cloudera

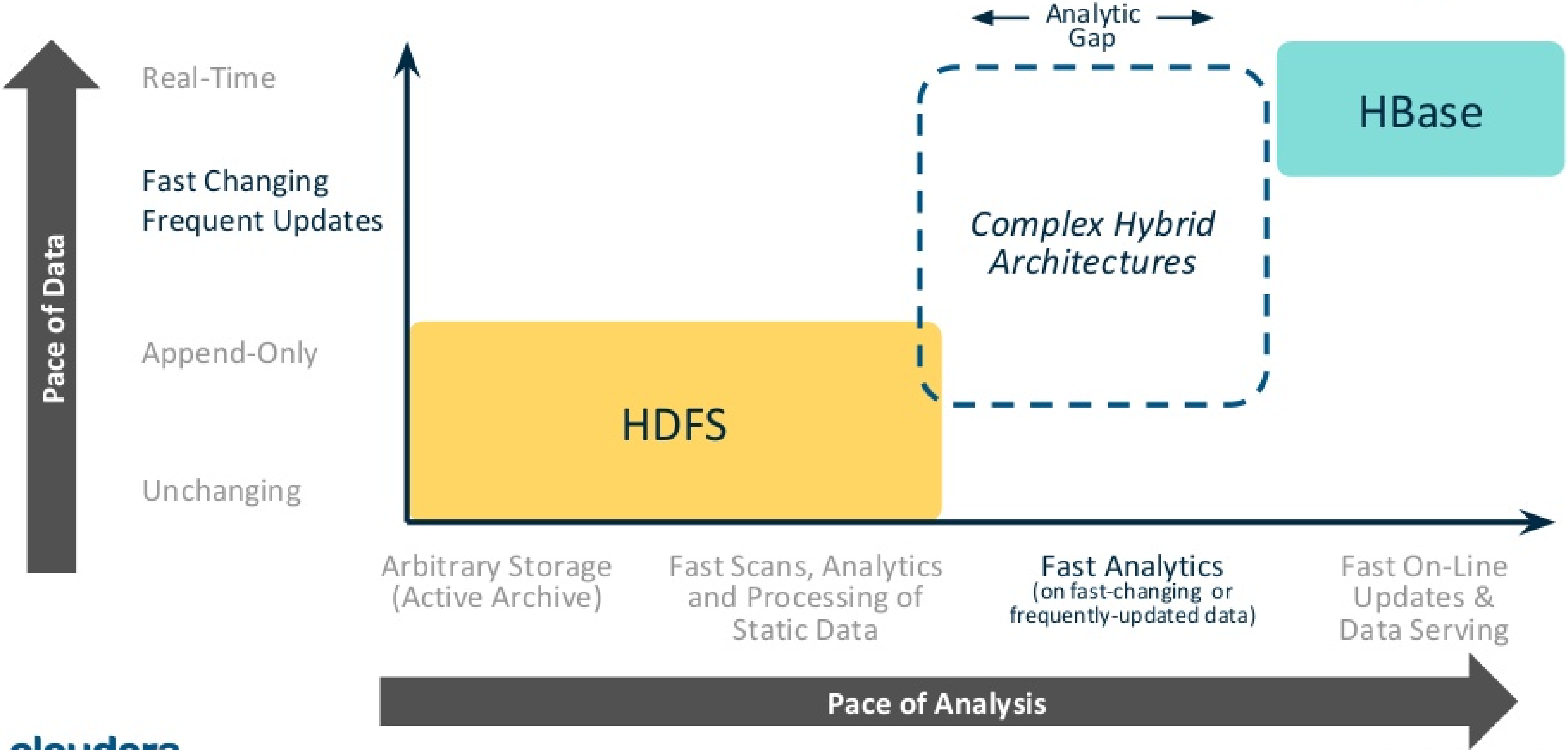
Apache Kudu & Apache Spark SQL for Fast Analytics on Fast Data

Mike Percy
Software Engineer at Cloudera
Apache Kudu PMC member

Kudu Overview

Traditional Hadoop Storage Leaves a Gap

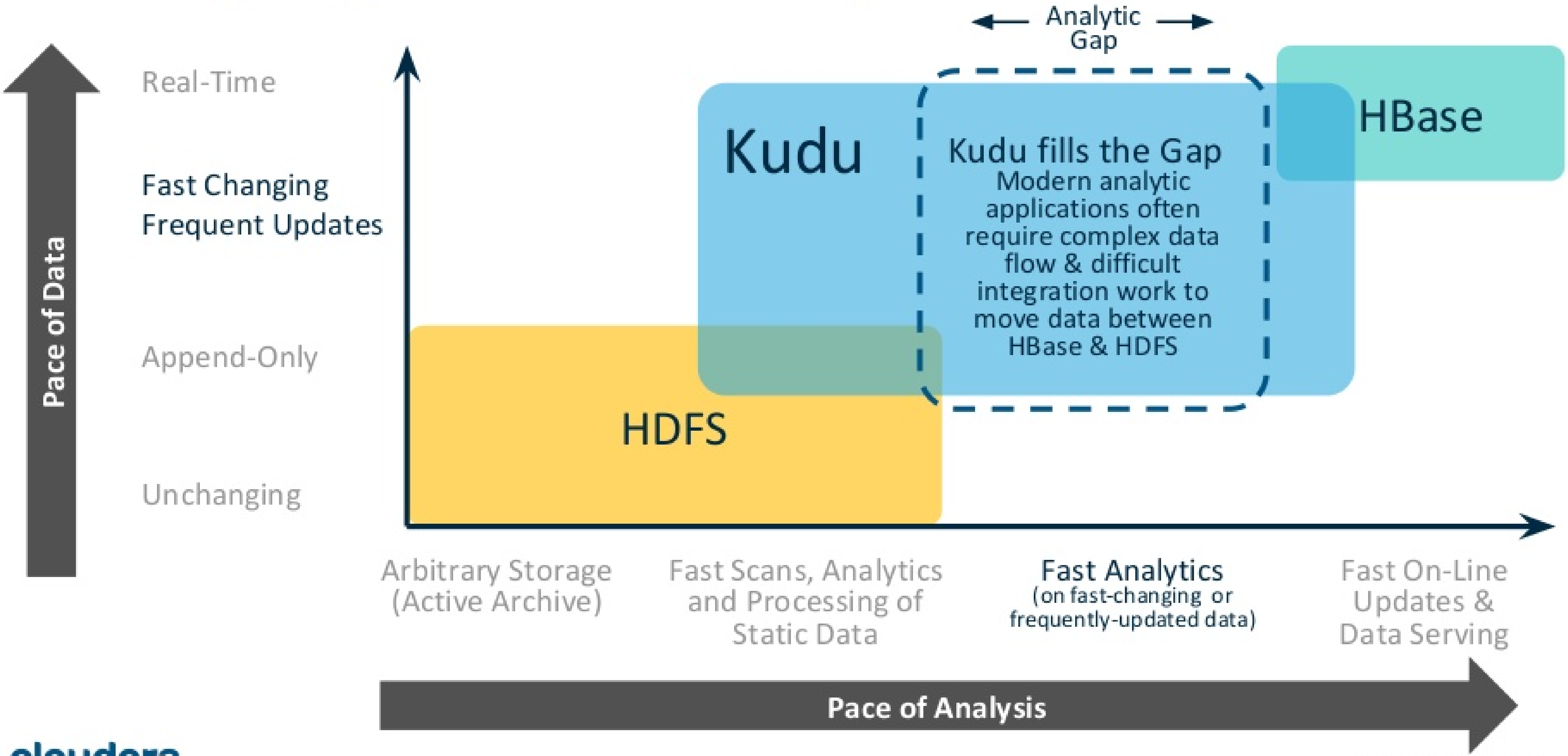
Use cases that fall between HDFS and HBase were difficult to manage



cloudera

Kudu: Fast Analytics on Fast-Changing Data

New storage engine enables new Hadoop use cases



cloudera

Apache Kudu: Scalable and fast tabular storage

Tabular

- Represents data in structured tables like a relational database
  - Strict schema, finite column count, no BLOBs
- Individual record-level access to 100+ billion row tables

Scalable

- Tested up to 275 nodes (~3PB cluster)
- Designed to scale to 1000s of nodes and tens of PBs

Fast

- Millions of read/write operations per second across cluster
- Multiple GB/second read throughput per node

cloudera

Storing records in Kudu tables

- A Kudu table has a SQL-like schema
  - And a finite number of columns (unlike HBase/Cassandra)
  - Types: BOOL, INT8, INT16, INT32, INT64, FLOAT, DOUBLE, STRING, BINARY, TIMESTAMP
  - Some subset of columns makes up a possibly-composite primary key
  - Fast ALTER TABLE
- Java, Python, and C++ NoSQL-style APIs
  - Insert(), Update(), Delete(), Scan()
- SQL via integrations with Spark and Impala
  - Community work in progress / experimental: Drill, Hive

cloudera

Kudu SQL access

- Kudu itself is just storage and native “NoSQL” APIs
- SQL access via **integrations** with Spark, Impala, etc.

Kudu "NoSQL" APIs - Writes

```
KuduTable table = client.openTable("my_table");
KuduSession session = client.newSession();
Insert ins = table.newInsert();
ins.getRow().addString("host", "foo.example.com");
ins.getRow().addString("metric", "load-avg.1sec");
ins.getRow().addDouble("value", 0.05);
session.apply(ins);
session.flush();
```

cloudera

Kudu “NoSQL” APIs - Reads

```
KuduScanner scanner = client.newScannerBuilder(table)
   .setProjectedColumnNames(List.of(“value”))
   .build();
while (scanner.hasMoreRows()) {
  RowResultIterator batch = scanner.nextRows();
  while (batch.hasNext()) {
    RowResult result = batch.next();
    System.out.println(result.getDouble(“value”));
  }
}
```

cloudera

Kudu “NoSQL” APIs - Predicates

```
KuduScanner scanner = client.newScannerBuilder(table)
  .addPredicate(KuduPredicate.newComparisonPredicate(
    table.getSchema().getColumn("timestamp"),
    ComparisonOp.GREATER,
    System.currentTimeMillis() / 1000 + 60))
  .build();
```

Note: Kudu can evaluate simple predicates, but no aggregations, complex expressions, UDFs, etc.

Tables and tablets

- Each table is horizontally partitioned into tablets
  - *Range* or *hash* partitioning
    - `PRIMARY KEY (host, metric, timestamp) DISTRIBUTE BY HASH(timestamp) INTO 100 BUCKETS`
    - Translation: `bucketNumber = hashCode(row[‘timestamp’]) % 100`
- Each tablet has N replicas (3 or 5), kept consistent with Raft consensus
- Tablet servers host tablets on local disk drives

Fault tolerance

- Operations replicated using Raft consensus
    - Strict quorum algorithm. See Raft paper for details
- Transient failures:
    - Follower failure: Leader can still achieve majority
    - Leader failure: automatic leader election (~5 seconds)
    - Restart dead TS within 5 min and it will rejoin transparently
- Permanent failures
    - After 5 minutes, automatically creates a new follower replica and copies data
- N replicas can tolerate up to (N-1)/2 failures

Metadata

- Replicated master
  - Acts as a tablet directory
  - Acts as a catalog (which tables exist, etc)
  - Acts as a load balancer (tracks TS liveness, re-replicates under-replicated tablets)
- Caches all *metadata* in RAM for high performance
- Client configured with master addresses
  - Asks master for tablet locations as needed and caches them

Integrations

Kudu is designed for integrating with higher-level compute frameworks

Integrations exist for:

- Spark
- Impala
- MapReduce
- Flume
- Drill

What Kudu brings to Spark

- Parquet-like scan performance with 0-delay inserts and updates
- Push down predicate filters for fast & efficient scans
- Primary key indexing for fast point lookups (compared to Parquet)

Spark DataSource

Spark DataFrame/DataSource integration

```
// spark-shell --packages org.apache.kudu:kudu-spark_2.10:1.0.0
// Import kudu datasource
import org.kududb.spark.kudu._
val kuduDataFrame = sqlContext.read.options(
    Map("kudu.master" -> "master1,master2,master3",
        "kudu.table" -> "my_table_name")).kudu

// Then query using Spark data frame API
kuduDataFrame.select("id").filter("id" >= 5).show()
// (prints the selection to the console)

// Or register kuduDataFrame as a table and use Spark SQL
kuduDataFrame.registerTempTable("my_table")
sqlContext.sql("select id from my_table where id >= 5").show()
// (prints the sql results to the console)
```

cloudera

Quick demo

cloudera

Writing from Spark

```
// Use KuduContext to create, delete, or write to Kudu tables
val kuduContext = new KuduContext("kudu.master:7051")

// Create a new Kudu table from a dataframe schema
// NB: No rows from the dataframe are inserted into the table
kuduContext.createTable("test_table", df.schema, Seq("key"),
                        new CreateTableOptions().setNumReplicas(1))

// Insert, delete, upsert, or update data
kuduContext.insertRows(df, "test_table")
kuduContext.deleteRows(sqlContext.sql("select id from kudu_table where id >= 5"),
                       "kudu_table")
kuduContext.upsertRows(df, "test_table")
kuduContext.updateRows(df.select("id", $"count" + 1, "test_table")
```

cloudera

Spark DataSource optimizations

Column projection and predicate pushdown

- Only read the referenced columns
- Convert 'WHERE' clauses into Kudu predicates
- Kudu predicates automatically convert to primary key scans, etc

```
scala> sqlContext.sql("select avg(value) from metrics where host =
'e1103.halxg.cloudera.com'").explain
== Physical Plan ==
TungstenAggregate(key=[], functions=[(avg(value#3),mode=Final,isDistinct=false)], output=[_c0#94])
+- TungstenExchange SinglePartition, None
   +- TungstenAggregate(key=[], functions=[(avg(value#3),mode=Partial,isDistinct=false)],
                        output=[sum#98,count#99L])
      +- Project [value#3]
         +- Scan org.apache.kudu.spark.kudu.KuduRelation@e13cc49[value#3]
              PushedFilters: [EqualTo(host,e1103.halxg.cloudera.com)]
```

cloudera

Spark DataSource optimizations

Partition pruning

```
scala> df.where("host like 'foo%'").rdd.partitions.length
res1: Int = 20
scala> df.where("host = 'foo'").rdd.partitions.length
res2: Int = 1
```

cloudera

Use cases

Kudu use cases

Kudu is best for use cases requiring:

- Simultaneous combination of sequential and random reads and writes
- Minimal to zero data latencies

Time series

- Examples: Streaming market data; fraud detection & prevention; network monitoring
- Workload: Inserts, updates, scans, lookups

Online reporting / data warehousing

- Example: Operational Data Store (ODS)
- Workload: Inserts, updates, scans, lookups

Xiaomi use case

- World's 4th largest smart-phone maker (most popular in China)
- Gather important RPC tracing events from mobile app and backend service.
- Service monitoring & troubleshooting tool.

High write throughput

- >20 Billion records/day and growing

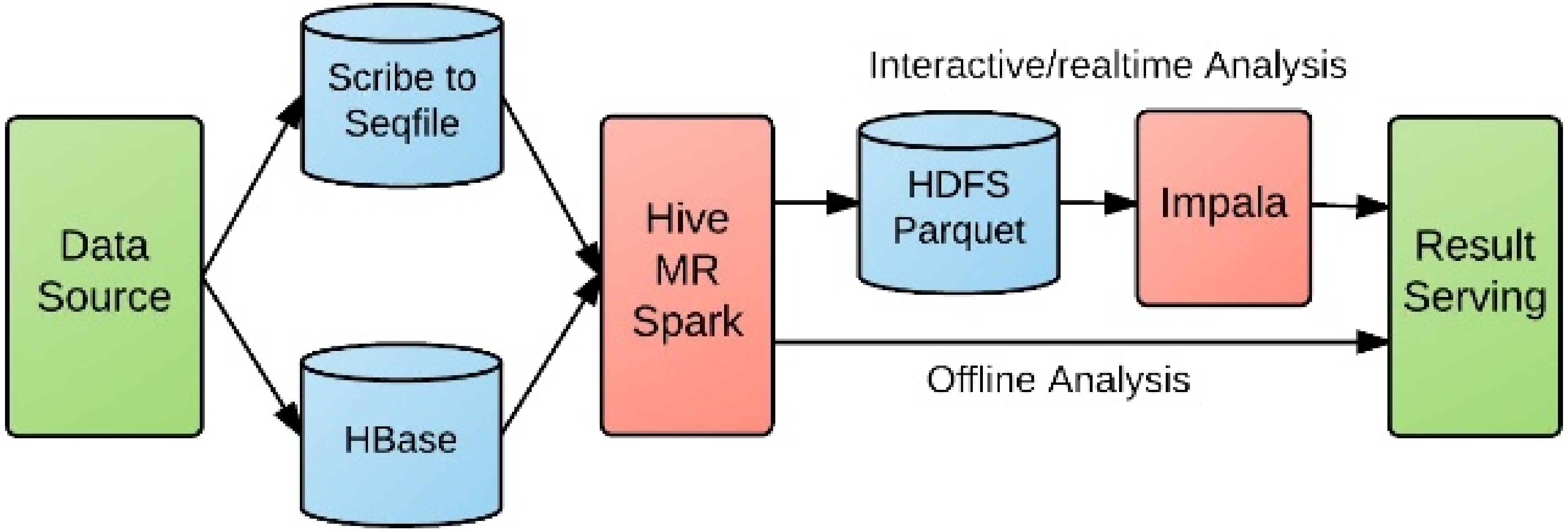
Query latest data and quick response

- Identify and resolve issues quickly

Can search for individual records

- Easy for troubleshooting

Xiaomi big data analytics pipeline
Before Kudu



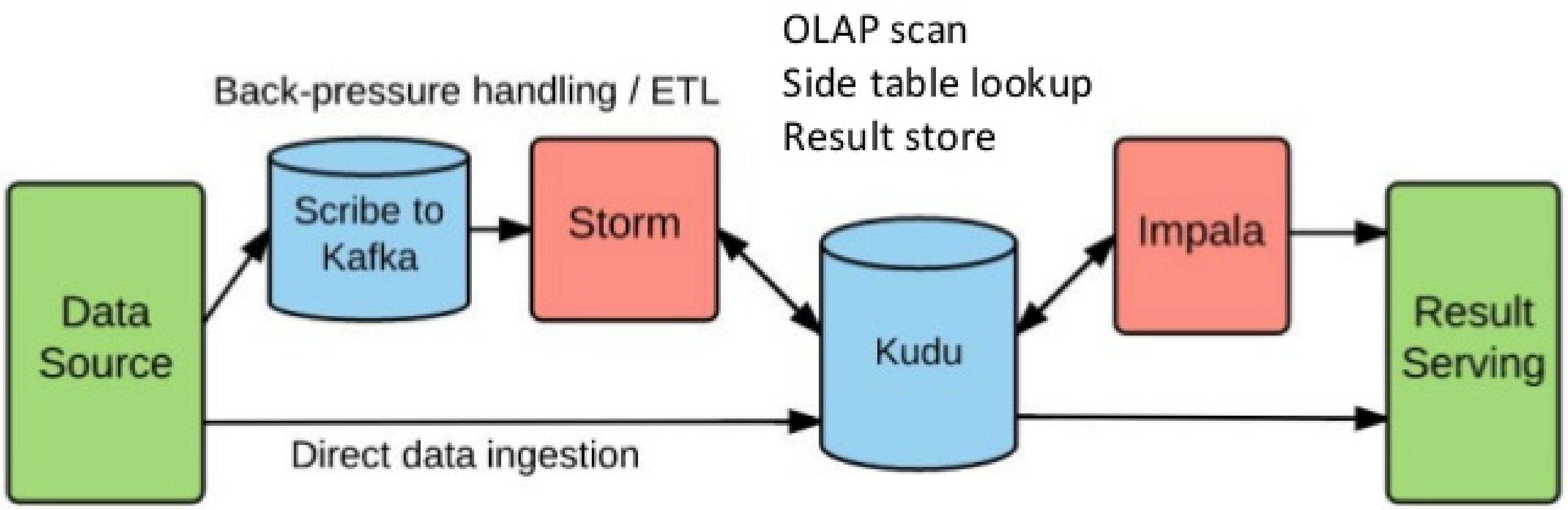
Long pipeline

- High data latency (approx 1 hour – 1 day)
- Data conversion pains

No ordering

- Log arrival (storage) order is not exactly logical order
- Must read 2 – 3 days of data to get all of the data points for a single day

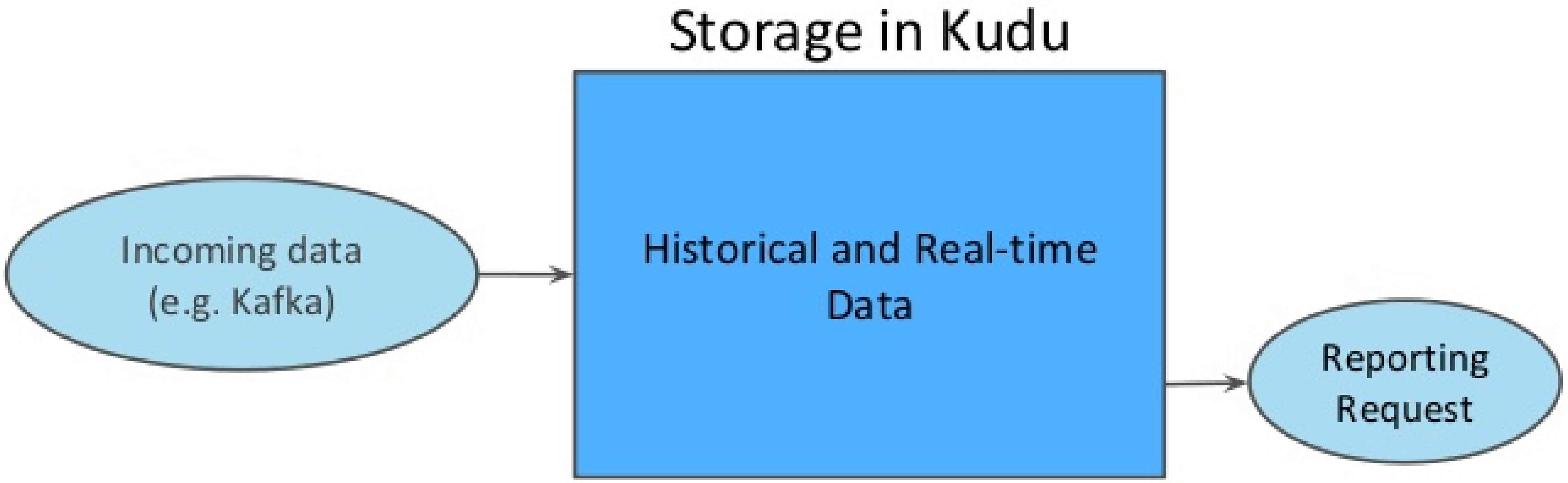
Xiaomi big data analytics pipeline
Simplified with Kafka and Kudu



ETL pipeline
- 0 – 10s data latency
- Apps that need to avoid backpressure or need ETL

Direct pipeline (no latency)
- Apps that don't require ETL or backpressure handling

Real-time analytics in Hadoop with Kudu



Improvements:

- One system to operate
- No cron jobs or background processes
- Handle late arrivals or data corrections with ease
- New data available immediately for analytics or operations

Kudu+Impala vs MPP DWH

Commonalities

- ✓ Fast analytic queries via SQL, including most commonly used modern features
- ✓ Ability to insert, update, and delete data

Differences

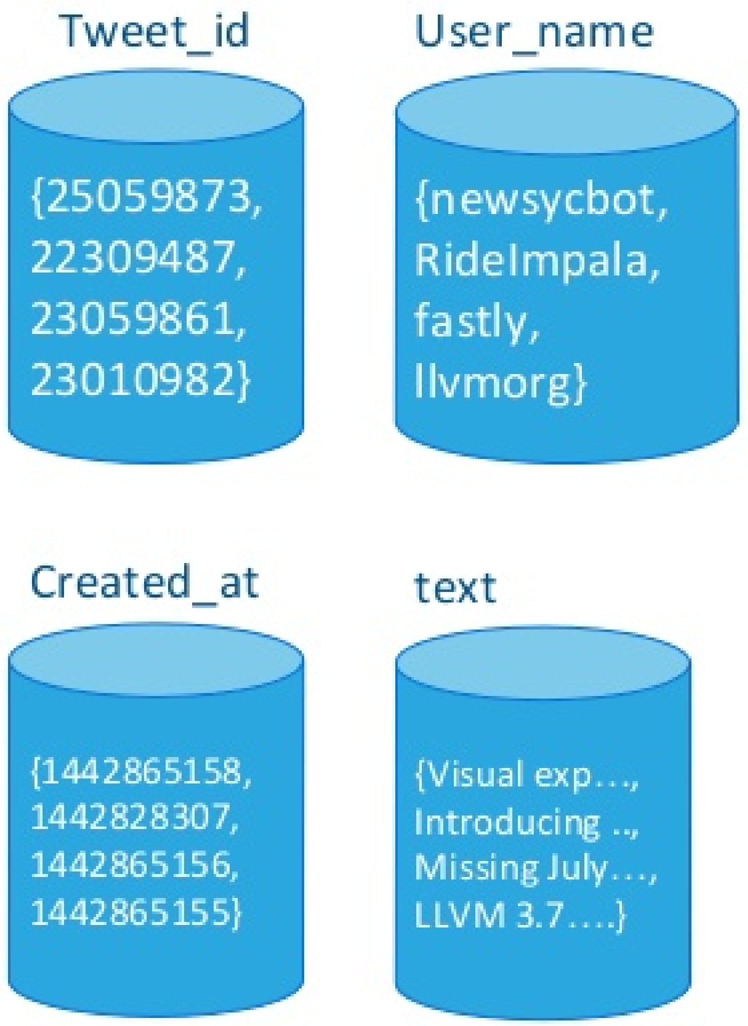
- ✓ Faster streaming inserts
- ✓ Improved Hadoop integration
  - JOIN between HDFS + Kudu tables, run on same cluster
  - Spark, Flume, other integrations
- ✗ Slower batch inserts
- ✗ No transactional data loading, multi-row transactions, or indexing

cloudera

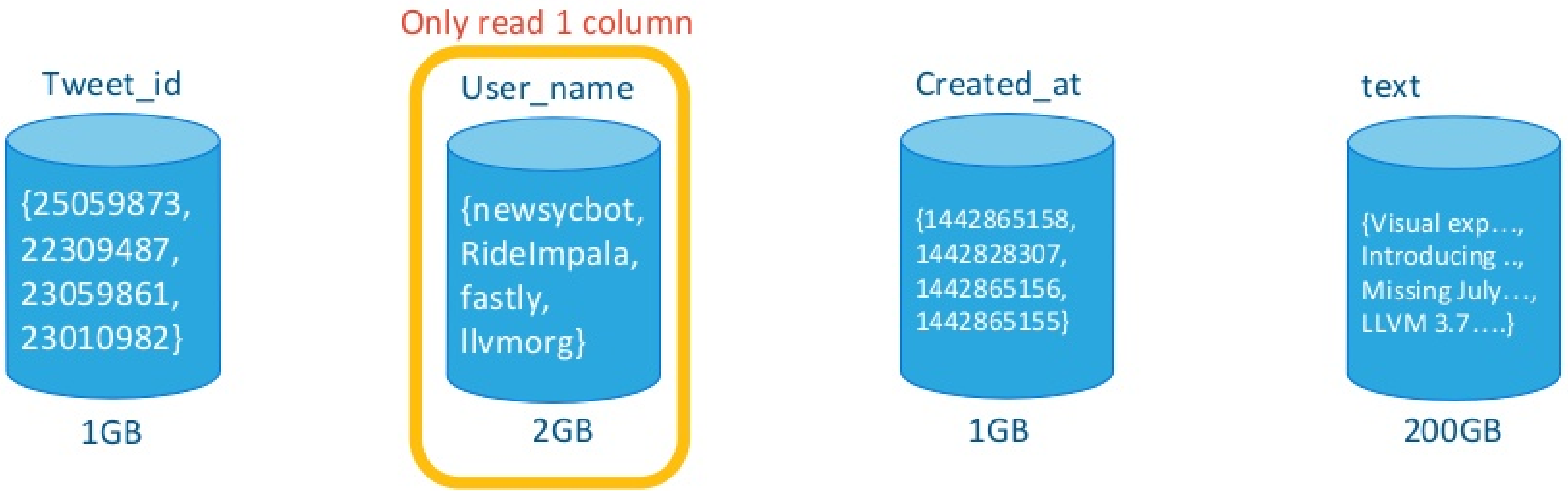
Columnar storage

Twitter Firehose Table

tweet_id	user_name	created_at	text
INT64	STRING	TIMESTAMP	STRING
23059873	newsycbot	1442865158	Visual Explanation of the Raft Consensus Algorithm http://bit.ly/1DOUac0 (cmts http://bit.ly/1HKmjfc)
22309487	RideImpala	1442828307	Introducing the Ibis project: for the Python experience at Hadoop Scale
23059861	fastly	1442865156	Missed July's SF @papers_we_love? You can now watch @el_bhs talk about @google's globally-distributed database: http://fastly.us/1eVz8MM
23010982	llvmorg	1442865155	LLVM 3.7 is out! Get it while it's HOT! http://llvm.org/releases/download.html#3.7.0



Columnar storage



SELECT COUNT(*) FROM tweets WHERE user_name = ‘newsycbot’;

Columnar compression

- **Many columns can compress to a few bits per row!**
- Especially:
  - Timestamps
  - Time series values
  - Low-cardinality strings

- **Massive space savings and throughput increase!**

Kudu Roadmap

Open Source “Roadmaps”?

- Kudu is an open source ASF project
- ASF governance means there is no guaranteed roadmap
  - Whatever people contribute is the roadmap!

- But I can speak to what my team will be focusing on
- **Disclaimer:** quality-first mantra = fuzzy timeline commitments

Security Roadmap

1) Kerberos **authentication**
   a) Client-server mutual authentication
   a) Server-server mutual authentication
   b) Execution framework-server authentication (delegation tokens)
2) Extra-coarse-grained **authorization**
   a) Likely a cluster-wide “allowed user list”
3) **Group/role mapping**
   a) LDAP/Unix/etc
4) **Data exposure hardening**
   a) e.g. ensure that web UIs dont leak data
5) **Fine-grained authorization**
   a) Table/database/column level

cloudera

Operability

1) **Stability**
   a) Continued stress testing, fault injection, etc
   b) Faster and safer recovery after failures
2) **Recovery tools**
   a) Repair from *minority* (eg if two hosts explode simultaneously)
   b) Replace from *empty* (eg if three hosts explode simultaneously)
   c) Repair file system state after power outage
3) Easier **problem diagnosis**
   a) Client "timeout" errors
   b) Easier performance issue diagnosis

Performance and scale

- **Read performance**
  - Dynamic predicates (aka runtime filters)
  - Spark statistics
  - Additional filter pushdown (e.g. “IN (...)”, “LIKE”)
  - I/O scheduling from spinning disks
- **Write performance**
  - Improved bulk load capability
- **Scalability**
  - Users planning to run 400 node clusters
  - Rack-aware placement

Client improvements roadmap

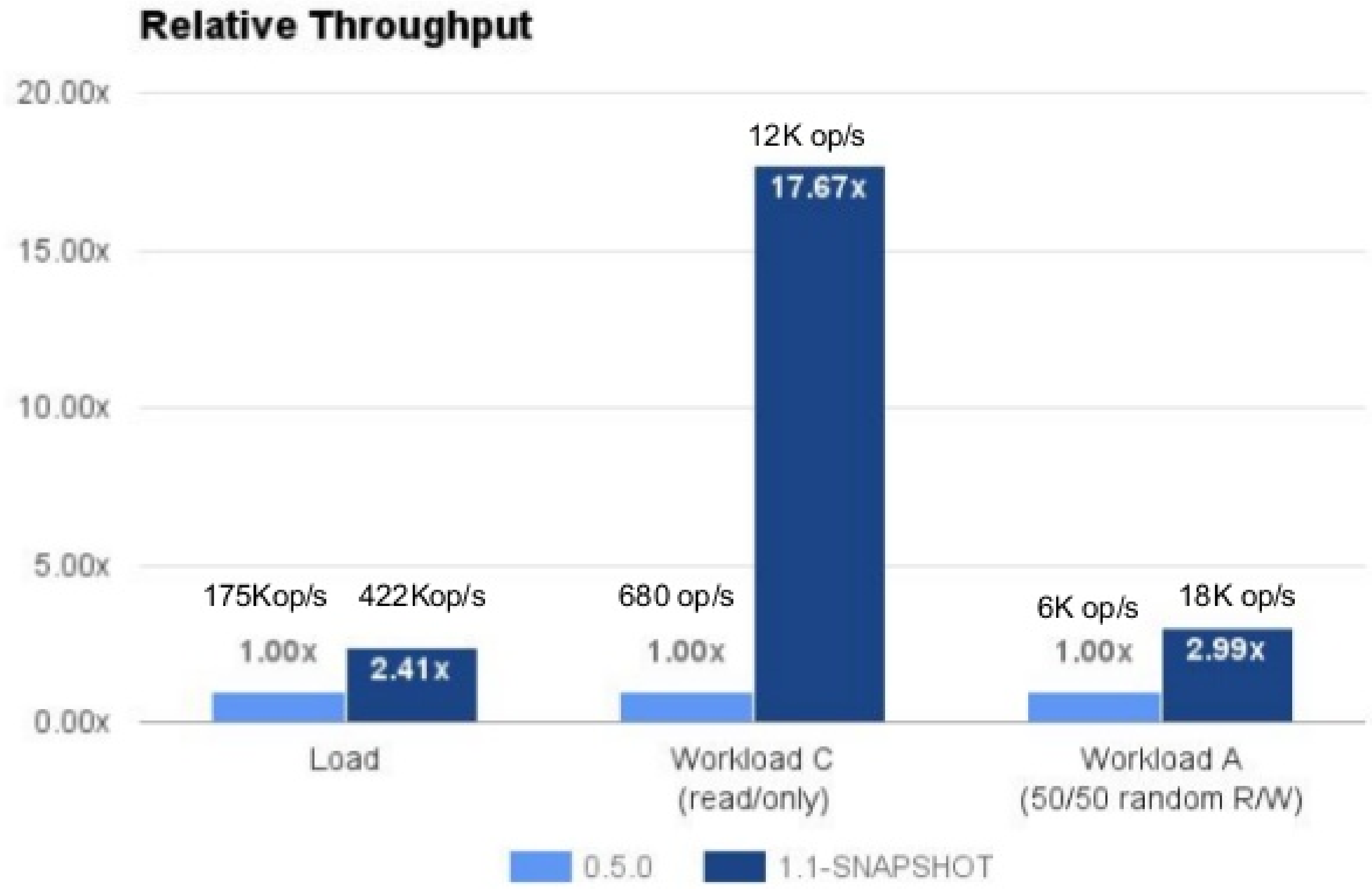
- **Python**
  - Full feature parity with C++ and Java
  - Even more pythonic
  - Integrations: Pandas, PySpark
- **All clients:**
  - More API documentation, tutorials, examples
  - Better error messages/exceptions
  - Full support for snapshot consistency level

Performance

- Significant improvements to compaction throughput
- Default configurations tuned for much less write amplification
- Reduced lock contention on RPC system, block cache
- 2-3x improvement for selective filters on dictionary-encoded columns

- Other speedups:
    - **Startup time** 2-3x better (more work coming)
    - **First scan** following restart ~2x faster
    - More compact and compressed **internal index storage**

Performance (YCSB)



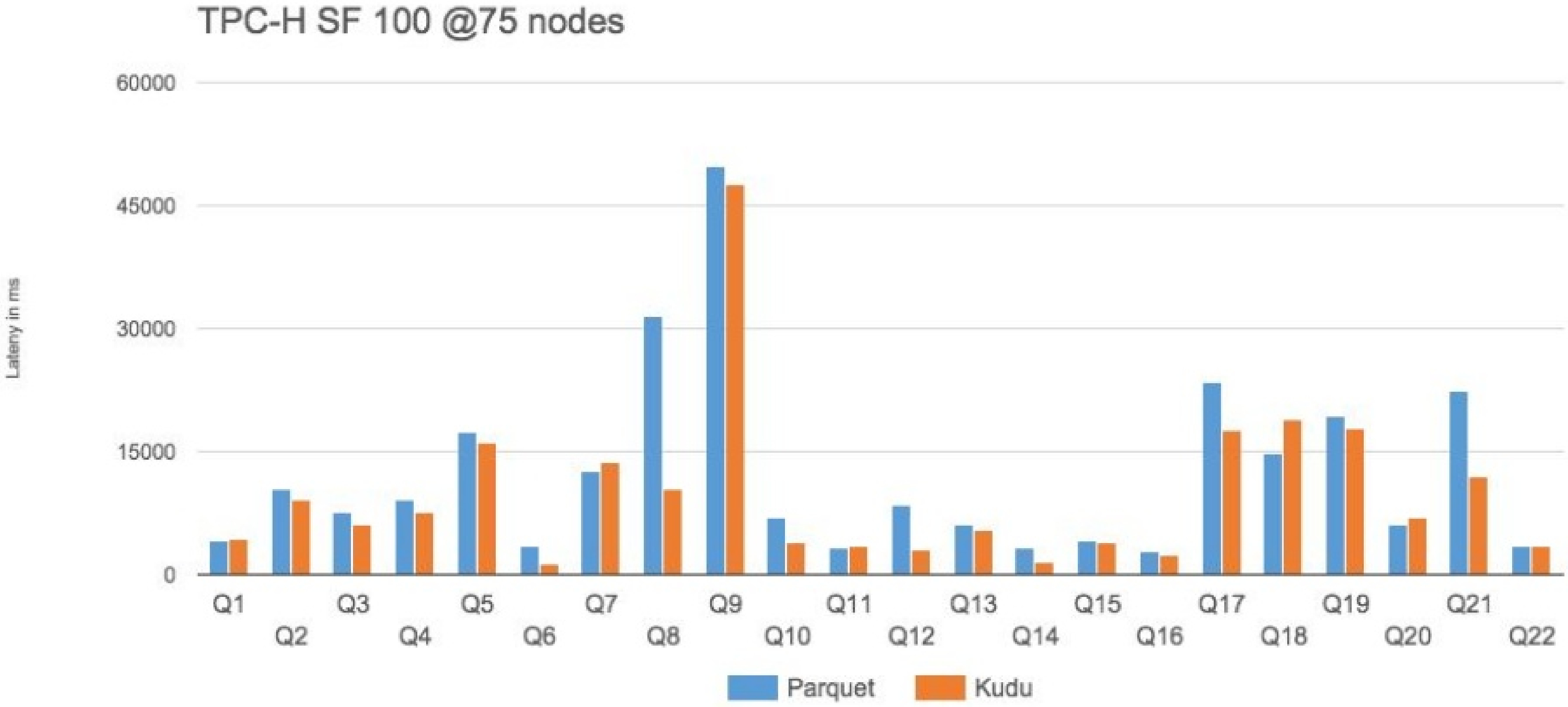
Single node micro-benchmark, 500M record insert, 16 client threads, each record ~110 bytes
Runs Load, followed by 'C' (10min), followed by 'A' (10min)

cloudera

TPC-H (analytics benchmark)

- 75 server cluster
  - 12 (spinning) disks each, enough RAM to fit dataset
  - TPC-H Scale Factor 100 (100GB)
- Example query:
  - `SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue FROM customer, orders, lineitem, supplier, nation, region WHERE c_custkey = o_custkey AND l_orderkey = o_orderkey AND l_suppkey = s_suppkey AND c_nationkey = s_nationkey AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey AND r_name = 'ASIA' AND o_orderdate >= date '1994-01-01' AND o_orderdate < '1995-01-01’ GROUP BY n_name ORDER BY revenue desc;`

cloudera

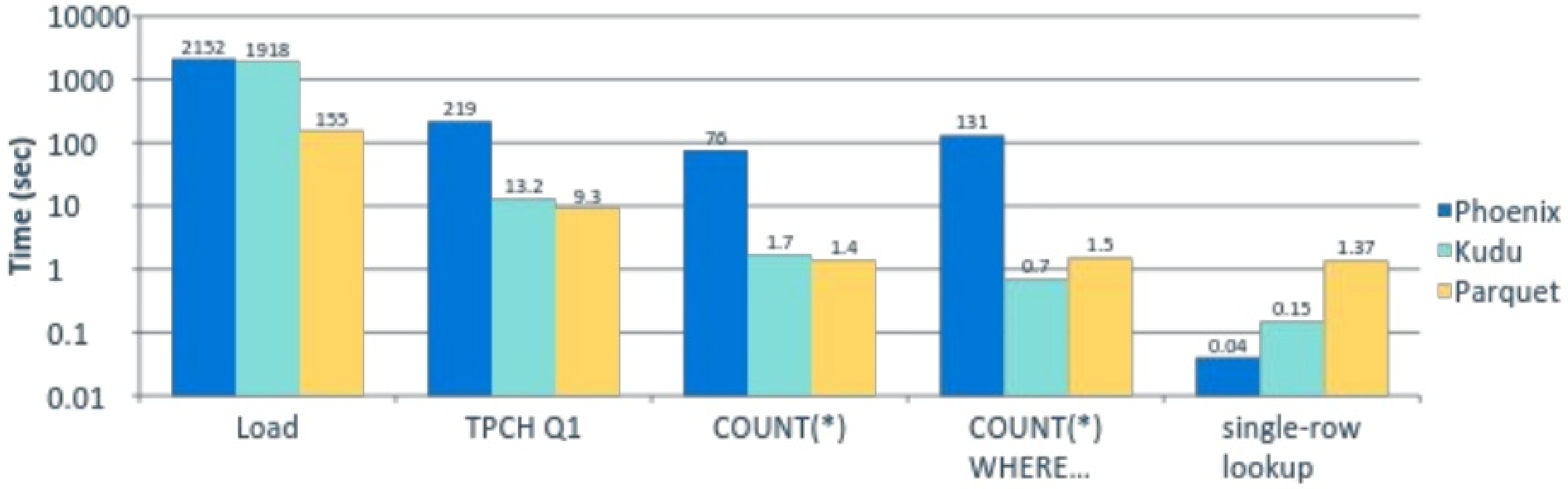


- Kudu outperforms Parquet by 31% (geometric mean) for RAM-resident data

cloudera

Versus other NoSQL storage

- **Apache Phoenix: OLTP SQL engine built on HBase**
- 10 node cluster (9 worker, 1 master)
- TPC-H LINEITEM table only (6B rows)



Joining the growing community

Apache Kudu Community

cloudera

mi

intel

ZOOMDATA

AGILDATA

IMPETUS

Personal

ENERNOC

StreamSets

JD.COM 京东

多·快·好·省

LOTAME

dremio

Getting started as a user

- On the web: kudu.apache.org
- User mailing list: user@kudu.apache.org
- Public Slack chat channel (see web site for the link)

- Quickstart VM
  - Easiest way to get started
  - Impala and Kudu in an easy-to-install VM
- CSD and Parcels
  - For installation on a Cloudera Manager-managed cluster

Getting started as a developer

- Source code: github.com/apache/kudu - all commits go here first
- Code reviews: gerrit.cloudera.org - all code reviews are public
- Developer mailing list: dev@kudu.apache.org
- Public JIRA: issues.apache.org/jira/browse/KUDU - includes bug history since 2013

Contributions are welcome and strongly encouraged!

cloudera

APACHE KUDU™

kudu.apache.org

@mike_percy | @ApacheKudu