Joker<?>

How to calculate CTR for 100M+ objects and not to die...

Dmitry Bugaychenko

@mail.ru group

OK.ru – the myths:

OK.ru – the reality:

OK.ru – the size

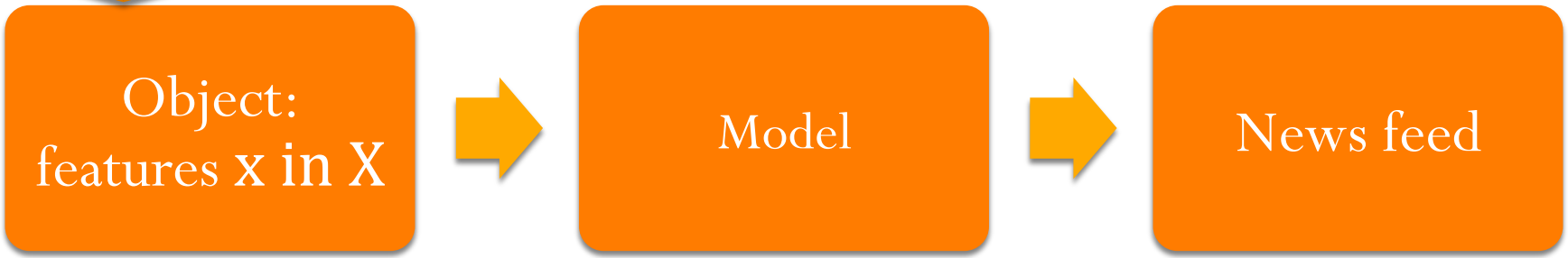
- 200 000 000 users, 12 000 000 000 social links, 10 000 000 communities…
- 8000 servers around the glob
- 1 Tb/s of traffic
- 6 TB of data for analysis daily
- …
- **9 billions news feed records daily**

What gets into your News Feed?

- Content your friends create (photos, videos, posts)
- Actions your friends make (likes, re-shares, comments)
- Content your communities create
- …
- **2000+ candidates to show instantly for an average user**

Lets add some hype on “algorithmic feed” ☺

Interesting part is here!



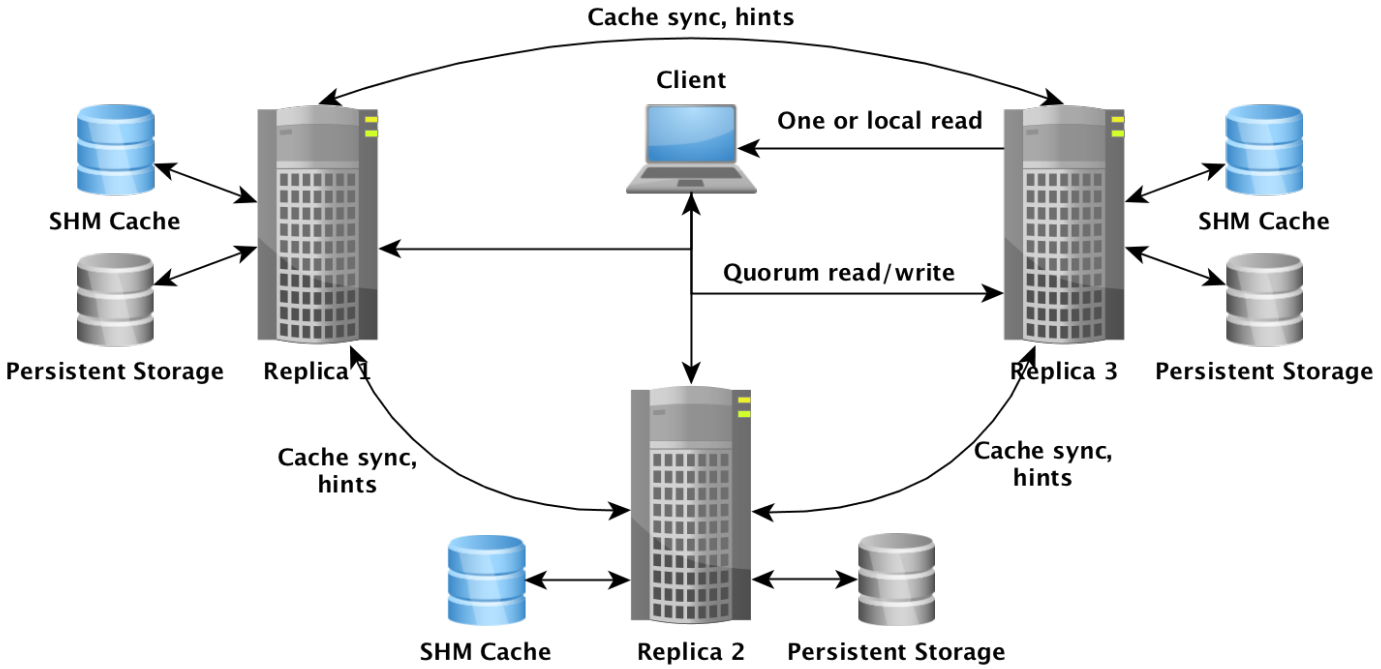
Some features

- Object age
- Number of likes
- Relations between user and author
- …
- Click Through Rate (CTR):
  - times item was clicked / times item was shown

CTR is very simple and common, but

- 800 000+ of impressions per second
- 7 000 000+ candidates to evaluate per second
- 100 000 000+ objects shown or reacted daily
- 1 333 000 000+ objects shown or reacted monthly

Typical storage at OK.ru



Read-update-write to “typical storage”?

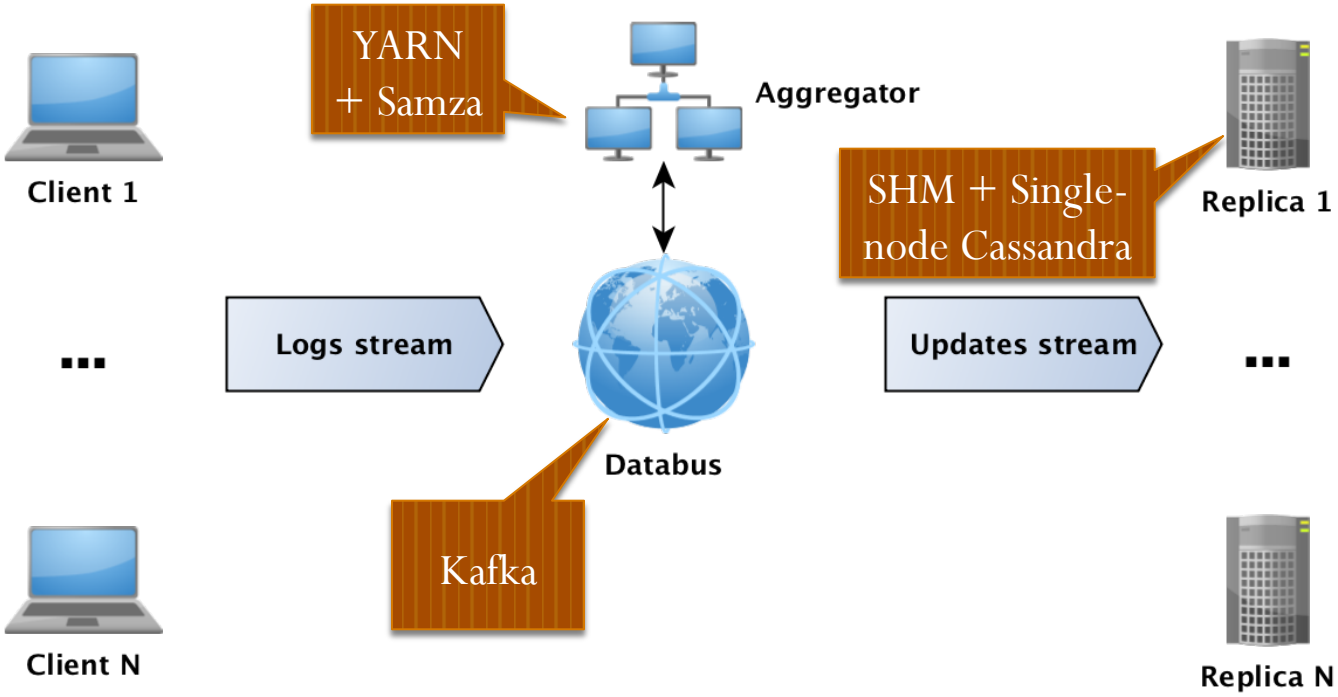
Cool, but…

- High frequency of updates
- High contention
- 10x more reads than writes

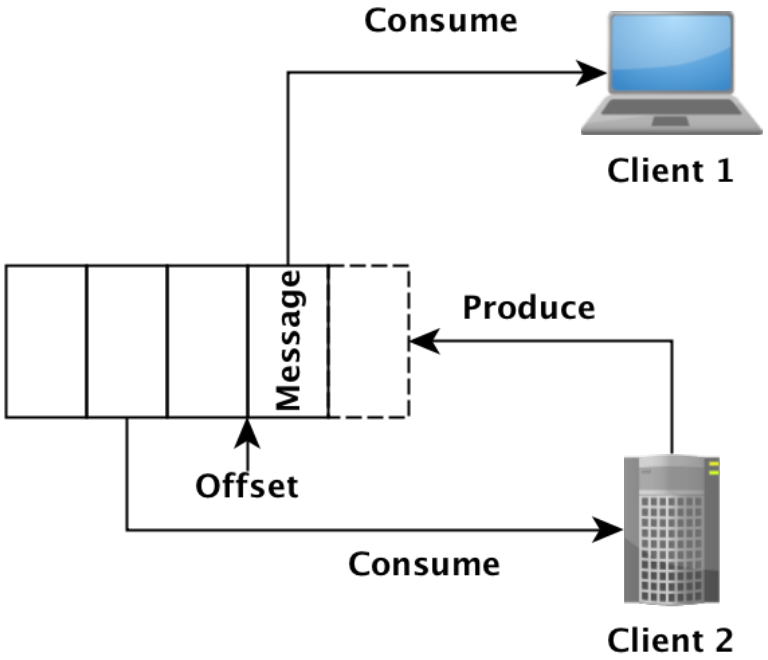
What we really want is

- Scale differently for read and write
- Eliminate contention
- Process different data with different algorithms
- 24/7 reliable connection to users

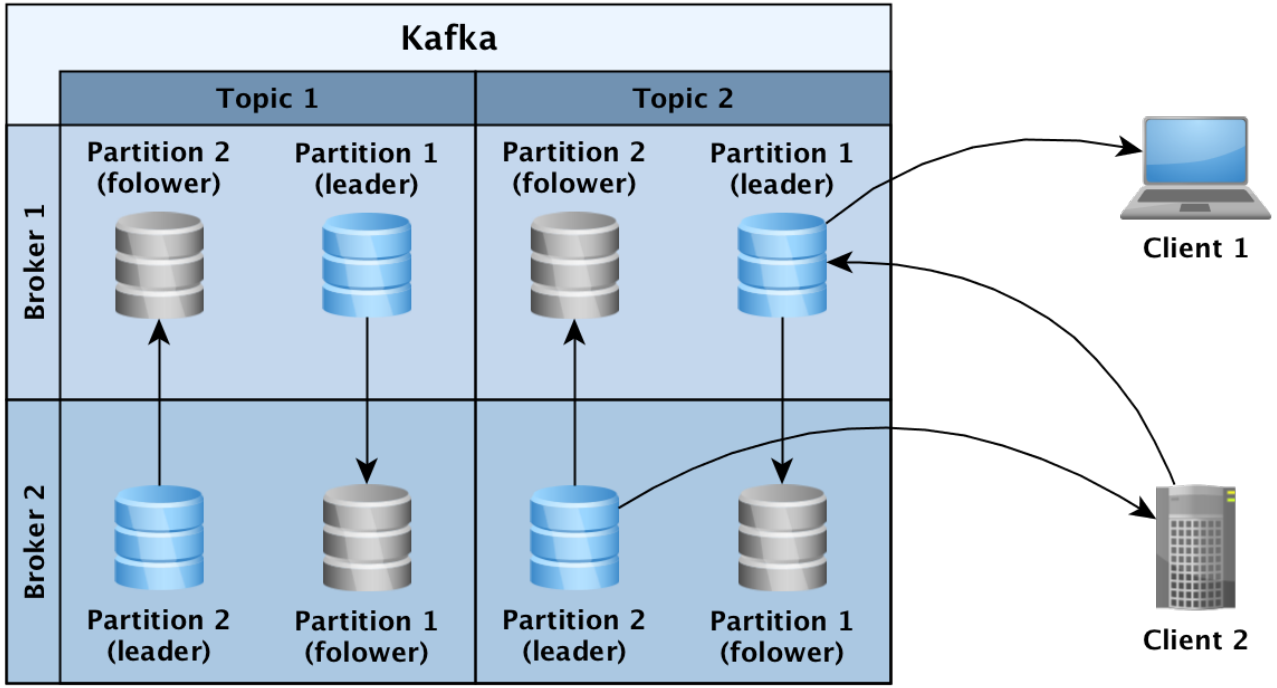
The solution: Distributed aggregator



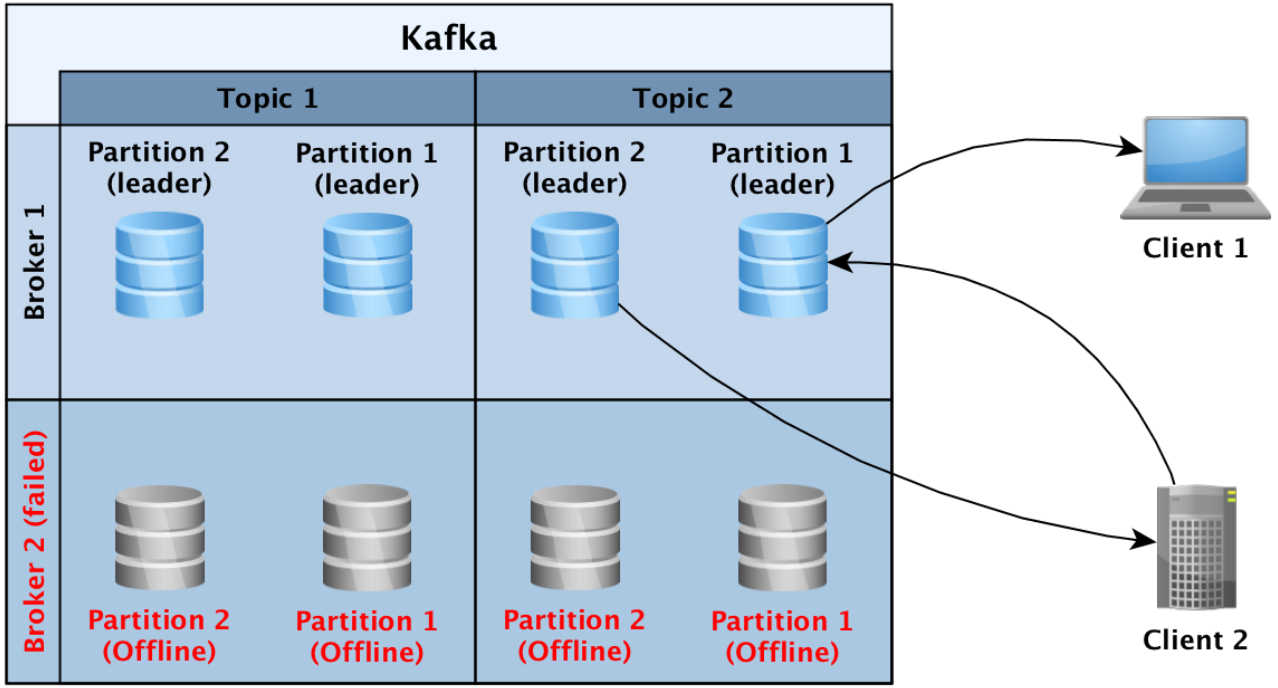
Fanout queue? What's that?



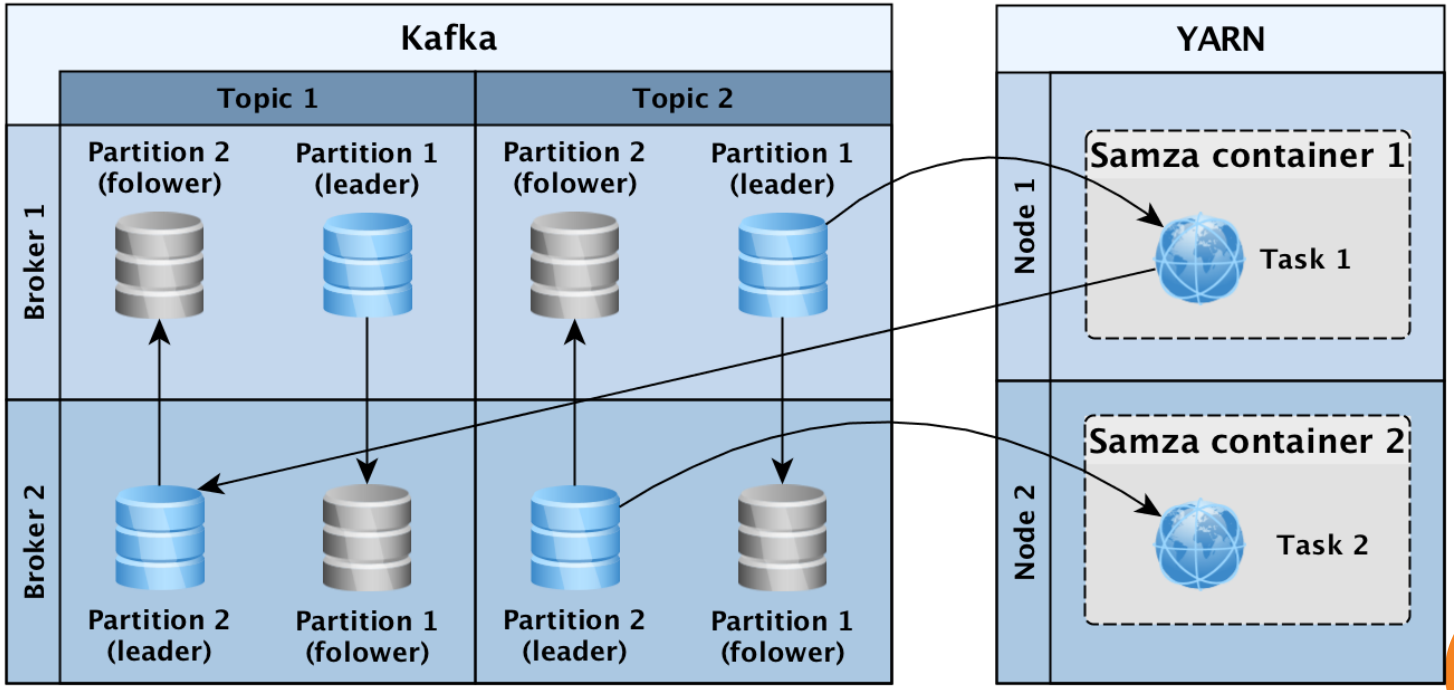
Apache Kafka



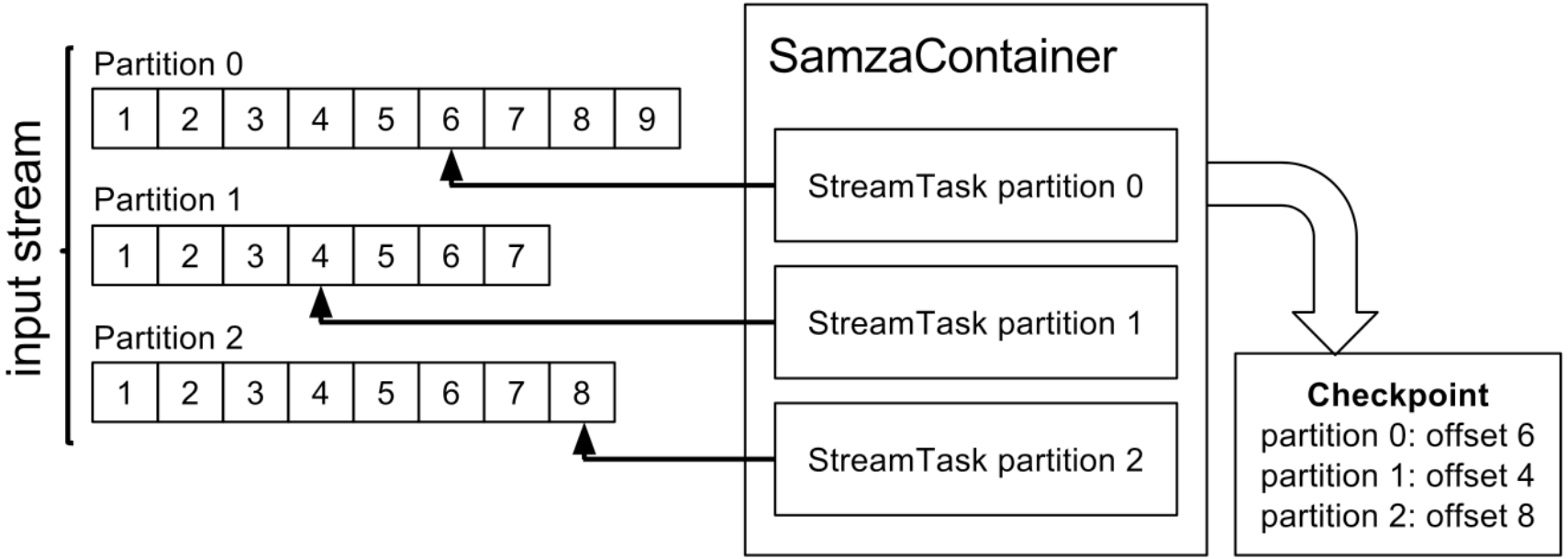
Apache Kafka broker failure



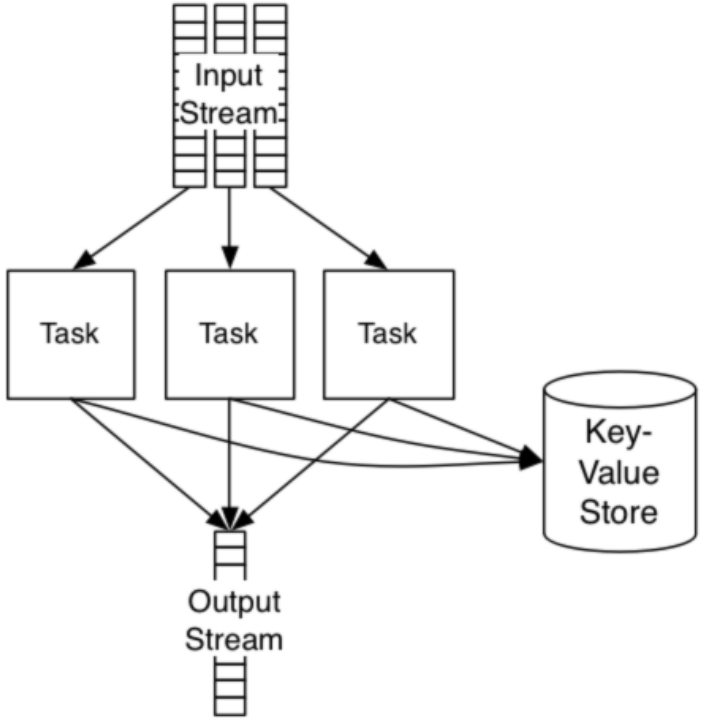
Apache Samza

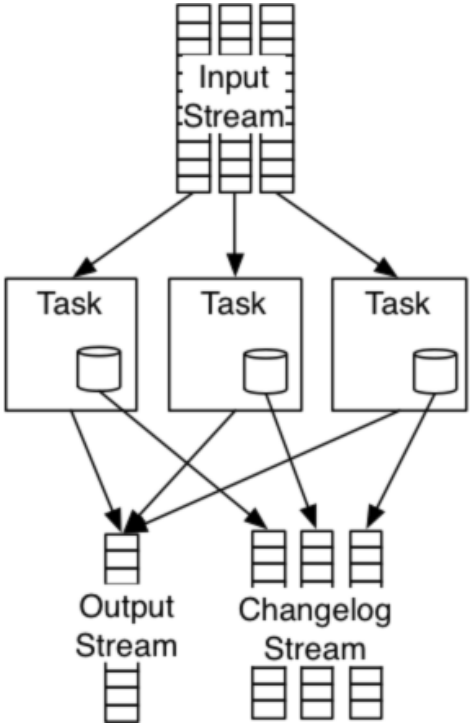


Apache Samza: Checkpointing

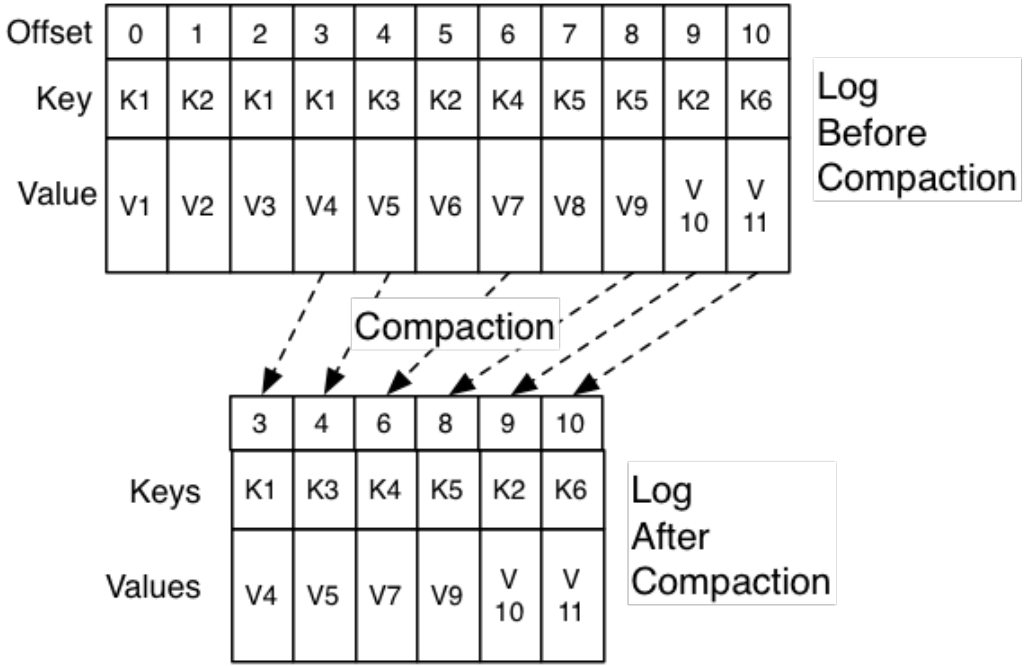


Apache Samza: State



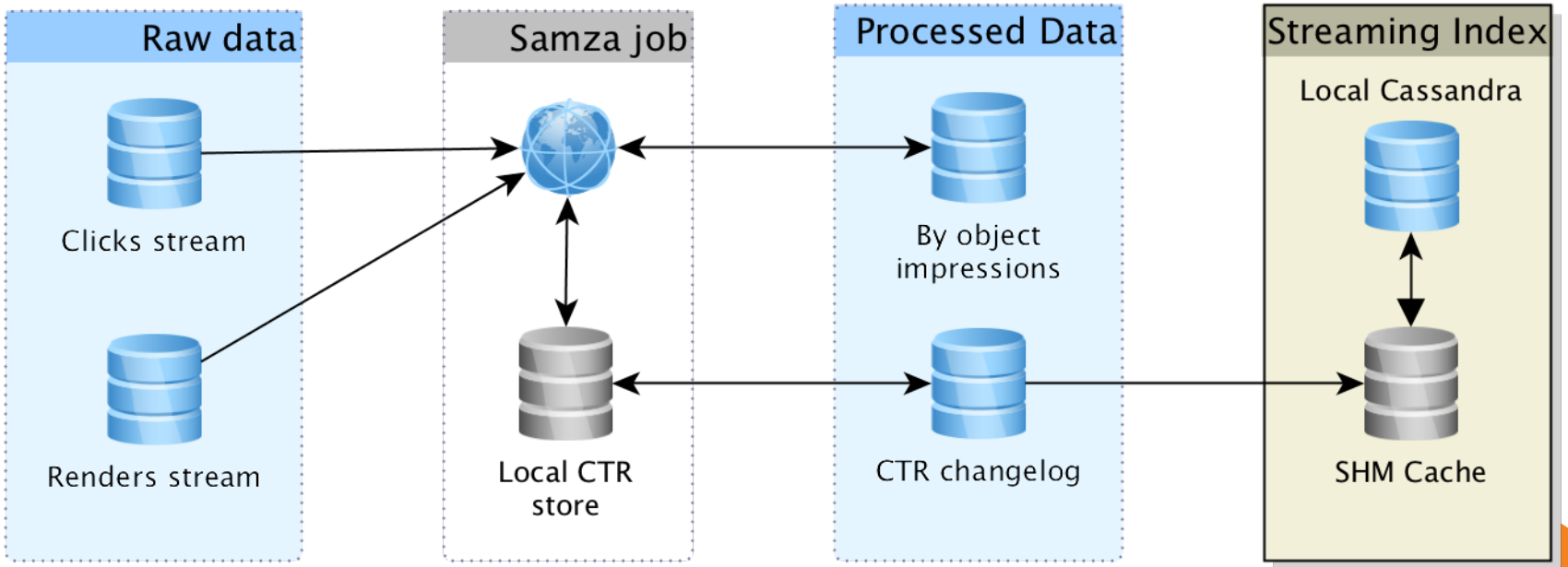


Apache Kafka: Log Compaction



Finally introducing...

Streaming CTR counter!



Did it work out of the box?

Did it work out of the box? No... ☹

Managed to fix/replace

- Multi-volume startup
- Lost watermarks
- Combined cleanup policy
- Rack-awareness
- Custom in-memory store
- **Monitoring**!

Still suffering

- Slow controlled shutdown
- Spontaneous date erasure under load
- Hanging tasks
- Can not read history from a broken disk

Monitoring

Kafka

- Broker in cluster
- All replicas on broker are in sync
- Disk errors

Samza

- Task is running
- Task is receiving messages
- Task offsets are up-to-date
- Index is receiving output of the task

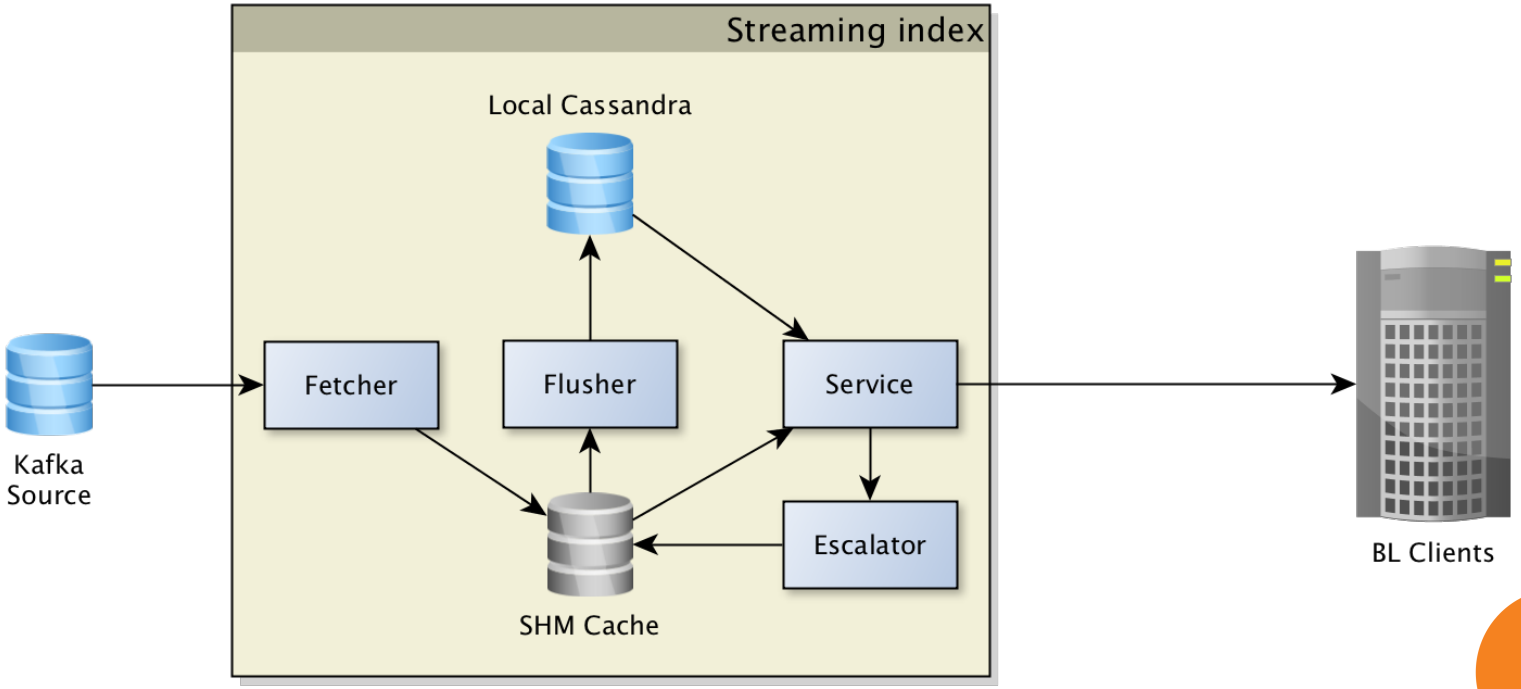
Streaming Index: The Bridge between YARN and production

- Fetches CTR values (and many more) from Kafka stream
- Stores **indexed** data in SHM cache
- Flushes updates to local Cassandra
- Servers read requests from BL clients

Streaming Index in numbers

- 100 000+ writes per node per second
- 400 000+ reads per node per second
- 700 000 000 objects in cache per node
- 3 partitions
- 6 replicas for each partition

Streaming index v1



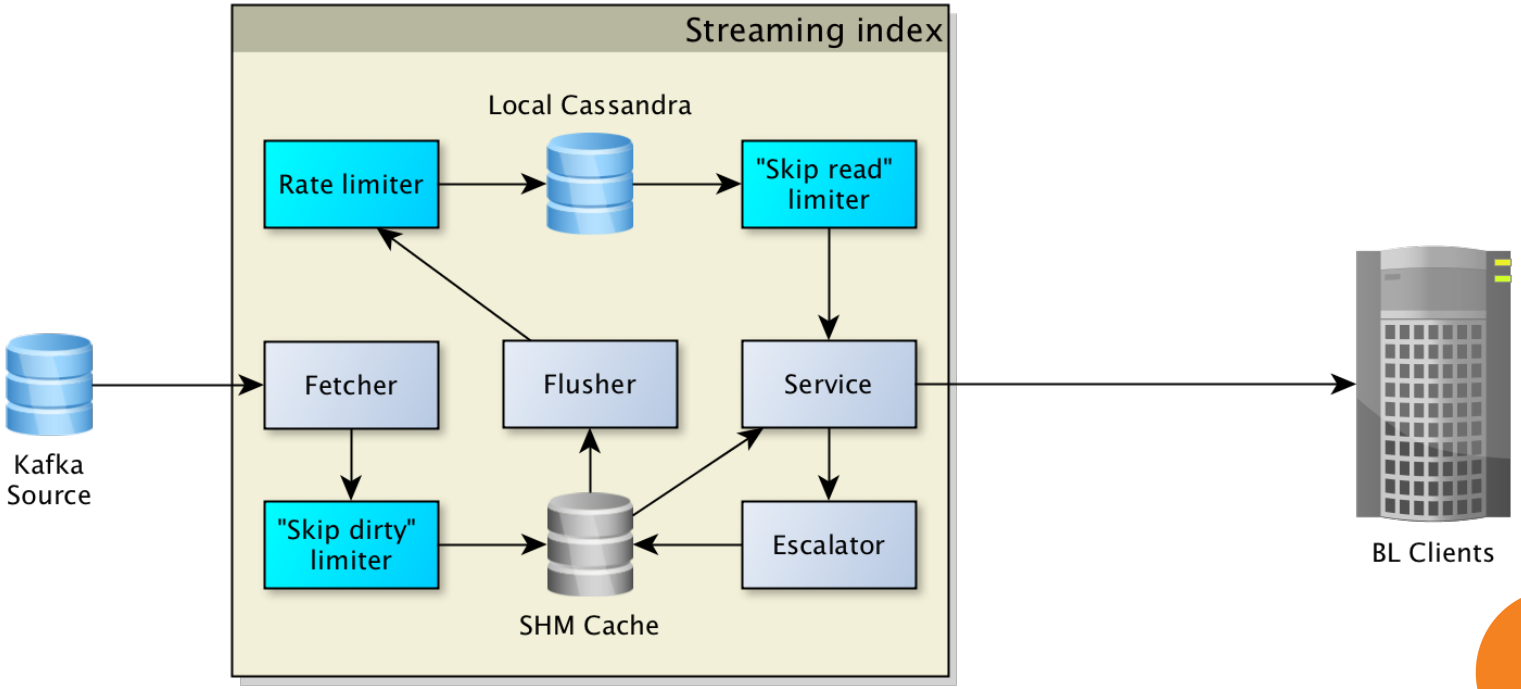
Why it didn’t worked

- Cassandra where too damn slow:
  - Huge IO for commit logs
  - Huge GC pauses (100+ ms for young gen)
  - Huge safepoint pauses
  - Full GC at the end

What have we done to fix it?

- Limited CTR commit intervals from Samza
- Disable commit log for verbose topics
- Added rate limiter for flusher
- Added “skip dirty” probability on write
- Added “skip read storage” probability on read
- Cassandra load reduced to 3333 w/s and 1000 r/s
- Data availability decreased from 97% to 96%

Streaming index v2



Client side: the wrong way

```
/**
 * Given set of objects and features to extract fetches them from the indexes
 *
 * @param objects  Objects to fetch data for.
 * @param features Features to fetch.
 * @return Map from the object ids to object features.
 */
@RemoteMethod(split = true, reduceStrategy = MapReduceFullResultsStrategy.class)
IDistributedDataWrapper<Map<ObjectId, Map<String, Object>>> getFeatures(
        @PartitionSource ObjectId[] objects, Set<String> features);
```

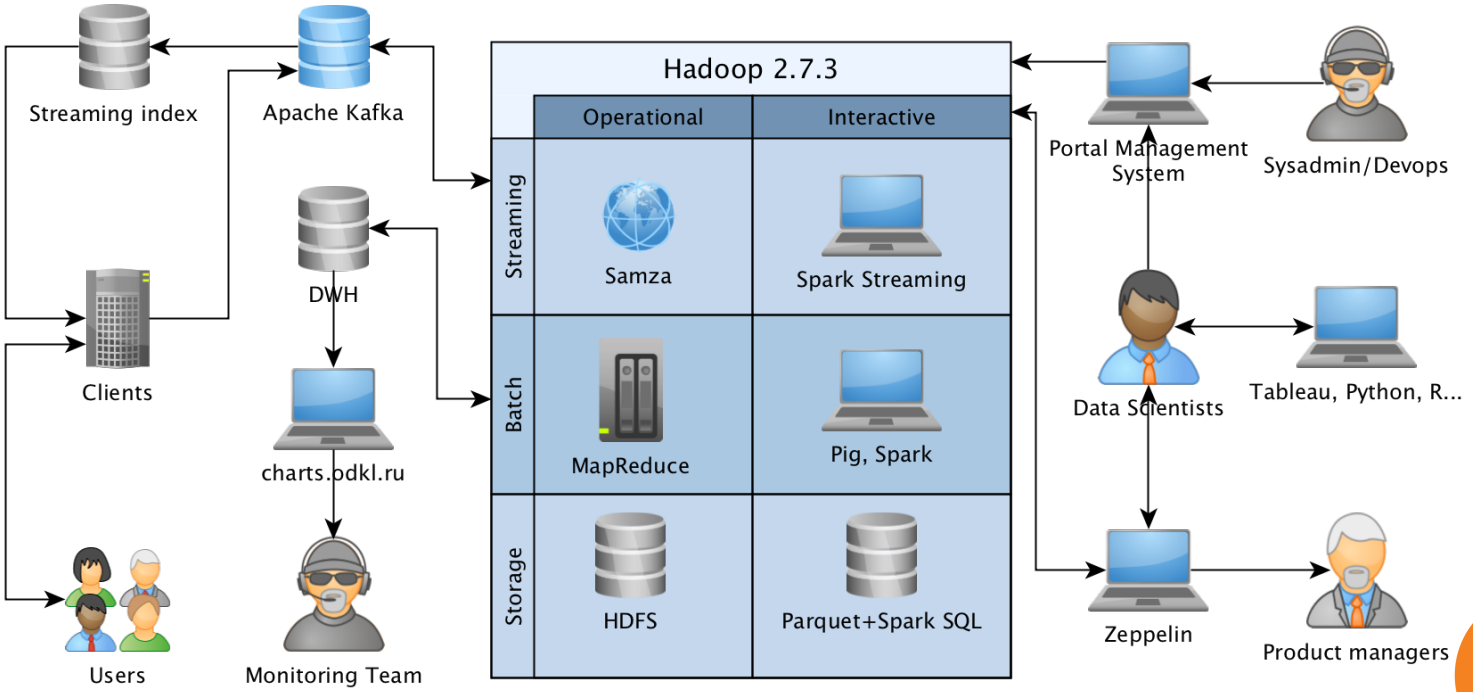
What are the problems?

- Same features for all objects
- Merging hash-maps on clients
- Hash-lookups for extracting data
- Server-side deserialization, followed by serialization and client deserialization

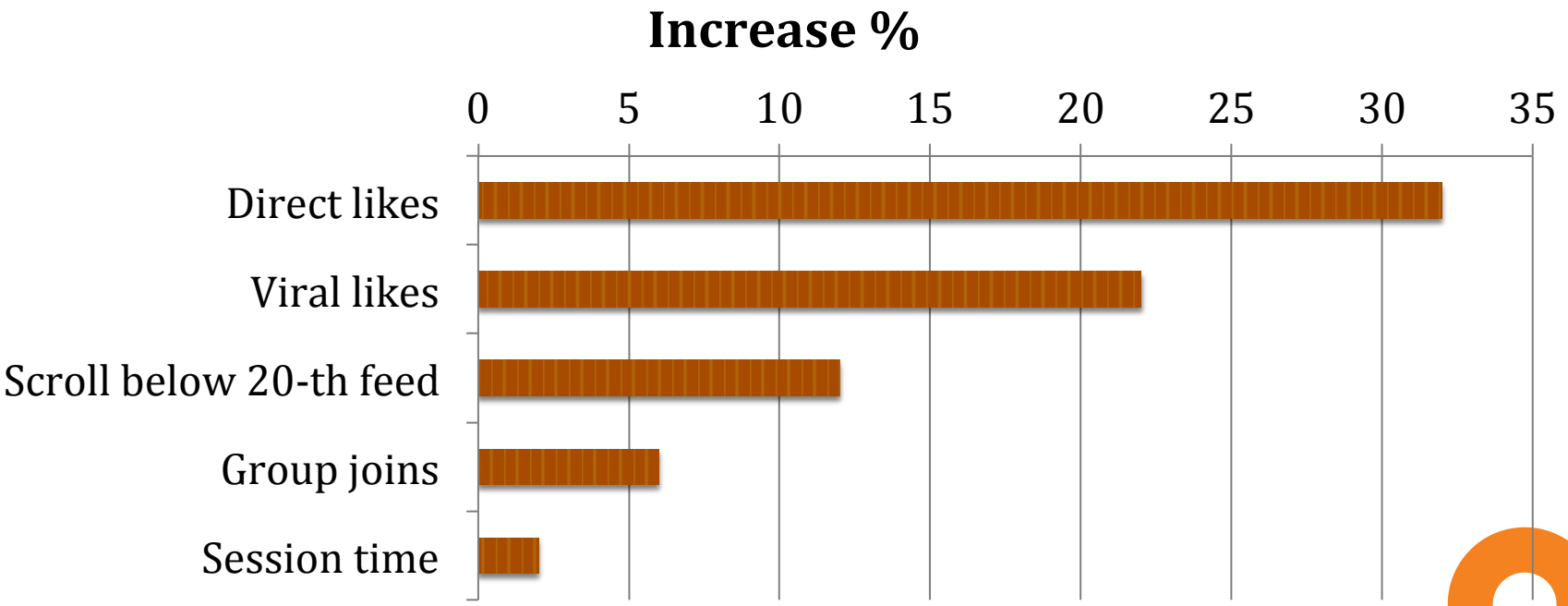
Client side: a better way

```
/**
 * Given set of object ids with attached index and features mask fetch exactly those
 * features and return result as a simple iterable.
 *
 * @param objects  Objects to fetch data with linked index and features mask.
 * @param features All features in the request. If i is the index of feature in the
 *                 list, then its mask is 1 << i. If certain object wants multiple
 *                 features their masks combined by bitwise |.
 * @return Results are returned as a collection of objects with index (matching index
 * of objects in request) each containing features identified by their indexes in
 * features array. Data ara passed as byte buffers - external serialization required.
 */
@RemoteMethod(split = true, reduceStrategy = IterablesReduceStrategy.class)
IDistributedDataWrapper<Iterable<IndexedResult>> getFeaturesWithMask(
        @PartitionSource IndexedObjectId[] objects,
        String[] features);
```

A bigger picture



Is it worth doing?



An incomplete list of streaming processing tools

Open-source

- Samza
- Storm
- Spark Streaming
- Kafka Streams
- **Flink**
- …

Proprietary

- Amazon Kinesis
- IBM InfoSphere Streams
- Azure Stream Analytics
- Oracle Stream Analytics
- TIBCO StreamBase
- …

Thank you for your attention!