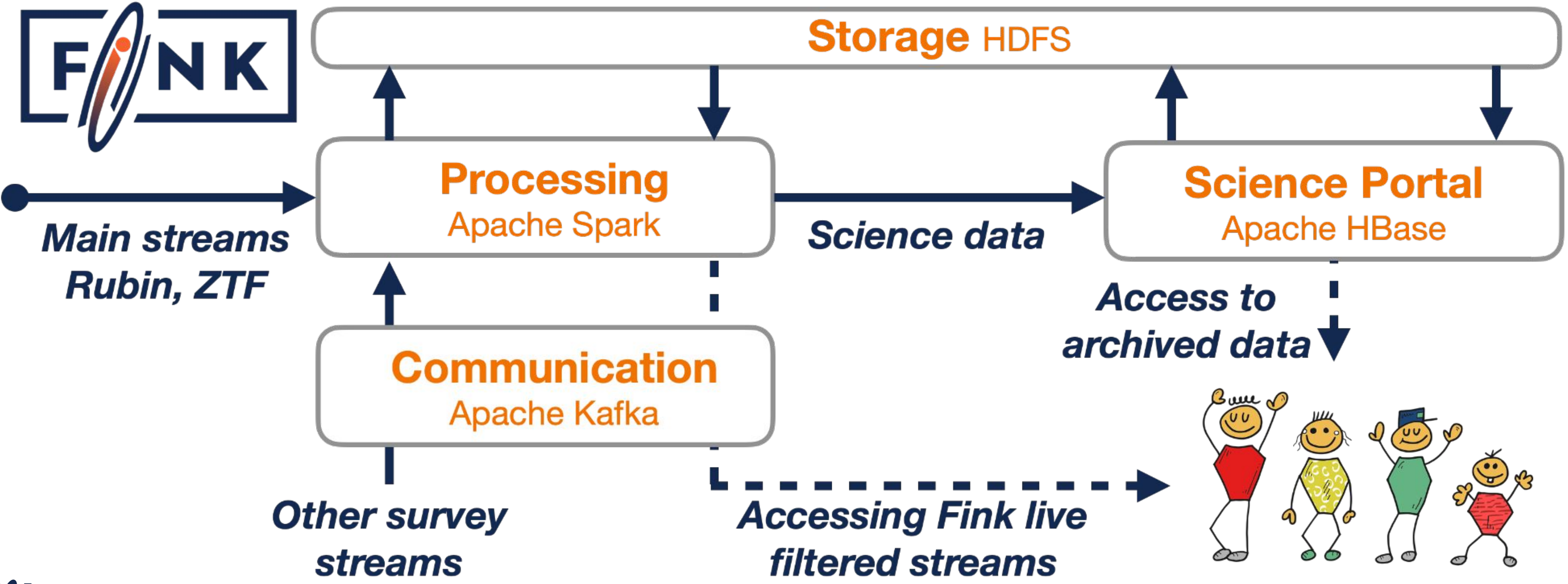
Distributed database for time-series using Apache HBase

lessons learned

Julien Peloton

2021-11-09

Fink pipeline



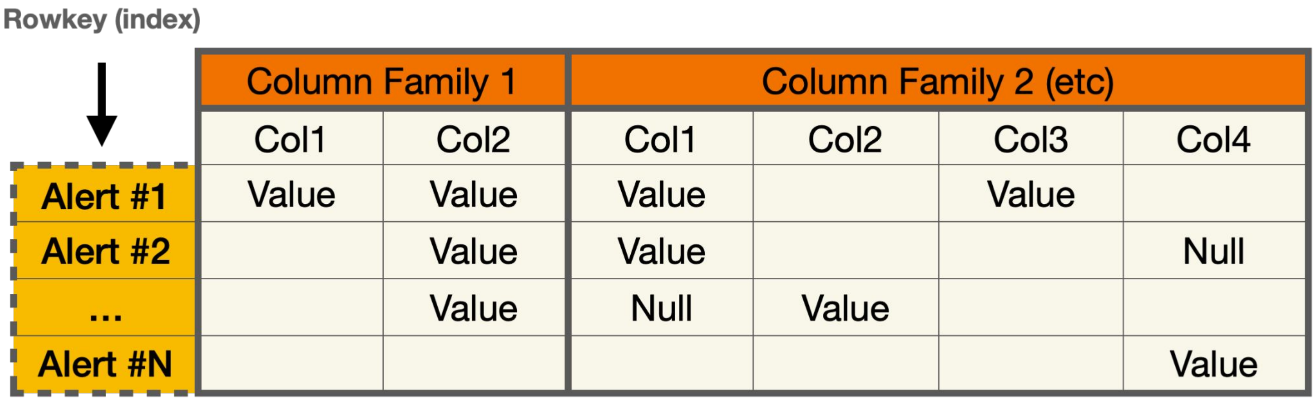
Apache HBase

- Initial release 2008
  - From Google BigTable
  - Open source
- Non-relational & distributed
- Wide column store
  - 2D key/value store
  - “3D” structure: row/col/time
- Compression, cache operations, fault-tolerance (through replication)…

HBase in Fink

- Under test since 2019 (experience comes from CERN engineers)
  - Currently using version 2.2.7 (2021-04-16). Latest stable is 2.4.8 (2021-11-03).
- Pseudo-distributed mode (1 machine x 16 cores)
  - HBase manages its own Zookeeper
- Data stored on HDFS (11 machines x 3.5 TB)
  - Used to store aggregated data.
  - Currently about 4 TB of ZTF alert data

HBase in Fink



Rowkey (index)

	Column Family 1		Column Family 2 (etc)			
	Col1	Col2	Col1	Col2	Col3	Col4
Alert #1	Value	Value	Value		Value	
Alert #2		Value	Value			Null
...		Value	Null	Value		
Alert #N						Value

Rowkey is a composite with objectId and emission time

Column family typically describe the provenance (ZTF, Fink, GW, GRB, etc.)

Columns describe fields (values can be null), e.g. objectId, magpsf, cutouts_*

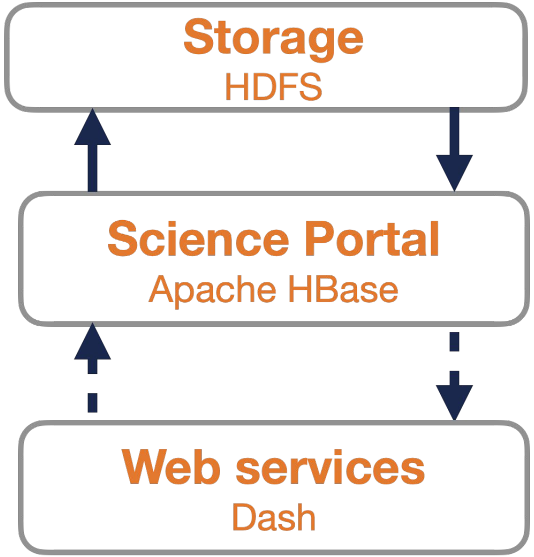
HBase in Fink

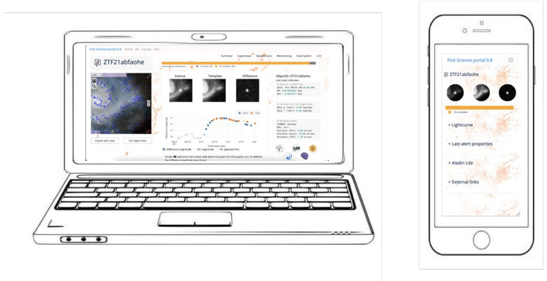
Used as the backend for our web services (incl. REST API)

Updated once a night, at the end of the observing night

- Streaming mode is not trivial (discussed later)

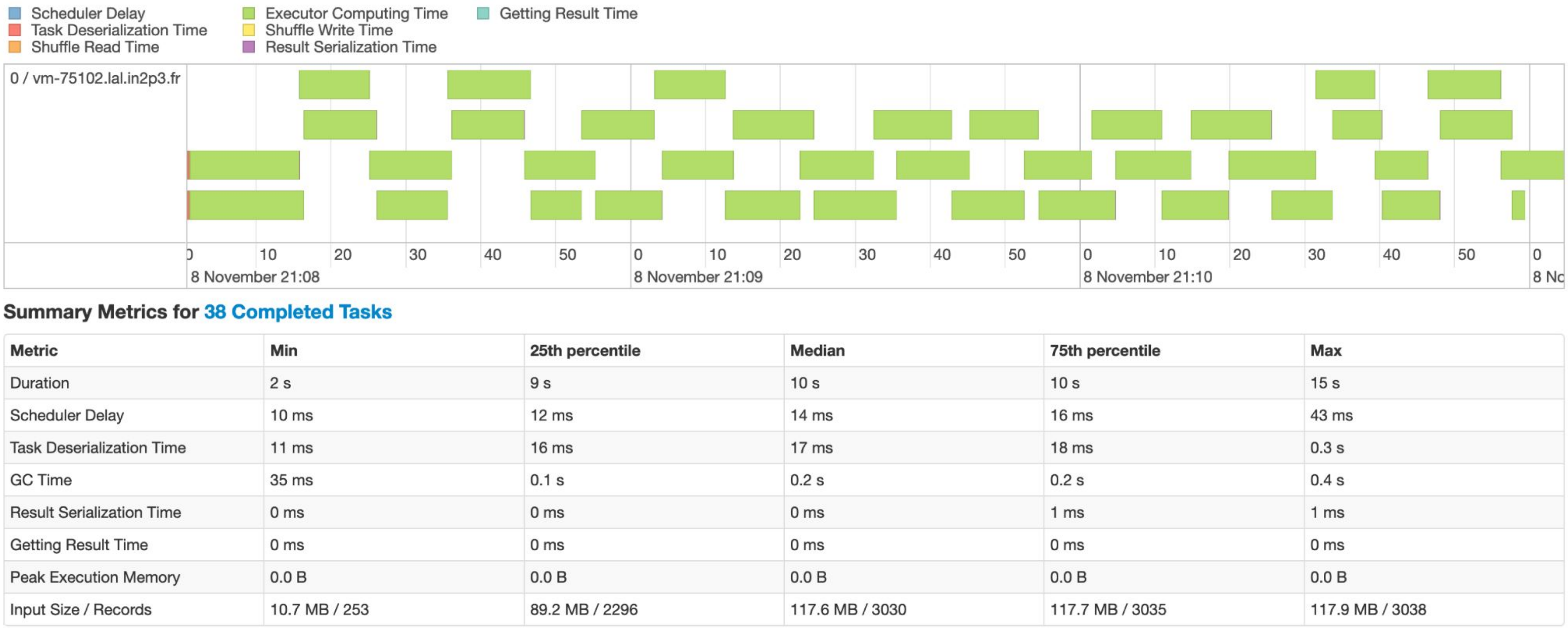
Support concurrency & can deal simultaneously with hundreds of requests without problems





Some typical numbers

Bulk load in HBase with Apache Spark (write perf per core)



Summary Metrics for 38 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	2 s	9 s	10 s	10 s	15 s
Scheduler Delay	10 ms	12 ms	14 ms	16 ms	43 ms
Task Deserialization Time	11 ms	16 ms	17 ms	18 ms	0.3 s
GC Time	35 ms	0.1 s	0.2 s	0.2 s	0.4 s
Result Serialization Time	0 ms	0 ms	0 ms	1 ms	1 ms
Getting Result Time	0 ms	0 ms	0 ms	0 ms	0 ms
Peak Execution Memory	0.0 B	0.0 B	0.0 B	0.0 B	0.0 B
Input Size / Records	10.7 MB / 253	89.2 MB / 2296	117.6 MB / 3030	117.7 MB / 3035	117.9 MB / 3038

HBase pros (1/2)

Mature and stable technology

- Developed by a small, but active set of developers

Simple to deploy

- Good old Java technology + Zookeeper on top

Despite no SQL syntax, the query language remains easy, and extending it is not difficult (e.g. user-defined functions in Java) without sacrificing too much the performances.

HBase pros (2/2)

Somehow large adoption in the community

- Many available plugins for Apache Spark, Apache Kafka, … Can serve as backend for JanusGraph, …

Schemaless database

- Easy to accommodate for alert schema evolution

Very efficient random access (on primary key)

- Less than 10ms response time with extremely modest hardware on standard queries for O(10) TB dataset

HBase cons (1/2)

Multi-index queries is not native

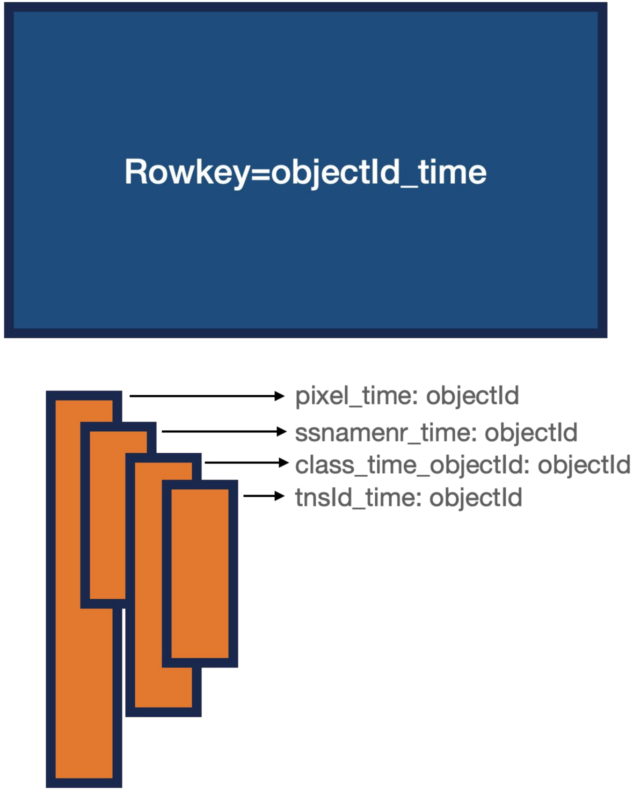
- Possible hacks: composite keys, index tables, …. But they are hacks.

Co-machin

Streaming is possible, but there is a price to pay

- “Compaction storm” can happen more often than you think…

Time travel rarely used in practice (never in production at least)



Rowkey=objectId_time

pixel_time: objectId

ssnamenr_time: objectId

class_time_objectId: objectId

tnsId_time: objectId

HBase cons (2/2)

SQL syntax can be made possible (e.g. using Phoenix) but not recommended

- Performance degradation

HBase runs on top of… HDFS. Other storages are not recommended (or just impossible to use).

Debug and optimisation can be painful

- Use of external repair tools, cryptic errors (JVM's fault)

Conclusion

Apache HBase can be used for time-series, and under some conditions it offers very good performances.

The main drawbacks to me are:

- Lack of efficient native multi-index capabilities
- Efficient streaming (write) is possible but need some hack
- Debug can be hell

We will continue to use it in Fink for time being, but other solutions are under investigation.