Yandex

Scalable ejabberd

Konstantin Tcepliaev

Moscow Erlang Factory Lite
June 2012

ejabberd

- XMPP (previously known as Jabber) IM server
- Erlang/OTP
- Mnesia for temporary data (sessions, routes, etc.)
- Mnesia or ODBC for persistent data
- Modular

Too naïve

- Works excellent on one or two always-on machines with several thousands connections
- When it comes to a pretty large installation (tens of considered faulty machines, hundreds thousand connections), problems start to emerge

ejabberd at Yandex

- XMPP IM server
- Backend for web chat service
- Transport for iPhone/Android mobile mail application
- PUSH notifications for Yandex.Disk

Scalability issues

- Direct message passing to PIDs without any guarantees
- Memory consumption: large #state{} of client processes, separate TCP receivers
- Mnesia-based storage

Message passing problem

- If remote node goes down, all messages sent there are lost
- In Erlang, PIDs aren't being invalidated — if process has exited already, your message is lost too
- No method to know whether message was received or not

Trial by error

- First thought: gen_fsm:sync_send_all_state_event/3
- Involves receiving reply, which is another message
- Synchronous, need to wait for timeout when calling dead process
- Decision: unacceptable

Trial by error

- Second try: node/1 to distinguish local and remote PIDs
- Local: Use is_process_alive/1, send directly if true, failover if false
- Remote: send message to {ejabberd_sm, node(PID)} and mess with it there.
- Downside: ejabberd_sm becomes a bottleneck.

Solution

- Final version: separate the messages
- <message/> stanzas are critical, should be sent as described before
- <iq/> and <presence/> aren't that important and may be sent as usual
- Further improvements for SM: process_flag(priority, high) or replacing with a process pool.

Memory consumption

- Client (C2S) process state consists of ~30 fields, of which ~10 are used only during stream negotiation
- Subscription lists are always stored in state.
- Receiver is another process, separate from C2S — two processes per client.

Large state

- Split C2S process callback module in two, with their own states
- gen_fsm for stream negotiation
- gen_server later
- gen_server:enter_loop(ejabberd_c2s, [], State, hibernate)
- Free bonus: 'hibernate' option triggers garbage collection

Subscription lists

- Subscription lists store information about contacts with which client should exchange presence information
- No need for them when user doesn't send presences
- Solution: retreive only when presence is sent.

Separate receiver process

- C2S and receiver share the same TCP socket
- C2S sends data to socket
- Receiver controls the socket, waits for {tcp, ...}, decodes XML and passes it to C2S.
- Solution: eliminate receiver process and let C2S control the socket and do all the job.

Memory: results

- Average memory consumption for one client connection have dropped ~40% to ~500kB
- Database load dropped slightly

Mnesia

- Transactional — doesn't scale after some limit
- Brain-splits are fatal
- Node deaths are fatal
- Any inconsistency is fatal

MongoDB

- Eventual consistency
- Replica sets for failover
- Sharding for load balancing
- Self-healing after brain-splits
- Theoretically unlimited scalability

Late 2010

- Mongo 1.2, no replica sets, unstable sharding
- Manual sharding in SM, using phash2 on usernames
- Replica pairs with master-to-master scheme
- Writes and reads both go to one replica, other is used as failover
- ETS caching of local sessions on each node

Early 2012

- Mongo 2.0.1
- Requests per second limit reached
- Master-to-master has become obsolete
- Replication deadlocking, hard to investigate and impossible to predict

Trying replica sets

- All database requests seem to need to retreive consistent data, so all operations are done on master nodes
- Secondaries are kept for failover
- Mnesia removed completely
- Problem: RPS on master is still the same, requests time out on load peaks.

MongoDB: final solution

- Session data doesn't actually always needs to be consistent
- Retreiving contacts' sessions from master to send them initial presence
- All other reads are done on secondaries, using slaveOk() option.

What it looks like now

- Reading from secondaries: ~60% requests
- Writing to primaries, with sharding
- Cluster easily survives brain-splits, network outages, node downtimes and massive user reconnects
- Huge potential for horizontal scaling

Questions?

f355@yandex-team.ru