two sockets messages, as shown in Figure 2. The first is a fixed-length `HMMD_SEARCH_STATUS` structure that contains two fields: a `status` field that contains an Easel status code that tells the client whether the search completed successfully or not, and a `msg_size` field, which tells the client how large (in bytes) the second message will be. The format of the second message depends

on whether any errors were encountered during the search. If an error occurred, the second message is simply a text string containing a description of the error.

If the search succeeded, the `status` field of the first message is set to `eslOK` and the second message contains the results of the search. This message begins with a `HMMD_SEARCH_STATS` structure that contains information about the search, including the time the search took and the number of hits found. This is then followed by a `P7_HIT` structure for each hit the search found, which describes the hit.

Serialization

Data structures and multi-byte values must be *serialized* before they can be sent over sockets. There are two aspects to serializing the types of data structures the daemon uses. Structures that contain pointers must be *flattened* by copying the data that their pointers point to into the block of data that will be sent over the socket, and multi-byte values must be converted to "network order," which is defined as big-endian, to prevent problems if the sending and receiving machines have different endiannesses. To serialize floating-point numbers, we assume that all computers use IEEE 754 representations and just convert the bytes that represent the floating-point number to and from network byte order. This is faster than the alternative approach of representing the floating-point number as an ASCII string and parsing it on the receiving side, and avoids any loss of precision, but will fail if the machines trying to communicate via sockets use different floating-point formats.

Figures 3–6 show how the data structures used in sending search results back to clients are serialized. The text streams the client uses to send commands to the daemon do not require serialization, as text streams do not contain pointers and are sequences of byte-wide characters that are not affected by endianness.

All of the data structures that hmmpgmd serializes contain variable-length fields, and many of them contain optional fields that may or may not be present in a given object. We handle optional fields by adding a byte of presence bit flags, one flag per optional field, to the serialized versions of data structures that contain optional fields. If the flag bit corresponding to a field is set, then that field is present in the data structure. Otherwise it is not present. None of the data structures hmmpgmd serializes have more than eight optional fields, so only one byte of presence flags/structure is required.

Most of the variable-length fields in our data structures are strings, which we represent as standard C-terminated strings. Routines such as `strcpy` are used to copy these strings into and out of the serialized data structures, making it unnecessary to encode the strings' length

First Message:

Return Code (4 Bytes)	Length (in Bytes) of Second Message (8 Bytes)

Second Message:

If Error(s) Occur:

String Describing the Error (Variable-Length)

If Search Successful:

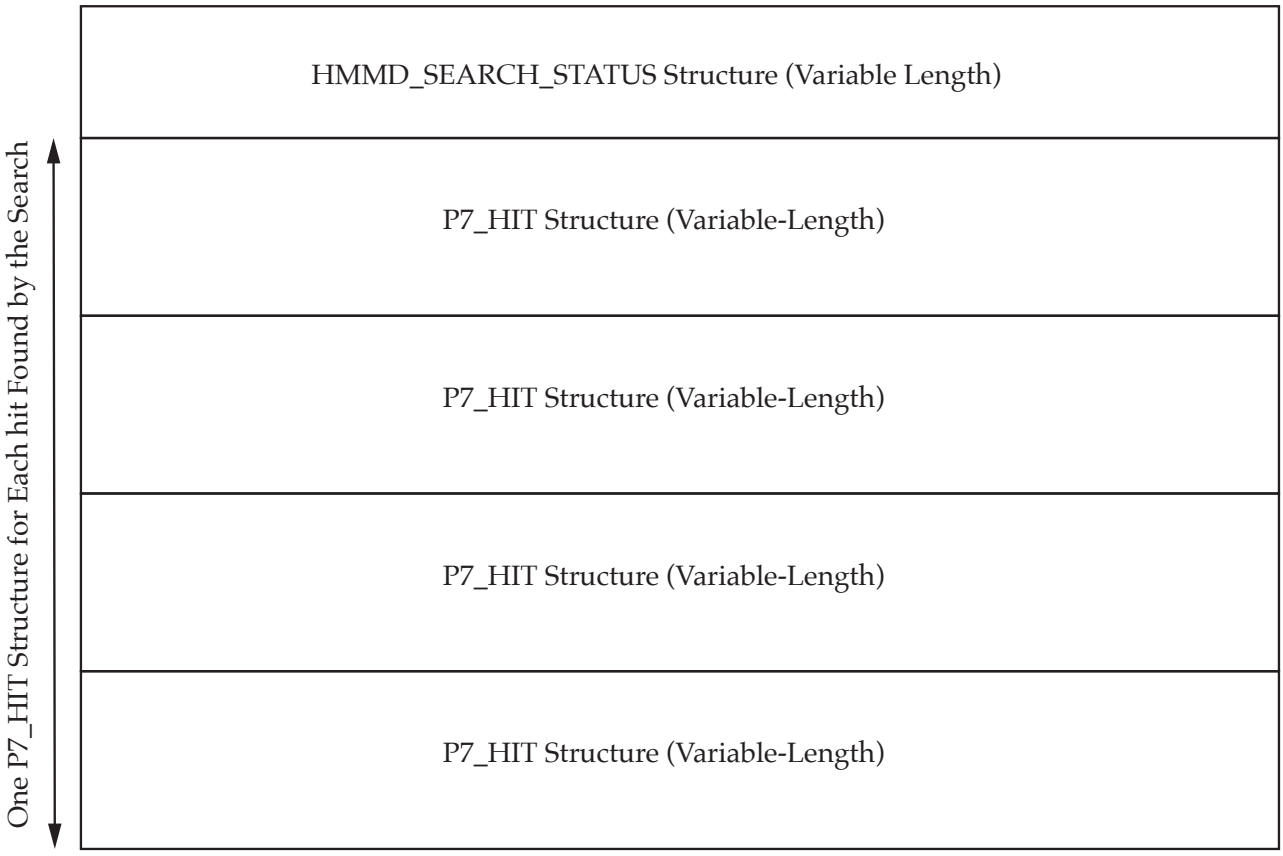

Figure 2: Format of the Messages the Daemon Sends to a Client When a Search Completes

Elapsed (8 bytes)		
User (8 bytes)		Sys (8 bytes)
Z (8 bytes)		DomZ (8 bytes)
Z_setby (1 byte)	DomZ_setby (1 byte)	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 3: Serialized HMMD_SEARCH_STATS Structure

in the serialized data structure. Some non-string fields, such as the `hit_offsets` field in the `HMMD_SEARCH_STATS` structure, have lengths that are determined by the value of some other field in the data structure, so it is not necessary to explicitly encode their lengths in the serialized data structures. One field, the `scores_per_pos` array in the `P7_DOMAIN` structure, has a length that cannot be determined from the other data in the data structure, so we add an explicit field to the serialized data structure to encode its length.

Because it is sometimes necessary to manipulate serialized data without deserializing the full set of results, we include two mechanisms to help locate sub-fields of the daemon's results. The `HMMD_SEARCH_STATS` structure contains a `hit_offsets` array that contains the offsets from the start of the block of serialized hits to the beginning of each serialized hit. Also, all of our serialized data structures except `HMMD_SEARCH_STATS` begin with a `size` field that contains the length of the base (i.e., without any included sub-structures) serialized data structure in bytes. For example, the `size` field of a `P7_HIT` structure will contain the length of the serialized `P7_HIT` structure not including the lengths of the `P7_DOMAIN` structures that the `P7_HIT` structure contains. These features allow software to quickly locate a specific `P7_HIT` structure within the array of results after decoding the `HMMD_SEARCH_STATS` structure, and then to skip through the `P7_HIT` structure as necessary to locate its sub-fields.

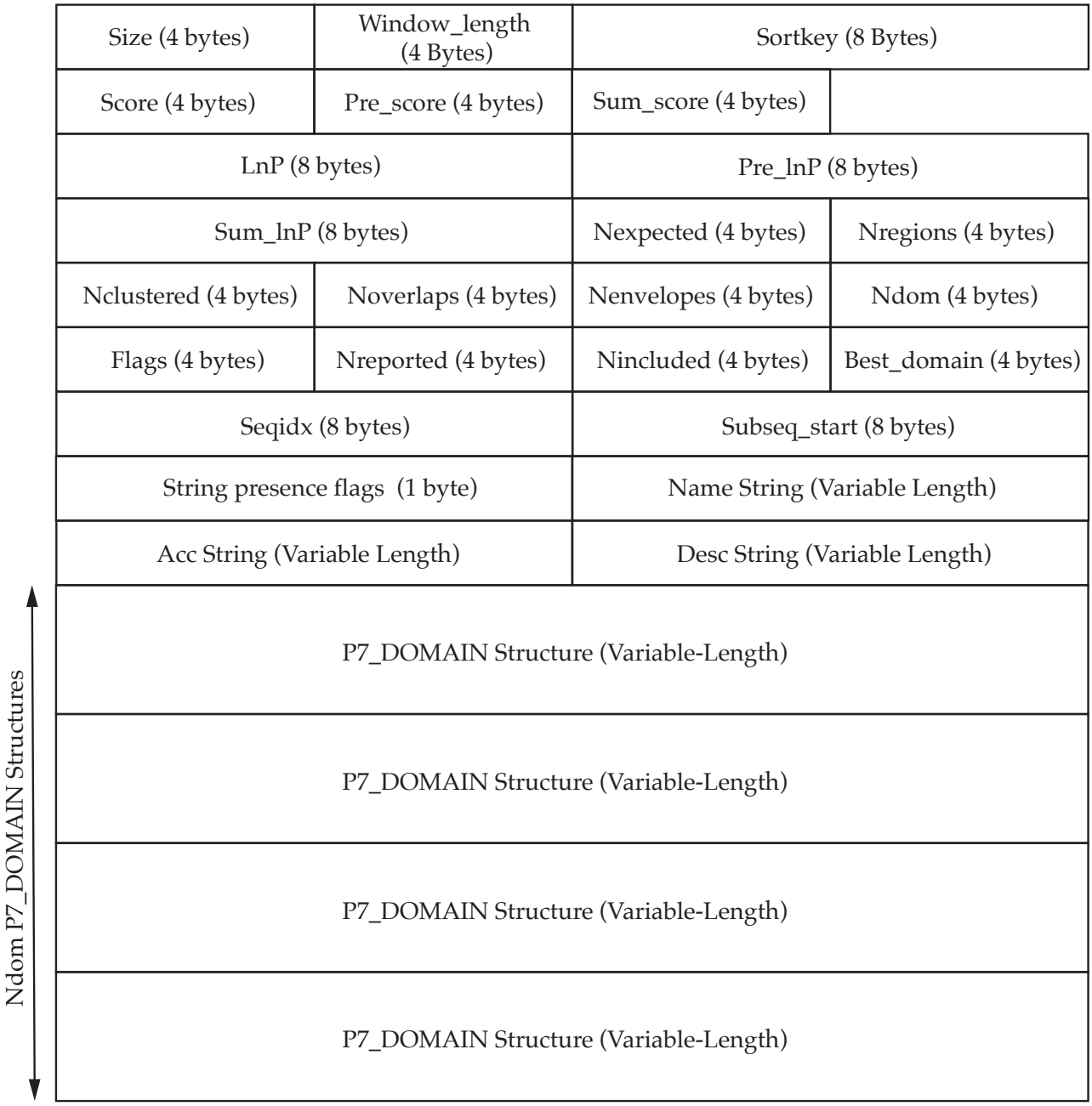

Figure 4: Serialized P7_HIT Structures

<table>
<tr><td>Size (4 bytes)</td><td>Ienv (8 bytes)</td><td colspan="2">Jenv (8 bytes)</td></tr>
<tr><td colspan="2">Iali (8 bytes)</td><td colspan="2">Jali (8 bytes)</td></tr>
<tr><td colspan="2">Iorf (8 bytes)</td><td colspan="2">Jorf (8 bytes)</td></tr>
<tr><td>Envsc (4 bytes)</td><td>Domcorrection (4 bytes)</td><td>Dombias (4 bytes)</td><td>Oasc (4 bytes)</td></tr>
<tr><td>Bitscore (4 bytes)</td><td colspan="2">lnP (8 bytes)</td><td>Is_reported (4 bytes)</td></tr>
<tr><td>Is_included (4 bytes)</td><td colspan="3">Scores_per_pos length (4 bytes)</td></tr>
<tr><td colspan="4">Scores_per_pos array (Variable # of elements, 4 bytes/element)</td></tr>
<tr><td colspan="4">ad (P7_Alidisplay, Variable Length)</td></tr>
</table>

Figure 5: Serialized P7_DOMAIN Structure

Functions to serialize and deserialize each of the daemon's data structures are provided in the .c files that contain the structure's routines.

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 6: Serialized P7_ALIDISPLAY Structure

Database File Formats

The formats hmmpgmd uses for HMM and sequence databases are derived from the formats used elsewhere in HMMER, with modifications to support integration with EBI's infrastructure and reduce the size of sequence databases.

HMM Database File Format

hmmpgmd uses the HMMER 3 profile HMM file format as described in the HMMER User's Guide for its HMM databases, with the requirement that the HMM file must have been processed by hmmpress before it can be loaded into hmmpgmd. When an HMM database is cached in RAM, it is represented as a simple array of P7_OPROFILE objects, although each HMM's name is replaced with its ID number (position in the database file). Each HMM file can only contain a single HMM database, and hmmpgmd does not support caching of multiple HMM databases simultaneously.

Sequence Database File Format

The hmmpgmd format that hmmpgmd uses for sequence database files is a variant of the FASTA format in which sequence names are replaced with numeric IDs, sequence descriptions and accession information are removed, and new sequence descriptions are added that enable efficient caching of multiple databases simultaneously and assist EBI's web servers. A FASTA file can be converted into a single-database hmmpgmd sequence file using the esl-reformat command. For example,

```
esl-reformat -id_map mydb.hmmpgmd.map hmmpgmd mydb.fasta > mydb.hmmpgmd
```

generates the hmmpgmd file "mydb.hmmpgmd[13]" from the FASTA file "mydb.fasta", using the optional -id_map flag to specify that the mapping between the sequence names and descriptions found in the FASTA file to the sequence ID numbers used in the hmmpgmd file should be written to "mydb.hmmpgmd.map". HMMER does not provide a tool to generate sequence files that contain multiple databases. The EBI uses

[13] The ".hmmpgmd" extension used here is only a convention. hmmpgmd does not enforce any restrictions on the names of its input files.

a tool flow of their own development to generate the multi-database files that they use in their HMMER server.

An hmmpgmd file begins with a header line that contains information about the file's contents, which takes the form

#res_cnt seq_cnt db_cnt cnt_1 fullcnt_1 cnt_2 fullcnt_2 ... date_stamp.

The "#" character at the start of the header line is mandatory, and allows HMMER/Easel's file parsing code to recognize the file as an hmmpgmd file instead of a standard FASTA file. The fields in the header line have the following meanings:

res_cnt Number of residues in the sequence file.

seq_cnt Number of sequences in the sequence file.

db_cnt Number of databases in the sequence file.

cnt_i The number of sequences in database i. This will always be less than or equal to seq_cnt.

fullcnt_i The number of sequences that should be used when computing E-values for database i. This will always be at least as large as cnt_i, and may be larger if database i contained redundant sequences that were collapsed out.

date_stamp The day and time when the file was created.

After the header line, the file must contain **seq_cnt** sequence entries, which use a FASTA-like format where the first line of each sequence entry takes the form

>seq_id database_membership domain_architecture taxonomy_id

and the second and following lines contain the amino-acid specification of the sequence[14]

[14] hmmpgmd only supports caching of amino-acid sequences due to the large size of nucleotide sequence databases.

The fields in a sequence's first line have the following meanings:

seq_id The sequence's ID number, which must be its position in the file, where the first sequence in the file has ID 1. This requirement exists because hmmpgmd generates a string encoding of each sequence's ID by incrementing an internal counter for each sequence and stores that string encoding in the sequence structure's name field. Thus, the sequence ID hmmpgmd caches will be the sequence's position in the database file regardless of the value of seq_id. Note that the IDs used by hmmpgmd are absolute and refer to each sequence's position in the original data file, not its position in a particular database.

database_membership This field is a bit-vector (represented as a text string of "1" and "0s" characters) that identifies the databases that contain the sequence, where a "1" in a position indicates that the

sequence is present in the corresponding database and a "0" indicates that it is not present. Databases are represented in ascending order from left to right in the string, so the leftmost position in the string determines whether the sequence is present in database 1, the next position determines whether the string is present in database 2, and so on. By convention, this field always contains as many digits as there are databases in the file, but the only absolute requirement is that it not contain more digits than there are databases in the file.

domain_architecture If provided, this optional field must contain the string representation of an integer that can be represented as a long int. When `hmmpgmd` finds a hit, it converts this field into a long integer and stores it in the `desc` field of the hit's `P7_HIT` structure, which is then returned to the client[15]. EBI uses this field to encode the ID of a database entry that describes the sequence's domain architecture and is used to improve the display of results on their web clients.

[15] This is a nasty hack that we should avoid in the future if at all possible because it stores a long int in a pointer field. On most current machines, long ints and pointers are both 64-bit quantities, but there is no guarantee that this will be true in the future.

taxonomy_id If provided, this optional field must contain the string representation of an integer that can be represented as a long int. When a hit occurs, `hmmpgmd` converts the string into a long int returns it in the `acc` field of the hit's `P7_HIT` structure[16]. EBI uses this field to encode the ID of a database entry that describes the sequence's taxonomy for use by their web clients.

[16] This has the same problems as the handling of the `domain_architecture` field

As an example, consider the following set of sequence entries:

```
>1 100
ACDEFGHIKLMNPQTVWY
>2 010
ACDKLMNPQTVWYEFGHI
>3 111
EFMNRGHIKLMNPQT
```

Sequence 1 in the set is part of database 1. Sequence 2 is part of database 2, while sequence 3 is part of databases 1, 2, and 3. This set of entries omits the optional `domain_architecture` and `taxonomy_id` fields from each sequence.

Manual Pages Related to the Daemon

hmmc2 *- example client for the HMMER daemon*

Synopsis

hmmc2 [*options*]

Description

Hmmc2 is a text client for the hmmpgmd or hmmpgmd_shard daemons. When run, it opens a connection to a daemon at the specified IP address and port, and then enters an interactive loop waiting for the user to input commands to be sent to the daemon. See the User's Guide for the HMMER Daemon for a discussion of hmmpgmd's command format.

Options

-i <**IP address**> Specify the IP address of the daemon that hmmc2 should connect to. Defaults to 127.0.0.1 if not provided

-p <**port number**> Specify the port number that the daemon is listening on. Defaults to 51371 if not provided

-S Print the scores of any hits found during searches.

-A Print the alignment of any hits found during searches. This is a superset of the "-S" flag, so providing both is redundant.

hmmpgmd *- daemon for database search web services*

Synopsis

hmmpgmd [*options*]

Description

The `hmmpgmd` program is the daemon that we use internally for the hmmer.org web server. It essentially stands in front of the search programs `phmmer`, `hmmsearch`, and `hmmscan`.

To use `hmmpgmd`, first an instance must be started up as a master server, and provided with at least one sequence database (using the `--seqdb` flag) and/or an HMM database (using the `--hmmdb` flag). A sequence database must be in hmmpgmd format, which may be produced using `esl-reformat`. An HMM database is of the form produced by `hmmbuild`. The input database(s) will be loaded into memory by the master. When the master has finished loading the database(s), it prints the line: "Data loaded into memory. Master is ready."

After the master is ready, one or more instances of hmmpgmd may be started as workers. These workers may be (and typically are) on different machines from the master, but must have access to the same database file(s) provided to the master, with the same path. As with the master, each worker loads the database(s) into memory, and indicates completion by printing: "Data loaded into memory. Worker is ready."

The master process and workers are expected to remain running. One or more clients then connect to the master and submit possibly many queries. The master distributes the work of a query among the workers, collects results, and merges them before responding to the client. Two example client programs are included in the HMMER src directory - the C program `hmmc2` and the perl script `hmmpgmd_client_example.pl`. These are intended as examples only, and should be extended as necessary to meet your needs.

A query is submitted to the master from the client as a character string. Queries may be the sort that would normally be handled by `phmmer` (protein sequence vs protein database), `hmmsearch` (protein HMM query vs protein database), or `hmmscan` (protein query vs protein HMM database).

The general form of a client query is to start with a single line of the form `@[options]`, followed by multiple lines of text representing either the query HMM or fasta-formatted sequence. The final line of each query is the separator `//`.

For example, to perform a `phmmer` type search of a sequence against a sequence database file, the first line is of the form `@--seqdb 1`, then the fasta-formatted query sequence starting with the header line `>sequence-name`, followed by one or more lines of sequence, and finally the closing `//`.

To perform an `hmmsearch` type search, the query sequence is replaced by the full text of a HMMER-format query HMM.

To perform an `hmmscan` type search, the text matches that of the `phmmer` type search, except that the first line changes to `@--hmmdb 1`.

In the hmmpgmd-formatted sequence database file, each sequence can be associated

with one or more sub-databases. The `--seqdb` flag indicates which of these sub-databases will be queried. The HMM database format does not support sub-databases.

Options

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

--master Run as the master server.

--worker <*s*> Run as a worker, connecting to the master server that is running on IP address <*s*>.

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

--wport <*n*> Port to use for communication between workers and the master server. The default is 51372.

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

--wcncts <*n*> Maximum number of worker connections to accept. The default is 32.

--pid <*f*> Name of file into which the process id will be written.

--seqdb <*f*> Name of the file (in `hmmpgmd` format) containing protein sequences. The contents of this file will be cached for searches.

--hmmdb <*f*> Name of the file containing protein HMMs. The contents of this file will be cached for searches.

--cpu <*n*> Number of parallel threads to use (for `--worker`).

hmmpgmd_shard - *sharded daemon for database search web services*

Synopsis

hmmpgmd_shard [*options*]

Description

The hmmpgmd_shard program provides a sharded version of the hmmpgmd program that we use internally to implement high-performance HMMER services that can be accessed via the internet. See the hmmpgmd man page for a discussion of how the base hmmpgmd program is used. This man page discusses differences between hmmpgmd_shard and hmmpgmd. The base hmmpgmd program loads the entirety of its database file into RAM on every worker node, in spite of the fact that each worker node searches a predictable fraction of the database(s) contained in that file when performing searches. This wastes RAM, particularly when many worker nodes are used to accelerate searches of large databases.

Hmmpgmd_shard addresses this by dividing protein sequence database files into shards. Each worker node loads only 1/Nth of the database file, where N is the number of worker nodes attached to the master. HMM database files are not sharded, meaning that every worker node will load the entire database file into RAM. Current HMM databases are much smaller than current protein sequence databases, and easily fit into the RAM of modern servers even without sharding.

Hmmpgmd_shard is used in the same manner as hmmpgmd , except that it takes one additional argument: --num_shards <*n*> , which specifies the number of shards that protein databases will be divided into, and defaults to 1 if unspecified. This argument is only valid for the master node of a hmmpgmd system (i.e., when --master is passed to the hmmpgmd program), and must be equal to the number of worker nodes that will connect to the master node. Hmmpgmd_shard will signal an error if more than num_shards worker nodes attempt to connect to the master node or if a search is started when fewer than num_shards workers are connected to the master.

Options

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

--master Run as the master server.

--worker <*s*> Run as a worker, connecting to the master server that is running on IP address <*s*>.

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

--wport <*n*> Port to use for communication between workers and the master server. The default is 51372.

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

--wcncts <*n*> Maximum number of worker connections to accept. The default is 32.

--pid <*f*> Name of file into which the process id will be written.

--seqdb <*f*> Name of the file (in `hmmpgmd` format) containing protein sequences. The contents of this file will be cached for searches.

--hmmdb <*f*> Name of the file containing protein HMMs. The contents of this file will be cached for searches.

--cpu <*n*> Number of parallel threads to use (for `--worker`).

--num_shards <*n*> Number of shards to divide cached sequence database(s) into. HMM databases are not sharded, due to their small size. This option is only valid when the `--master` option is present, and defaults to 1 if not specified. `Hmmpgmd_shard` requires that the number of shards be equal to the number of worker nodes, and will give errors if more than `num_shards` workers attempt to connect to the master node or if a search is started with fewer than `num_shards` workers connected to the master.

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.