Unified Business Logic API and Storage Layers a.k.a. “Re-Current”

Draft, July 2023

Permalink to this doc

On the origin of the term “Re-Current”:
http://dima.ai/static/current.pdf

Product 101

Think of the following out of the box:

- **A data storage with the APIs and the performance comparable to Redis.**
  - But the amount of data to work on is unbounded!
  - Because, while most of the data remains in memory, most of it lives on SSDs.
- **This storage is distributed, and it never loses data.**
  - Each mutation to the data is first replicated to N other locations.
  - As long as (N+1)/2 locations are available and connected to each other, the system lives on.
- **The storage designed for attaching eventually consistent replicas.**
  - The very service is a large, high-throughput, low-latency OLTP engine.
  - Behind the scenes of this OLTP is the Event Store, and CDC is included in the package.

Product 102

The data storage layer is not just “Redis-like”, it is generic.

- **Redis data structures, including the MULTI statement, are just a demo.**
  - The idea of a strongly linearizable order of mutations is unconditionally respected.
  - The actual mutations can implement arbitrary complex logic.
    - As long as it “evaluates” in a short, provably bounded time.
- **Strongly consistent mutations and queries go through the leader.**
  - Instead of Lua, a statically typed high-performance language is used.
  - The storage is transactional by design, and any programmable invariant can be respected.
- **Eventually consistent operations are horizontally scalable.**
  - Mutations and strongly consistent reads have to get their sequence numbers stamped.
  - Eventually consistent read or query operations can be executed against any replica.

Technology 101

The implementation relies heavily on:

- **The fact that a single CPU can do a lot.**
  - As proven by Redis.
- **The fact that distributed consensus is a solved problem.**
  - As proven by etcd.
- **The fact that API gateways are finally merging with schema registries.**
  - From Swagger to Buf.build and various gRPC proxy implementations.
- **And the fact that storage is cheap these days, while “DBs” are not.**
  - There is just no pure solution on the market the combines the above.
    - Hence this slide deck!

Product 201

Realistically, the data is, first and foremost, an event log.

- **The overall solution is the Event Store.**
  - An immutable, append-only event log.
  - And this is why CDC and data feeds come in organically.
- **The bodies of CQRS commands and the schema, are first-class citizens.**
  - They are stored alongside other data in the same storage space.
  - And they are subject to the same evolution invariants to ensure correctness.

Technology 201

One of the major building blocks is the Total Order Broadcast (TOB) engine.

- **Existing distributed consensus solutions are slow.**
  - They can handle a few cross-DC mutations per second, but not more.
- **We are in the throughput game, not in the latency game.**
  - It is more important to persist the data than to respond quickly.
  - And distributed data replication is slow by design.
  - So, it is acceptable to have ~200ms latency for what has to be strongly consistent.
- **And the fact that storage is cheap these days, while “DBs” are not.**
  - There is just no pure solution on the market the combines the above.

Product 301

External interfaces are part of the system itself:

- **Requests enter the system through a “frontend cluster” gateway.**
  - This is essential to ensure linearizability for mutations and for strongly consistent reads.
- **This gateway is extensible by design.**
  - In particular, HTTP+JSON and gRPC methods are organically wrapped to CQRS commands.
  - Both the strongly consistent and the eventually consistent paths are covered.
- **The end-to-end system is deployed organically, as a single logical unit.**
  - AWS, GCP, or really anything Kubernetes-friendly.
  - S3 is used as the cheapest storage layer on the market, if AWS is used.
  - SaaS deployments can be liberally priced based on compute and storage costs.

Technology 301

Behind the scenes, we are looking at a proprietary ledger.

- **CQRS commands <=> orders to execute smart contracts.**
  - Linearizability <=> an immutable, append-only event log.
  - Hence the Event Store paradigm, where CDC and data feeds come in organically.
- **The CQRS queries have exclusive access to all the data.**
  - If they need to be strongly consistent, they go through the leader.
  - If they can be eventually consistency, any replica would do.
- **The schema of the data is also incremental, and stored on the ledger.**
  - Along with the schema, schema evolution rules are stored as needed.
  - The end users do not notice that the very schema is evolving.
    - As long as the CQRS commands in use are kept backwards- and forward-compatible.
- **Same applies to the outer-world-facing interfaces.**
  - TL;DR: The CQRS queries and commands are natively mapped to HTTP+JSON / gRPC calls.

Technology 302

Peak strongly consistent data access is never compromised.

- **Asynchronous, long-running jobs can be run separately.** The lifecycle is:
  - Implement new CQRS commands and queries to “read old, write new” data.
    - Push these changes, thus activating “read-through migration”.
    - The data begins to be moved, most actively used pieces first.
  - Mark the now-read-only data as immutable.
    - This is a static change. CQRS commands that try to mutate this data will fail to build.
  - Run a batch job that converts this data into new data structures in the background.
    - Logically, one could “touch” every single bit of data in some throttled way.
    - But it’s better to perform this “cold migration” off the “main linearized thread”.
  - Once complete, retire the old data.
- **To ensure high throughput, the runtime of each command is limited**
  - In particular, no full scans and no batch mutations of the data on the leader.
- **If a new “index” has to be added to the data, the above process applies.**
  - So that efficient lookups can be implemented over data structures with no up front indexes.

Product 401

To present the above, we solve a few “standard” problems with it. The demos include:

- **A “simple” “decentralized trusted ledger” solution.**
  - Handle 10M++ “Alice-pays-bob” transactions per second easily.
  - With guaranteed invariant validation.
  - And that’s a single ledger; 100M++ is easy with sub-ledgers and cross-shard txns.
- **Effectively, a “large Redis”.**
  - Implement a wide enough subset of Redis commands.
  - Show these commands to run on terabytes-large datasets.
- **Effectively, a “fast Kafka”.**
  - The protocol can handle more throughput than Kafka, with stronger durability guarantees.
- **“Standard” SysDesign interview questions**
  - **“Design Twitter”:** “feed generation” and “celebrities”.
  - **“Design Ticketmaster”:** strongly consistent transactions.
  - **“Design Telegram”:** 1:1 and channels and groups.

Implementation 101

The system consists of several components:

- **The Gateway / Frontend Cluster.**
  - To orchestrate sending commands in parallel to N geo-distributed backends.
  - Client libraries to interface with it (incl. time skew compensation, etc.)
  - And HTTP+JSON and/or gRPC gateways for user-facing APIs.
- **The internal “Front Row Orchestrator”.**
  - Logically, a wrapper of over some etcd, so of N clusters one is always the leader.
    - Since our leader should handle far more RPS than etcd.
  - Implementation-wise, this component is also responsible for linearizing the input stream.
    - In a way, it’s “Kafka on steroids”, with one topic & one partition, but millions of TPS.
    - To accomplish this, each “message” in this event bus is just a few bytes long.
      - Thus, the front row also needs to act as a distributed in-memory storage.
      - To map cooked CQRS commands’ bodies into some 64-bit hashes.
- **The executor of the commands.**
  - This is relatively straightforward as the commands are pre-packaged.
  - Challenging sub-problems include dynamic loading of code, schema evolution, and the “ledger-like” behavior.
  - This layer includes the “infinitely accessible” “memory canvas”, with paging, LRU caching, S3 backups, etc.
- **The replication protocol and “hot standby”-s.**
  - In reality, there will be more than one “single topic single partition” linearized “Kafka” stream.
  - For data replication purposes, instead of executing each command on each node, just the “execution log” can be replicated.
    - It’s not a commands log, but also not the logical replication log; it’s a log of traces of which parts of which command should be executed.

Links

- ”The Pyramid” slides, from 2015, the origins of Current.
- An earlier version of the ideas in these slides, from ~3 years ago.
- A Miro board with a top-level view of everything, ~2 years ago.