COMARCH

Erlang in the battlefield

Łukasz Kubica

Telco BSS R&D Department

Cracow Erlang Factory Lite, 2013

Agenda

- Introduction to the SCM
- Erlang vm and upgrades
- Tracing
- Mnesia
- Final thoughts
- Questions

The Session Control Module

- A part of the Comarch Billing System
- Does AAA for SIM cards
- Enforces limits (e.g. max data volume per month)
- Can do fraud detection (e.g. IMEI binding for M2M SIMs)
- Performs on-line charging (using a C node)
- Evolved from a real-time Data Processing Server (rtDPS) developed in C/C++
- On production for a long time - still no downtime

Agenda

- Introduction to the SCM
- **Erlang vm and upgrades**
- Tracing
- Mnesia
- Final thoughts
- Questions

Always build on a rock

- Customer does not care about technology, when a 3rd party component fails, whole system fails
- A language used for development is less important than a platform (vm, libraries)
  - Ever traced a memory corruption or a leak on a live system?
  - Or maybe tried to tune java GC times?
  - Not enough or too much logs?
  - Your allocator’s heap got fragmented?
- Our experience with Erlang
  - The vm is compact and written in plain C
  - Great memory stability - no heap fragmentation.
  - Traceability
  - Good performance (for a vm)
  - Simple yet powerful language
  - But nothing is perfect ...

Just give me a little tuning

- Bind your schedulers
  - Scheduler context switching is a problem
  - Binding schedulers give a big performance boost
  - Remember to leave some room for other processes!
  - The biggest problem: `sched_setaffinity` simply does not work on some virtual configurations, so the binding itself does not work too
- Turn off the load compacting (`+scl`)
  - All your cores are belong to us - so don’t let them sleep
  - We (and apparently Basho) have experienced severe and occasional performance drop which seems to be connected with load compacting
  - Processes simply are not homogeneous - think many workers using one `gen_server`
- The biggest VM problem - scheduler tuning is hard, and it has changed in R16.

Releases and upgrades - the Good

- Hot code loading is great, you can hotfix easily
- Release system is done right - you can prepare upgrade that will determine what to do when installed. This is a real benefit
- You must be prepared for node restarts in more complex cases. In HA system you have spare nodes, but during upgrade your system is not so HA
- In fact upgrades are the most risky thing you can do on a live HA system

Releases and upgrades - the Ugly

- Records are NOT done right in erlang
  - Say we have `myfun(#record{field=X})`
  - Now let’s add a new field to the record
  - And imagine you have N modules with such matchspeces
  - And try to run an upgrade ...
  - You can tell one version from another, but you code will become a total mess
- Records versioning should be supported out of the box
- It may seem that atomic loading of multiple modules might do, but things are more complex

Releases and upgrades - the Bad

- A real fun begins when you have records and mnesia
- Solution that typically works:
  - Upgrade binaries on all nodes
  - Make all modules support old and new version and use old by default
  - Switch a param and make all you modules write a new version and convert from old on read
- The problem is that you will not notice if you binaries support the new version in a wrong way - until it’s too late

Agenda

- Introduction to the SCM
- Erlang vm and upgrades
- **Tracing**
- Mnesia
- Final thoughts
- Questions

A quick look at erlang tracing

- One of the biggest and most important erlang features – you simply have to know it
- You can trace both system events (like GC, process scheduling) and calls
- It is so good that we do not use any debug logs anymore
- Just remember one thing - when you trace calls, trace flags are bound to module instances
  - Beware on-demand code loading, only modules loaded before tracer setup will be traced
  - When you reload a module, you should setup tracing again
- Tracing is useful in two ways
  - Obviously, it allows to check what is going wrong
  - But it can also be used for system profiling and even monitoring (thanks to low performance penalty)

COMARCH

Sequential tracing - a godsend

- Imagine a system which spawns a process per request (not hard, isn't it ?) with 1500 req/sec
- Once a few minutes you get a request for a certain SIM which mysteriously fail. You have a callstack in your logs, but it does not help much
- You can either release new binaries or simply learn sequential tracing
  - Find a function with argument allowing you to identify the subject (e.g. IMSI number)
  - Launch `dbg` or `ttb` with this function adding a matchspec for your entry point which activated sequential trace
  - For every other function add matchspec which matches only when process is infected
- We did a simple tool with predefined set of modules/functions. Uses the `ttb` module.

Sequential tracing cont.

- Entry point matchspec

```
dbg:fun2ms(fun (Args) when hd(Args) == Trace
               set_seq_token(send ,true),
               set_seq_token(’receive’,true),
               set_seq_token(timestamp,true),
               exception_trace()
           end)
```

- Standard matchspec

```
dbg:fun2ms(fun (_) when is_seq_trace() ->
                exception_trace()
           end)
```

Performance monitoring

- `etop` is nice, but is process oriented. When you have 1500 processes/sec it is rather hard to use it
- `fprof` is an offline tool - traces all and have huge performance penalty
- `eprof` let’s you profile a live system, but the API takes ONE single MFA at a time :-(
- However, it’s easy to create your own tool:
  - Use call tracing with `timestamp` and `exception_trace` - you can measure time between call and return/exit
  - You can enable some GC tracing for even more info
  - Then simply aggregate your data (you will have to record some data per process)
  - Add some info from sockets, system, `process_info` (see `etop` code for some undocumented API’s)

Agenda

- Introduction to the SCM
- Erlang vm and upgrades
- Tracing
- **Mnesia**
- Final thoughts
- Questions

Mnesia is cool

- Great functionality out-of-the-box, for free
- The idea is quite simple, and simplicity is good
- Very elegant programming model (funs as activities)

Mnesia - transactions and efficiency

- Transactions are generally on the slow side (compared to ETS)
- The locking model avoids deadlocks, but if transaction lasts too long, the sleep strategy takes the toll
- Sometimes, if only one node modifies data, it is better to make a `gen_server` plus dirty for shared resources - but you loose rollback
- The `dist_auto_connect` trap
  - Experiment – <u>**physically**</u> disconnect a replica node during high load
  - Watch your system die …
  - Why ? To resolve each transaction your system needs a `net_setuptime` seconds
- We got rid of transactions almost completely (when you have some shared resources, you should do everything to have them modified only locally)

Mnesia - memory tables

- Performance of memory tables is great, much better then disc tables
- But you have to be very careful
  - You can easily persist table using `dump_tables`, but it locks table for read, so it was no-go for us ...
  - You can use backup module, which uses snapshot strategy, but then you have a startup problem if all nodes go down
  - For disc tables, mnesia keeps track of longer running node, and `wait_for_tables` will timeout if nodes are started out of sequence
  - But not for memory tables, they are simply loaded empty (if there happens that no other node is present during startup)
  - So depending on your strategy, you can either load old state, new state or stay with empty tables

Mnesia - indexes and table loading

- Indexes are unusable if you have many records per key
  - Index simply holds a list of keys. Every operation on it is a list operation
  - When you start a node, index is build up from the ground, so there is lots of operations on long lists
  - Suppose we have 1 million records, which are logically grouped into 100 groups and you have an index on `group_id` field
  - Such a table will load for ages, much longer then it is safe for a HA system
- Table replication holds a read lock too, so when you start a new node, performance will suffer
- Partial solution - use frags, but in our case this did not solve our grouping problem
- So, we did a mnesia customization and we use ETS for index instead of a list (so basically, ETS index holds an ETS table id instead of a simple list)

Final thoughts

Erlang is a solid platform to build HA applications on. There are some gotchas, but nothing can simply be perfect. Considering the SCM, erlang seems to be a sweet spot - development is robust, library ecosystem is large and high quality, the VM is very stable.

You simply feel that it has been done by professionals for professionals and that’s a lot.

COMARCH

Thank you

Lukasz.Kubica@comarch.com