Gateway
* Service Discovery
* REST API
* Load Balancing: Service Load

Cache
* Keep-alive connection
* Multiple Simultaneous Connections
* Query Language

Services:
* SQL, NoSQL Databases
* Tasks distributed across multiple requests
* Service discovery
* Status Endpoint
* Task timeouts
* Unit Testing
* RPC

Implementation Details

1 The gateway (Elixir, for high availability and for being fault tolerant)

The gateway has service discovery by keeping the services addresses in memory, keeping them updated with heartbeats.

The Gateway has an api to search for words

The gateway knows about services and their load and prioritises where to redirect api calls

The gateway keeps the connection non-blocking open and remembers for which user to send back the response

2 Cache (redis)

A nice redis cache with sanitized search terms as keys and list as value

3 Frontend Services (Python)

Services use a connection pool for MySQL database

Each service is given to process a request by load balancer

Each service registers itself to the gateway and send heartbeats to confirm

Each service exposes an SSE of its status to provide info about load

Each service will implement retry policy

4 Backend Services

Visit supervisor will know about crawlers with service discovery and keep a batch of links to be visited/parsed

The crawlers will send the results in a standardized form to parsed supervisor via rpc, to eliminate decoding

The parsed supervisor will batch results into persistent redis to be added to “Processed” SQL db, also will add the link lists from parsed pages to the Redis + SW queue (which will be polled for updates in batch by visit supervisor)

The service worker will watch for the memory not to get overloaded and cache the results into non-volatile memory

5 Updaters

To keep high availability for the data, the keyword/link db will be duplicated with active/passive pattern.

The update will be written to the passive db and all the data will be available with the switch of the db access proxy

API's

Gateway:

inspect get() -> SSE Connection STATUS
search get: (): string[] -> array of links (probably, with short description) || ERR
register-service post(address) -> OK

Redis:

get-cache get (terms) -> mimic gateway get || ERR
set-cache set (terms, results) -> OK || ERR

Frontend Services:

inspect get() -> SSE Connection STATUS

search tcp recv(terms): string[] -> array of links
send STATUS, ERR, KILL

Frontend Connection pool:

inspect get() -> SSE Connection STATUS
read get(query) -> relational table

Access Synchroniser

inspect get() -> SSE Connection STATUS
listen get () -> SSE Connection
* Halt
* Read address
* Write address

Parsed Supervisor:

inspect get() -> SSE Connection STATUS
saveResult(message: {address, terms[], newLinks[]}) — **gRPC**

Visit Supervisor:

inspect get() -> SSE Connection STATUS
register-crawler post(address) -> OK

Crawlers:

inspect get() -> SSE Connection STATUS
work tcp recv(string) -> OK
send STATUS, ERR, KILL