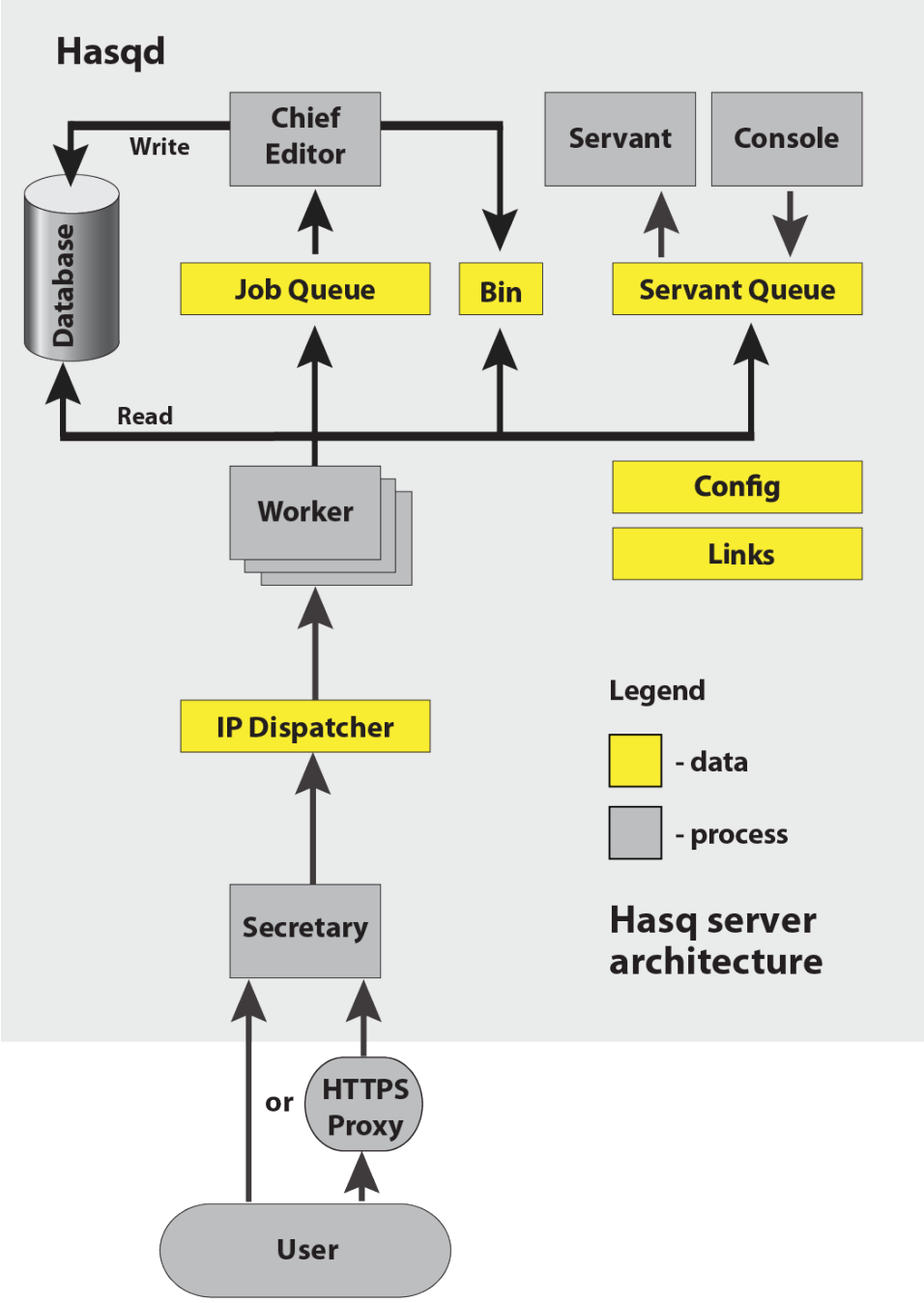
Hasq Server Architecture

23 May 2016, ver 0.4.1

A diagram of the Hasq server architecture is presented below. Further document sections refer to the component names shown in the diagram.



Hasq concept

Hasq technology aims to provide a mechanism for fast, reliable and secure transfer of ownership of digital tokens between peers. One aspect of this is keeping Hasq databases open to public at all times and making it extremely difficult for anybody to make fraudulent changes. To some extent the way a Hasq server handles information is similar to a traditional publisher printing newspapers and then libraries keeping copies of those newspapers. Once some information is made public, it is impossible then to hide it. Modifying the content in one of the newspaper copies may fool a number of people, but having multiple copies allows for this fraud to be noticed and rectified. Hasq servers act as publishers keeping a copy of what has been published earlier not only by this server, but also by other servers in the group.

Naming scheme

The similarity between a traditional publisher and functions of a Hasq server prompted the names of server components. For example, the Secretary much like in real life is responsible for handling incoming requests, while Workers execute those requests and the Chief Editor makes important decisions about modifying the Database. Other components' names follow this scheme.

Description of components

The entire codebase of a Hasq server is written in C++. It compiles under almost any version of Linux and Windows. Mac support is currently pending.

Secretary

The Secretary handles all aspects of dealing with the incoming user requests. Requests which do not comply with the server's request protocols or are erroneous will be rejected. The Secretary utilises an event driven asynchronous approach to handling TCP connections. This makes it a high performance, high concurrency server component with low memory use. A number of hasqd command line options affect the Secretary's behaviour:

tcp_port or **p** - sets the port number to listen to for incoming connections

cycle - determines the length of the Secretary cycle

dpul - causes the Secretary to print a dot character every cycle

dsec - turns the Secretary's debug output on

net_readtime - sets maximum reading time from a TCP socket

Once the Secretary receives a properly formed command/message, the TCP socket object containing this message will be passed on to the IP Dispatcher.

IP Dispatcher

IP Dispatcher is a storage container for TCP socket objects with valid incoming messages/commands. This container supports a form of sorting which allows Workers to extract objects from the Dispatcher in a way that guarantees fair distribution of Workers' time between requests coming from different IP addresses. This is needed to prevent user attempts to get advantage by sending many requests from the same IP address. When IP Dispatcher reaches its capacity, any further requests will be dropped

until one of the requests received earlier is processed. The capacity of this container can be changed from the default value by using the **qwkr** hasqd command line option.

Workers

Workers are processes responsible for executing the majority of user requests. They extract requests from the IP Dispatcher. Depending on the type of the extracted request different scenarios are possible.

The most common request is to read some data from the Database. If such a request is detected, it will be executed immediately by the Worker who extracted it. The status of the execution (success or failure) along with the requested data (if any) will be sent back to the user.

Another common request type is a command to add a new record to the Database. Since this command takes more time to execute than a request to read data, Workers place it in the Job Queue for later processing by the Chief Editor. Once the command is in the Job Queue, the Worker who placed it there sends a notification to the user containing their job ID and frees itself for processing further incoming requests. The job ID can be later used in a job status enquiry.

Workers may also access Bin and Servant Queue. The functions of these components are described in further document sections.

A number of hasqd command line options provided for changing the default Workers' behaviour:

dwkr - turns Workers' debug output on

workers - sets a number of workers in the system. Note: increasing a number of workers does not necessarily lead to increased performance.

Job Queue

Job Queue is a simple container used for storing requests to add new records to the Database. These requests will be later extracted and processed by the Chief Editor. If the Job Queue is full, in response to new requests users get a busy message. The length of the queue can be adjusted with the use of the **qced** hasqd command line option. Increasing the capacity of the Job Queue may help in situations where busts of requests to add a new record are expected.

Chief Editor

The main role of this component is to deal with new record creation requests. The Chief Editor extracts them from the Job Queue and processes on the first-in-first-out basis.

Upon extracting a request, the check is performed to verify that the new record matches the previous one in the corresponding chain of records. If it does, the Chief Editor issues a command to the Database to add this record. The result of this operation will be placed in the Bin.

If new record cannot be matched to the previous one, the Chief Editor sets the status of this operation to 'error' and also places it in the Bin.

Specifying **qced** hasqd command line option causes the Chief Editor to print debug messages.

Bin

Bin is a simple container for objects representing status of requests processed by the Chief Editor. Each object has an associated ID which is the same as a job ID sent back to a user earlier. Having this ID's allows users to enquire about the status of their jobs. When Workers detect such enquiries, they perform a search in the Bin. If an object with the same ID is found, its associated job status will be extracted and sent back to the user.

When Bin reaches its capacity, new job status objects will be pushing old objects out of the Bin. This means that a user has limited time to enquire about their job status before the status is lost. Increasing the capacity of the Bin lengthens this time. The size of the Bin can be changed from the default value by the **qbin** hasqd command line option.

Database

Database is one of the core Hasq server components. Depending on the settings it stores all or some of the records published by users.

Internally, Database is a collection of databases which may or may not be compatible. Each separate database consists of components of two types: slices and indices. Slices contain a chronological list of records as they arrive. Indices have the same information organised in a way which allows for fast search operations. Except for a few exotic cases, indices can be regenerated from the information contained in slices.

The internal Database design allows for reading and writing operations almost always to be performed simultaneously which greatly improves performance.

A number of hasqd command line options are provided for changing the default values of some of the Database parameters:

db - sets Database root directory

lastdata_max - sets search limit for *lastdata* network command

range_max - sets limit on how many records can be extracted by *range* network command

Servant, Console and Servant Queue

These components are mainly used for administrative and testing purposes. They are not essential for typical Hasq server applications.

If **console=1** hasqd command line option is specified, it causes a Hasq server to create the Console component which listens to input from the terminal window the server is running in. A limited set of commands which can then be issued, can be viewed by typing 'help' after hasqd is started.

Servant presents a way for the server administrator to instruct the server to perform certain actions during start time. Servant supports a language called HSL (Hasq Servant Language). HSL statements can be passed on to a starting server directly in the command line. Alternatively, a name of the file with HSL statements to execute can also be specified. Both options are realised with the use of the **script** (or **s)** hasqd command line option. A few other options can also be specified:

dprn - causes Servant to print its messages

dsvt - turns Servant's debug output on

qsvt - sets Servant Queue size

Config

Config is a collection of all settings used by a Hasq server while it's working. Some of the settings can be changed by a server administrator via different hasqd command line options. Other settings are fixed or dynamically acquired and cannot be changed.

Links

A Hasq server can be part of a larger group of Hasq servers which share information among them. Each server in the group is connected to a number of other servers which it directly communicates with. Those servers are called neighbours.

When a user creates a new record on a particular server, this server notifies its neighbours about this event. The neighbours then request the new record and in turn notify their neighbours. This causes a quick spreading of the new record over the entire network of servers.

At regular intervals servers check the availability of their neighbours. If some of the neighbours cannot be reached, a process of network re-organisation starts in order to maintain the optimal network configuration among live servers. The same process runs when new servers join the group.

Links component is a container for the list of the server's neighbours as well as for some other data related to network configuration. The **iplock** hasqd command line option can be used for locking certain servers from being deleted from the list of neighbours during the re-organisation process.

The latest version of this document can be downloaded from http://hasq.org