

facebook

Facebook Messages & HBase

Nicolas Spiegelberg
Software Engineer, Facebook

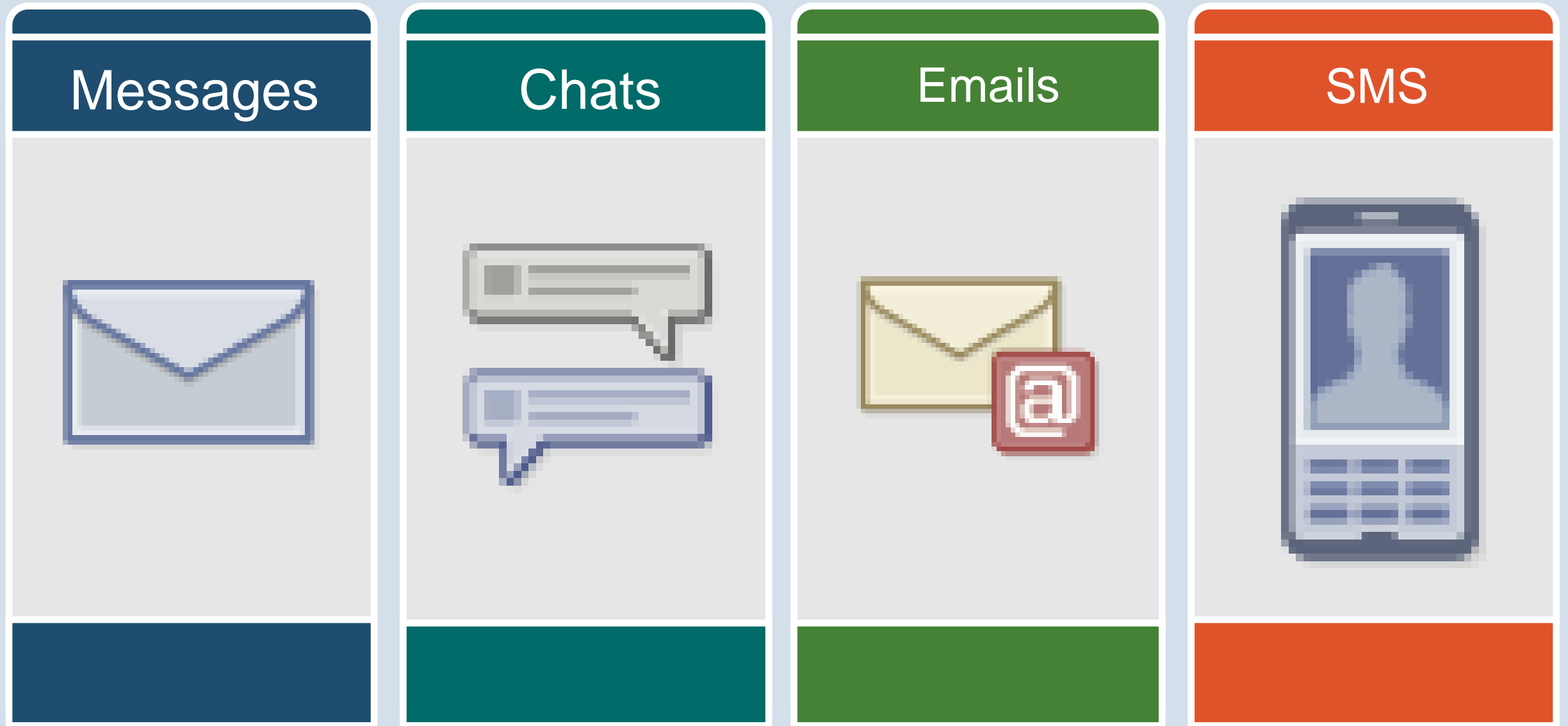
April 8, 2011

Talk Outline

- About Facebook Messages
- Intro to HBase
- Why HBase
- HBase Contributions
- MySQL -> HBase Migration
- Future Plans
- Q&A

Facebook Messages

The New Facebook Messages





Andrew Bosworth
Edit My Profile

News Feed

Messages 1

Other

Events

Friends

Groups Team

Groups Engineering 1

Engineering Spea...

More +

Photos

Quikvote

More +

Friends on Chat



Messages

+ New Message



Search Messages



James Wang, Dave Troiano

remember that time i made prime rib roast?

33 minutes ago · o x



Will Bailey

k i'll follow up with him

2 hours ago · o x



Ben Chiaramonte

like the song

2 hours ago · o x



Kenny Lau

anything should push you

2 hours ago · o x



Ross Bayer

whatsup?

3 hours ago · o x



Mark Zuckerberg

= and i'll dig into the engineering side more

6 hours ago · o x



Eric Antonow

pretty bad -- you win this round

17 hours ago · o x



Pat Kinsel

... and i thought you called yourself a Visio fan for life.

22 hours ago · o x

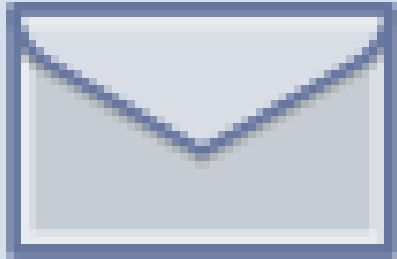


Katie Zacarian

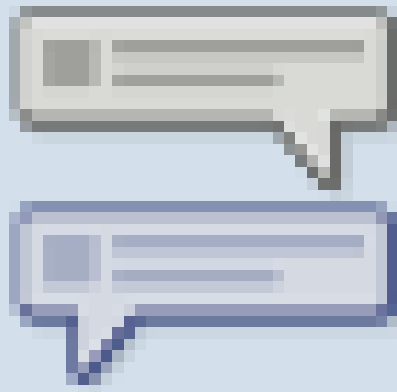
"people respect him"

Yesterday · o x

Monthly data volume prior to launch



$$15\text{B} \times 1,024 \text{ bytes} = 14\text{TB}$$




$$120\text{B} \times 100 \text{ bytes} = 11\text{TB}$$

Messaging Data

- Small/medium sized data  HBase

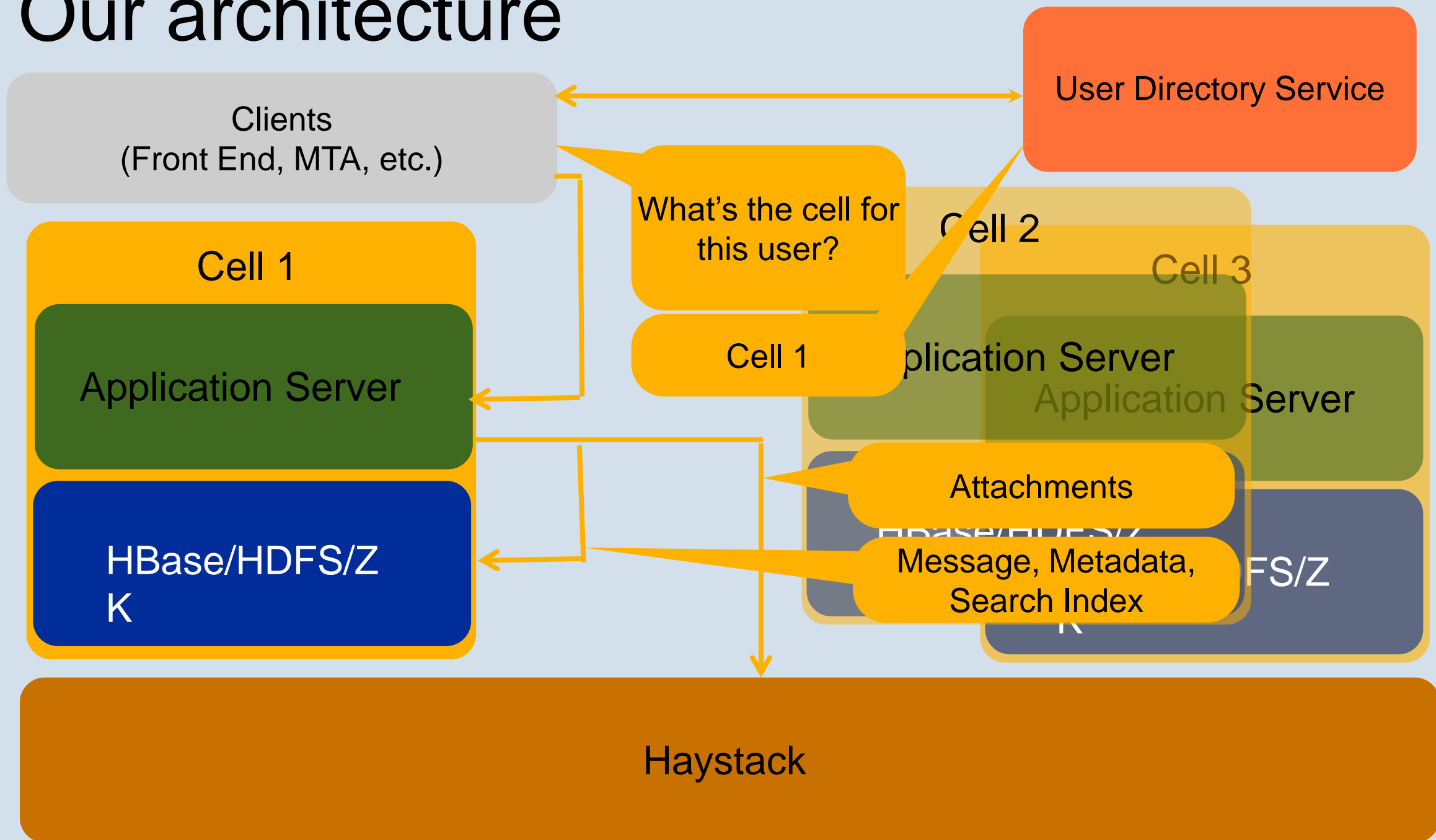
- Message metadata & indices
- Search index
- Small message bodies

- Attachments and large messages  Haystack
- Used for our existing photo/video store

Open Source Stack

- Memcached --> App Server Cache
- ZooKeeper --> Small Data Coordination Service
- HBase --> Database Storage Engine
- HDFS --> Distributed FileSystem
- Hadoop --> Asynchronous Map-Reduce Jobs

Our architecture



About HBase

HBase in a nutshell

- distributed, large-scale data store
- efficient at random reads/writes
- initially modeled after Google's BigTable
- open source project (Apache)

When to use HBase?

- storing large amounts of data
- need high write throughput
- need efficient random access within large data sets
- need to scale gracefully with data
- for structured and semi-structured data
- don't need full RDMS capabilities (cross table transactions, joins, etc.)

HBase Data Model

- An HBase table is:
 - a sparse , three-dimensional array of cells, indexed by:
RowKey, ColumnKey, Timestamp/Version
 - sharded into *regions* along an ordered RowKey space
- Within each region:
 - Data is grouped into column families
 - Sort order within each column family:

Row Key (asc), Column Key (asc), Timestamp (desc)

Example: Inbox Search

- Schema
 - **Key:** RowKey: **userid**, Column: **word**, Version: **MessageID**
 - **Value:** Auxillary info (like offset of word in message)
- Data is stored sorted by <userid, word, messageID>:

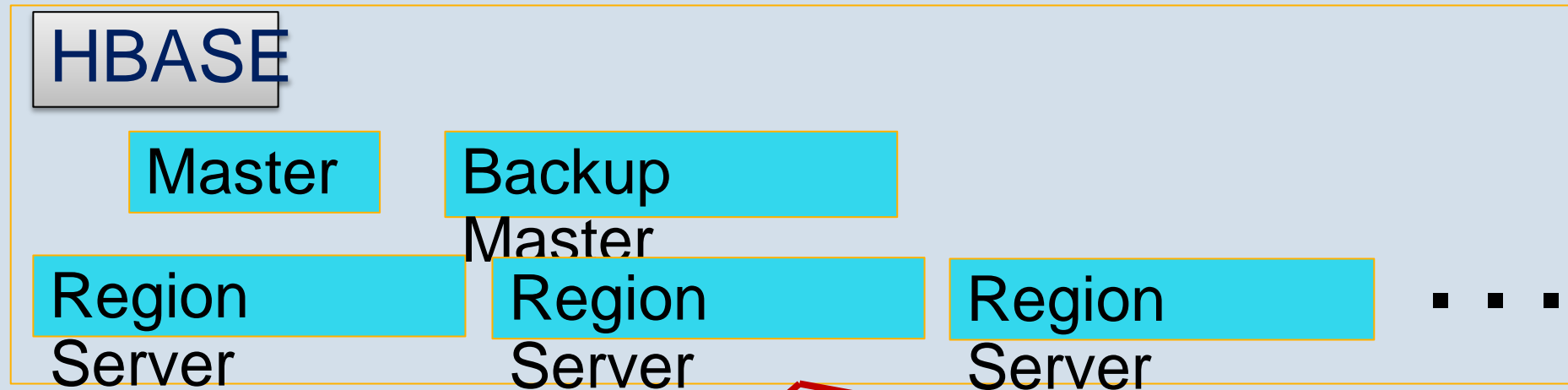
User1:hi:17->offset1
User1:hi:16->offset2
User1:hello:16->offset3
User1:hello:2->offset4
...
User2:.....
User2:....
...

Can efficiently handle queries like:

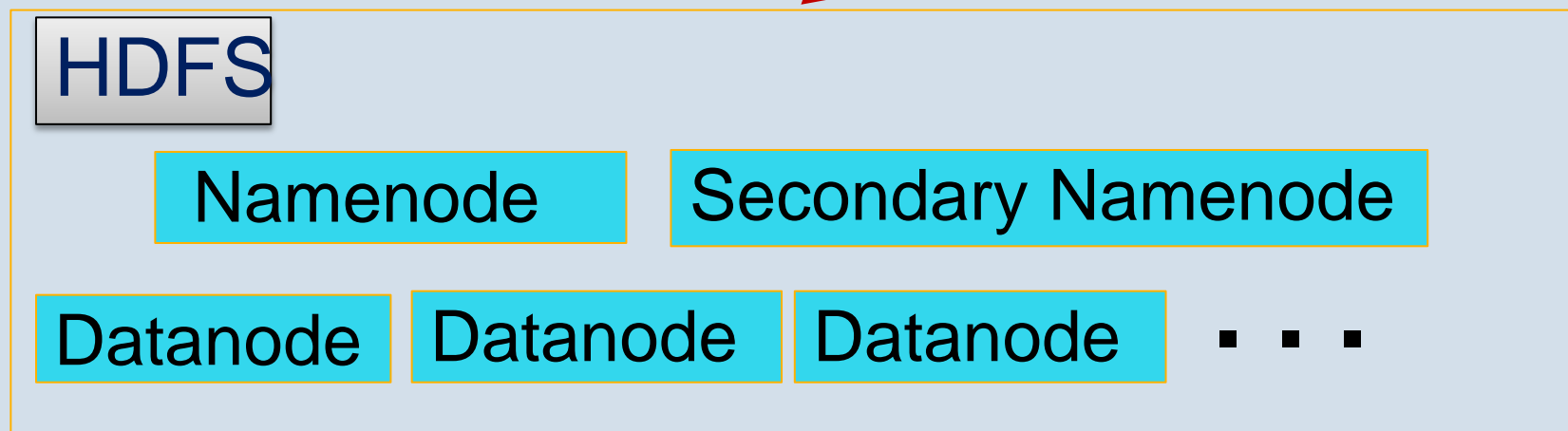
- Get top N messageIDs for a specific user & word
- Typeahead query: for a given user, get words that match a prefix

HBase System Overview

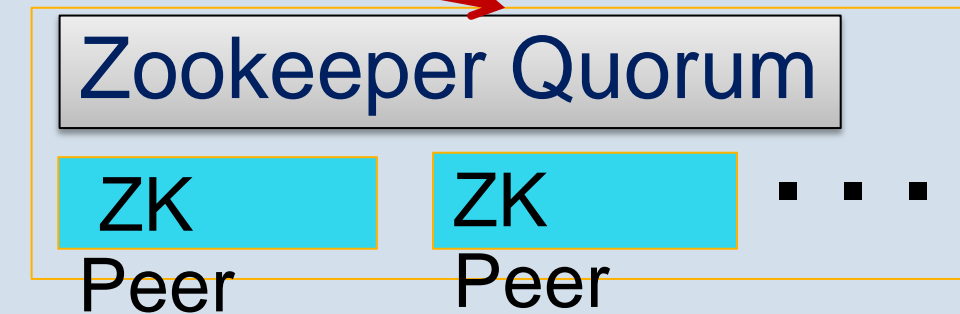
Database Layer



Storage Layer

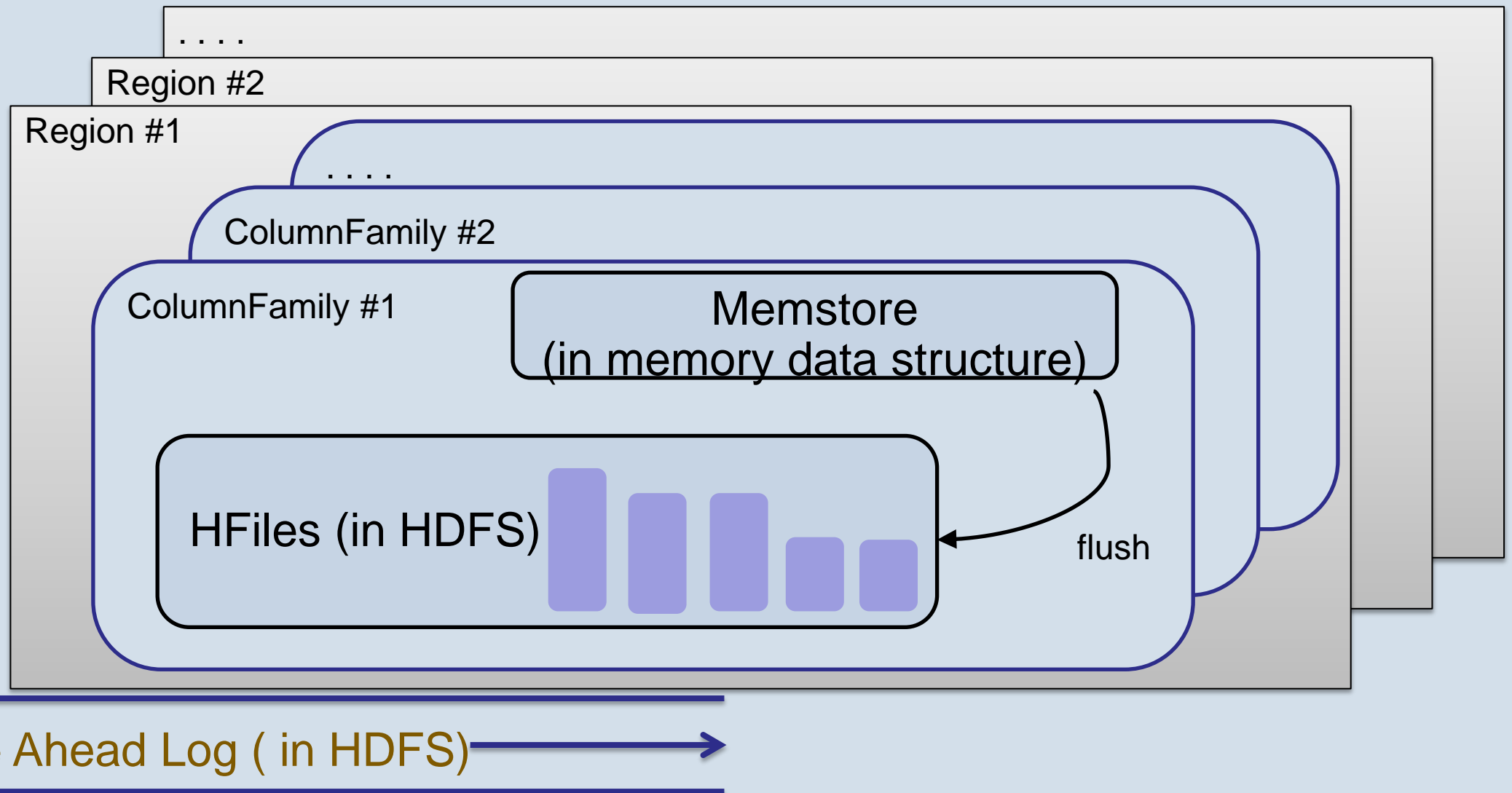


Coordination Service



HBase Overview

HBASE Region Server



HBase Overview

- Very good at random reads/writes
- Write path
 - Sequential write/sync to commit log
 - update memstore
- Read path
 - Lookup memstore & persistent HFiles
 - HFile data is sorted and has a block index for efficient retrieval
- Background chores
 - Flushes (memstore -> HFile)
 - Compactions (group of HFiles merged into one)

Why HBase?

Performance is great, but what else...

Horizontal scalability

- HBase & HDFS are elastic by design
- Multiple table shards (regions) per physical server
- On node additions
 - Load balancer automatically reassigns shards from overloaded nodes to new nodes
 - Because filesystem underneath is itself distributed, data for reassigned regions is instantly servable from the new nodes.
- Regions can be *dynamically* split into smaller regions.
 - Pre-sharding is not necessary
 - Splits are near instantaneous!

Automatic Failover

- Node failures automatically detected by HBase Master
- Regions on failed node are distributed evenly among surviving nodes.
 - Multiple regions/server model avoids need for *substantial* overprovisioning
- HBase Master failover
 - 1 active, rest standby
 - When active master fails, a standby automatically takes over

HBase uses HDFS

We get the benefits of HDFS as a storage system for free

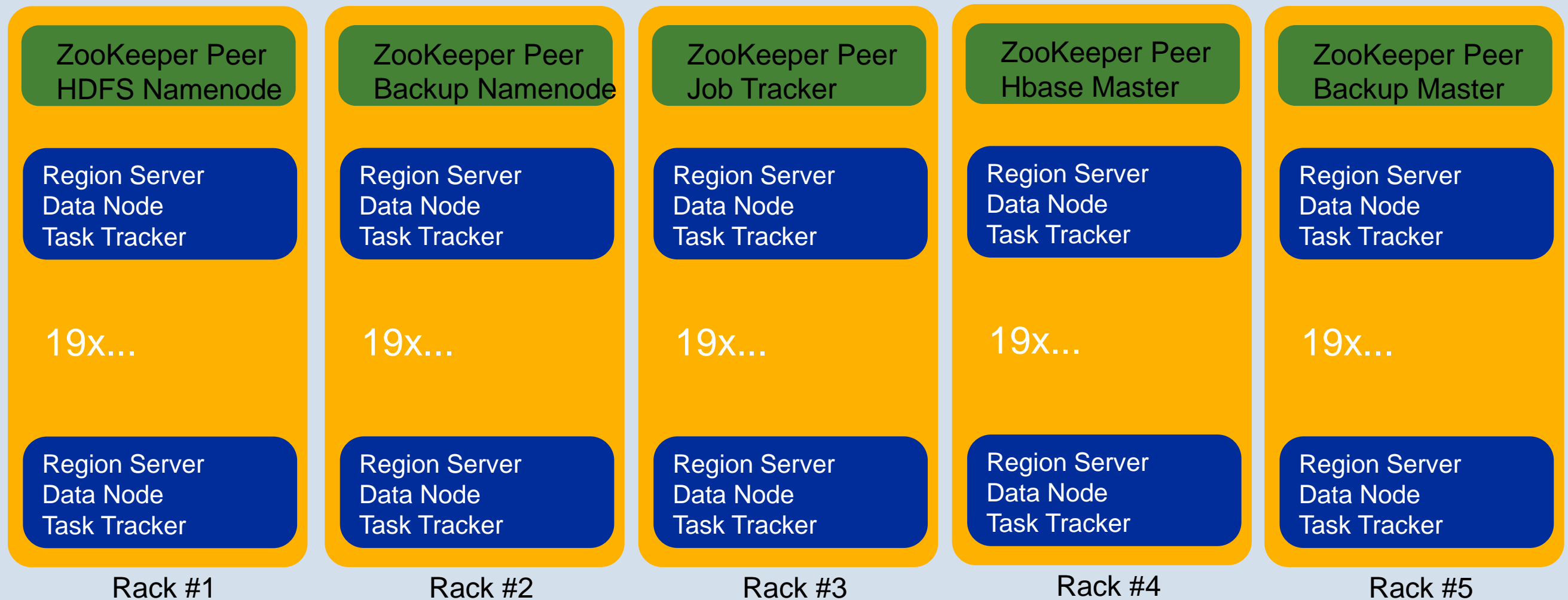
- Fault tolerance (block level replication for redundancy)
- Scalability
- End-to-end checksums to detect and recover from corruptions
- Map Reduce for large scale data processing
- HDFS already battle tested inside Facebook
 - running petabyte scale clusters
 - lot of in-house development and operational experience

Simpler Consistency Model

- HBase's strong consistency model
 - simpler for a wide variety of applications to deal with
 - client gets same answer no matter which replica data is read from
- Eventual consistency: tricky for applications fronted by a cache
 - replicas may heal *eventually* during failures
 - but stale data could remain stuck in cache

Typical Cluster Layout

- Multiple clusters/cells for messaging
 - 20 servers/rack; 5 or more racks per cluster
- Controllers (master/Zookeeper) spread across racks



HBase Enhancements

Goal: Zero Data Loss

Goal of Zero Data Loss/Correctness

- *sync* support added to hadoop-20 branch
 - for keeping transaction log (WAL) in HDFS
 - to guarantee durability of transactions
- Row-level ACID compliance
- Enhanced HDFS's Block Placement Policy:
 - Original: rack aware, but minimally constrained
 - Now: Placement of replicas constrained to configurable node groups
 - Result: Data loss probability reduced by orders of magnitude

Availability/Stability improvements

- HBase master rewrite- region assignments using ZK
- Rolling Restarts – doing software upgrades without a downtime
- Interrupt Compactions – prioritize availability over minor perf gains
- Timeouts on client-server RPCs
- Staggered major compaction to avoid compaction storms

Performance Improvements

- Compactions
 - critical for read performance
 - Improved compaction algo
 - delete/TTL/overwrite processing in minor compactions
- Read optimizations:
 - Seek optimizations for rows with large number of cells
 - Bloom filters to minimize HFile lookups
 - Timerange hints on HFiles (great for temporal data)
 - Improved handling of compressed HFiles

Operational Experiences

- Darklaunch:
 - shadow traffic on test clusters for continuous, at scale testing
 - experiment/tweak knobs
 - simulate failures, test rolling upgrades
- Constant (pre-sharding) region count & controlled rolling splits
- Administrative tools and monitoring
 - Alerts (HBCK, memory alerts, perf alerts, health alerts)
 - auto detecting/decommissioning misbehaving machines
 - Dashboards
- Application level backup/recovery pipeline

Working within the Apache community

- Growing with the community
 - Started with a stable, healthy project
 - In house expertise in both HDFS and HBase
 - Increasing community involvement
- Undertook massive feature improvements with community help
 - HDFS 0.20-append branch
 - HBase Master rewrite
- Continually interacting with the community to identify and fix issues
 - e.g., large responses (2GB RPC)

Data migration

Another place we used HBase heavily...

Move messaging data from MySQL to HBase



Move messaging data from MySQL to HBase

- In MySQL, inbox data was kept normalized
 - user's messages are stored across many different machines
- Migrating a user is basically one big join across tables spread over many different machines
- Multiple terabytes of data (for over 500M users)
- Cannot pound 1000s of production UDBs to migrate users

How we migrated

- Periodically, get a full export of all the users' inbox data in MySQL
- And, use bulk loader to import the above into a *migration* HBase cluster
- To migrate users:
 - Since users may continue to receive messages during migration:
 - double-write (to old and new system) during the migration period
 - Get a list of all recent messages (since last MySQL export) for the user
 - Load new messages into the *migration* HBase cluster
 - Perform the join operations to generate the new data
 - Export it and upload into the final cluster

Future Plans

HBase Expands

Facebook Insights Goes Real-Time

- Recently launched real-time analytics for social plugins on top of HBase
- Publishers get real-time distribution/engagement metrics:
 - # of impressions, likes
 - analytics by
 - Domain, URL, demographics
 - Over various time periods (the last hour, day, all-time)
- Makes use of HBase capabilities like:
 - Efficient counters (read-modify-write increment operations)
 - TTL for purging old data

Future Work

It is still early days....!

- Namenode HA (AvatarNode)
- Fast hot-backups (Export/Import)
- Online schema & config changes
- Running HBase as a service (multi-tenancy)
- Features (like secondary indices, batching hybrid mutations)
- Cross-DC replication
- Lot more performance/availability improvements

Thanks! Questions?

facebook.com/engineering