Haproxy.c

-Starting HaProxy – It starts listening

* Init -
* signal\_register\_fct()
  + (signal.c) Register a function and an integer argument on a signal. A pointer to the newly allocated sig\_handler is returned. The caller is responsible for unregistering the function when not used anymore.
* Check status of global.rlimit\_memmax, global.rlimit\_nofile..
* Retries
  + Start\_proxies
    - (Proxy.c ) This function creates all proxy sockets.
    - The sockets will be registered but not added to any fd\_set, in order not to loose them across the fork().
    - The proxies also start in READY state because they all have their listeners
  + Protocol\_bind\_all:
    - (Proxy.c) :binds all listeners of all registered protocols
  + the father launches the required number of processes – fork()
  + protocol\_enabke\_all :
    - (procy.c )enables all listeners of all registered protocols. This is intended to be used after a fork() to enable reading on all file descriptors.
  + Cleanup

Frontend.c

* Frontend\_accpet : does an accept and causes the lisener to disable
  + Setsockopt (Linux call) – sets the socket options depending on thr protocol
  + Then bases on the type of connection
  + conn\_get\_from\_addr
  + ACL\_supported keywords and structures defined

Backend.c

* recount\_servers
  + – recounts the number of usable active and backup servers for proxy. They are returned to p->srv\_act and p->srv\_bck
  + Computes active and backend weights (Does not update tot\_weight or tot\_used)
  + Checks if the servers are usable , if it is then gets the values
* Update\_backend\_weight
  + updates the backedn’s tot\_weight and tot\_used once the server weights are updates
* getserversh – following source hash method
  + tries to find running servers for proxy
  + Depending on the number of active/backup servers it will either look for active or backup servers.
  + Don’t hash if only 1 server is left
* Getserveruh –following the URI hash method
  + Same as above
* Getserverph – following the URL hash parameter
  + Same as above
  + Looks for the specific parameter in the URL and hashes it to compute the server od
  + This is useful in optimizing performance by aviding bounces between servers in context where sessions are shared but cookies are not usable
* Getserverph-post – same as above checkes body contents
* Getserverhh – following the header parameter hash method
* Getserverrch - checks cookies and then acl\_fetch\_rdp\_cookie
* Assign\_server –
  + This function applies the load-balancing algorithm to the session, as defined by the backend it is assigned to. The session is then marked asassigned'.
  + This function MAY NOT be called with SN\_ASSIGNED already set. If the session had a server previously assigned, it is rebalanced, trying to avoid the same server, which should still be present in target\_srv(&s->target) before the call.
  + The function tries to keep the original connection slot if it reconnects to the same server, otherwise it releases it and tries to offer it.
  + fwrr\_get\_next\_server(lb\_fwrr.c) etc functions to allocate the server using the LB policy
* \*Assign\_server\_address
  + Assigns a server address to a session
* Assign\_server\_and\_queue : assigns server and queue – could be proxy;s queue or server;s queue
* Assign\_tproxy\_address
* Connect\_server: This function initiates a connection to the server assigned to this session
* Backend\_lb\_algo\_str: returns the string of the LB policy used in dumpstats
* Backend\_parse\_balance: parses the config file anfd gets the LB policy from that
* All the fetching functions..This is called by cfg\_parse\_listen of cfg\_parse.c
* Supported ACL declaration

Server.c

* srv\_downtime- returns the downtime used by dumpstats
* srv\_getinter – used by checks.c
* srv\_register\_keyword: register the server list for the next parsing session used by ssl\_sock.c
* srv\_find\_kw – used by cfgparse.c
  + Return a pointer to the server keyword <kw>
* Srv\_dump\_kws: Dumps all registered "server" keywords to the <out> string pointer. Used by cfgparse.c
* Srv\_parse\_id -
* Srv\_parse\_weight -

Checks.c : Deals with the health of the servers, if one server goes down redispatched the connections to the pther servers..

* Defines various status states
* get\_check\_status\_description
* get\_check\_status\_info(
* get\_analyze\_status - calls analyse\_statuses- matches the status form the declares status and returns the appropriate one. used by health\_adjust
* server\_status\_printf- prints the status of the server called when the state of the server changes, it goes down, comes up or checking health.
* set\_server\_check\_status: show information in logs about failed health check if server is UP or succeeded health checks if server is DOWN. – used by various functions in checks.c
* set\_backend\_down: sends a log message when server goes down, called in checks.c by set\_server\_down and set\_server\_disable
* redistribute\_pending – resdistributes pending connction when a server goes down. Called from set\_server\_down and set\_server\_disable
* check\_for\_pending - Check for pending connections at the backend, and assign some of them to the server coming up. The server's weight is checked before being assigned connections it may not be able to handle. The total number of transferred connections is returned. Called from set\_server\_up, set\_server\_enabled and set\_server\_warmup
* shutdown\_sessions : Shutdown all connections of a server: called from server\_down and shutdown\_backend\_sessions .
* shutdown\_backend\_sessions – called from set\_server\_up
* set\_server\_down etc more functions.
* health\_adjust
* event\_srv\_chk\_w
* process\_chk: anages a server health-check. Returns the time the task accepts to wait, or TIME\_ETERNITY for infinity. Called by start\_checks
* start\_checks- called from init function of haproxy.c

proxy.c

* + Proxy\_cap\_str : Returns a string containing a name describingng capabilies to report error message. Specifically, it will return the words "frontend", "backend", "ruleset" when appropriate, or "proxy" for all other

cases including the proxies declared in "listen"

* + Proxy\_mode\_str : returns string containing the mode of the proxy
  + Get\_backend\_server: returns the first backedn and first server with the given name
  + Proxy\_parse\_timeout :
  + Proxy\_parse\_rate\_limit:
  + Findproxy\_mode : finds a proxy with matching name, mode and with satisfying cpapbilities.
  + Findproxy : proxy with matching name
  + Findserver : Server with matching name within selected proxy
  + Proxy\_cfg\_ensure\_no\_http : checks if it is in http mode if not ignore cookie etc
  + Init\_new\_proxy : initialize the new proxy or peer
    1. Memset
    2. List\_init
    3. Reset\_timeout
  + Start\_proxies: This function creates all proxy sockets.

Lisetener.c

* listener\_accept: This function is called on a read event from a listening socket, corresponding to an accept. It tries to accept as many connections as possible, and for each calls the listener's accept handler (generally the frontend's accept handler)
* Other listener functions

Proto\_http.c: has all the http related functions , called from process \_task in the main – haproxy.c

Session.c

* Session\_accpept() – called from lisetener\_Accept

Observations:

* Follows an event-driven architecture
* TCP over HTTP – creates sockets –listens -> accepts and then calls session\_accept functions
* Listeners –Accept - session\_accept.
* Frontend – does an accept – receives the requests
* Backend – related to the LB policies etc., update server weights and assign servers
* Haproxy.c – main initialize and starts listening.
* Checks.c- responsible for checking the server health