Default stack and other evils

Bloody story of RPKI Validator

Mikhail Puzanov
RIPE NCC
RIPE NCC RPKI Validator

→ RPKI validator project is a part of our Resource Public Key Infrastructure suite of RIPE NCC.

→ It is daemon installed on the user’s servers.

→ It has to be relatively humble with resources.

→ It has to be stable and reasonably fast on a wide range of platforms and hardware.

→ It has to put reasonable configuration burden on the users (don’t ask them to install and configure an RDBMS or http proxy).

→ We have to think about corner cases more than for internal services working in our data centers.
What does RPKI Validator have to do

- Has to validate ~60000 signed objects of total size slightly above 100mb.
- Data is updated, all the updated data in 7 days would be 2-3 gigabytes.
- This data (normally for 48 hours) is stored in a local database by the validator.
- Above 900 000 BGP announcement total, more than 10% or them are signed.
- With the currently used Java crypto-libraries (bouncycastle) validation for all trust anchors takes about 90-120 seconds on one modern CPU core.
- Almost linearly faster with more CPU cores.
RPKI Validator

- RPKI Validator 3 is the current version
- Validator 3 started as a replacement of Validator 2 (there was a version 1 long ago)
- Validator 2 had problems
  - Memory consumption (above 3.5Gb for the current size of the repositories)
  - Rare, but recurring stability issues (OOM, embedded database corruption, database deadlocks)
  - It’s written in Scala and it’s hard to expect PRs from the community.
- Let’s rewrite it.
- Let’s rewrite it in Java 8.
RPKI Validator 3

- Written using all the “default” classical Java stack
- Spring Boot for DI and REST API/HATEOAS
- H2 as an embedded DB with Hibernate ORM
- More extensive, normalized data model for smarter behaviour
- A lot of in-memory data moved to the DB
- Quartz for multiple types of jobs
- Angular for the UI
- Project took about 10 months to deliver.
RPKI Validator 3

And then Validator 3 had problems (surprise-surprise)

- Memory consumption (aimed at 1Gb heap, much less than version 2, but we still got regular OutOfMemoryException’s)
- Multiple stability issues: “doing nothing”, “slow start”, “crashed and stuck”, ... etc.
- H2 database size goes through the roof for some users (we had a bug report about 50Gb).
- Very slow “warm-up” in some cases.
Task scheduling in Spring sometimes just “doesn’t work”.

Hibernate keeps unpredictable amount of objects in memory, resulting in OOMs.

Some Hibernate queries are slow to the point of REST API calls timing out.

Concurrent work with the DB causes weird race conditions.

Transaction management is hard to get right.

H2 not always recover after application crash

H2 doesn’t have online garbage collection, so the database only grows.

Combination of ORM and query planner from H2 is not always efficient, resulting in very slow queries.
RPKI Validator 3

➔ A few months of fixing bugs almost every week.
➔ Growing user base and growing number of bug reports.
➔ We needed to change the design.
➔ We need to change embedded DB, the core of all troubles.
RPKI Validator 3 - LMDB to the rescue

- It’s been there for quite some time and it is proven to be reliable.
- Dead simple: ordered key-value store where both keys and values are byte arrays.
- Low-level Java-bindings in **lmdbjava** library with tiny native library.
- Full ACID with snapshot CC: readers don’t block writers and vice versa.
- MMAP implementation, zero-copy reading, no configuration, no cache management, no WAL, no separate compaction steps, instant crash recovery.
- LMDB is faster than even the low-level back-end of H2 in almost all benchmarks.
RPKI Validator 3 - LMDB bright side

➔ Literally every database query became faster, from “a little faster” to “orders of magnitude faster”.

➔ “Associated 35650 objects with the validation run 0000000000000014 in 58ms”

➔ Able to respond to REST API calls with good latency under CPU usage of 600%.

➔ 2-3 times smaller database.

➔ Quick start and shutdown, not a single case of “cannot restart after dirty shutdown” in two months of testing.

➔ Heap size went down from 1Gb to 640Mb, without “out of memory” problems.

➔ Multiple strange bugs disappeared at once.
RPKI Validator 3 - LMDB dark side

➔ No type-safety, everything is a byte array, really low-level basic API.
➔ Database is not self-aware, no metadata, no schema and no schema migrations.
➔ Had to implement serialisation and indexes ourselves, type-safe key-value maps, safe transaction API, etc., code base grew pretty significantly.
  ◆ SLOC before LMDB ~11000
  ◆ SLOC after LMDB ~15200
➔ Native library in dependencies.
Got bitten by a Data Corruption Bug!

Very rare and very subtle: some values (1 or 2 out of 100000) are corrupted after a couple of days of running, happens on Linux and Mac, but not *BSD.

Very hard to reproduce. A month of work to figure out where exactly it comes from -- no result.

We believe it’s somewhere between JVM and the mmap-ed off-heap segments, not in any Java code and not in LMDB itself.

Positive side effect while fixing: reduced the amount of updates to the minimum.

Had to give up and find yet another DB.
RPKI Validator 3 - Xodus to the rescue

- Another key-value store with a some fancy features on top.
- Written by JetBrains in pure Java, no native libraries in dependencies.
- Pretty much the same ACID semantics as LMDB.
- Reasonable performance: bearably slower (2-3 times) than LMDB on average.
- Even smaller database with better space reuse due to using multiple files.
- The only flaw so far: high memory consumption by large writing transactions, had to increase Xmx from 640 to 1024 and then even to 1536mb by default.
- No crashes, no data corruptions, nothing really bad or very exciting about it.
- Tested for a month and ended up in version 3.1.
Lessons learned

➔ As a rule of thumb, do not use Hibernate whenever resource usage has to be under control.
➔ As an even better rule of thumb, don’t use Hibernate at all.
➔ H2 looks neat at first but is not very reliable and unpredictably eats disk space.
➔ Spring Boot is slow, eats memory and doesn’t really do much more.
➔ LMDB is great, but there’s something fishy going on between LMDB and JVM.
Lessons learned

➔ Measure performance. Talking about performance without solid numbers is almost always waste of time, most of the assumptions happen to be wrong.

➔ Solid numbers are hard to get. VisualVM sampler “lies”, VisualVM profiler “lies”, System.nanoTime() “lies”, correlate all of them.

➔ There is a sweet spot between offloading work to frameworks and writing code manually. We went towards the first option way too much and paid the price.

➔ Spend time on research, don’t pick up the “default stack” right away.
Finally

- RPKI Validator 3.1 is pretty stable, we haven’t seen stability bug reports.
- We plan to work on it mainly to improve usability and packaging.
- It is yet to be decided what is going to happen with it the future.