WOTD

51CTO

全球 软件开发技术峰会

World Of Tech 2017

2017年12月1日-2日 • 深圳中洲万豪酒店

DEVELOPMENT

New HBase Brings New Era

Ted Yu

Senior Staff Engineer

About Me

Ted Yu

- HBase PMC since 2011
- Senior Staff @Hortonworks

Outline

Recent releases
Versioning / Compatibility
Changes in behavior
Major Features in HBase-2.0
Phoenix- Latest features

DISCLAIMER!!!

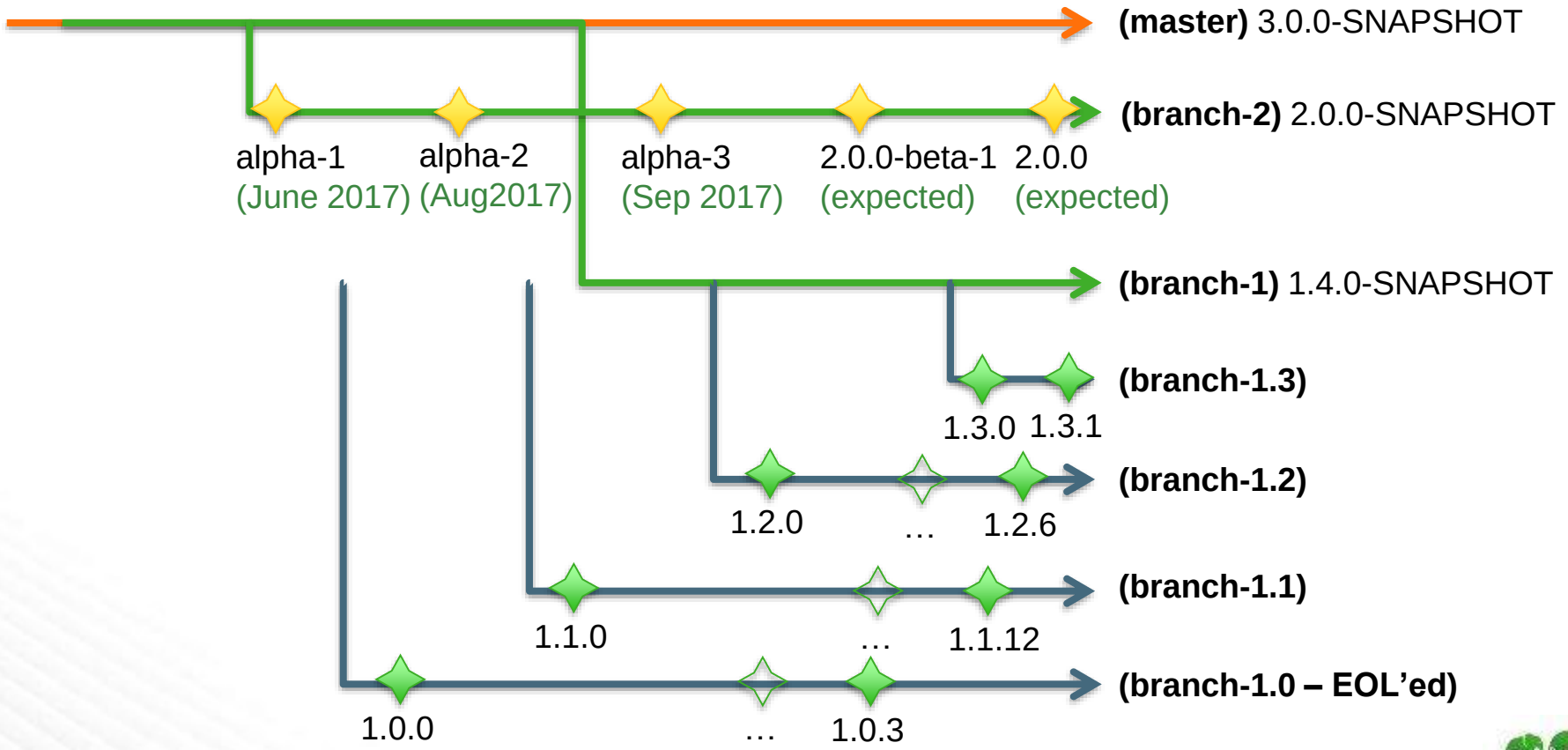
HBase-2.0.0 is NOT released yet (as of October 2017)

Community is working hard on this release

Expected to arrive before end of year 2017

3 alpha versions have been released, beta releases to come

2017 H2 (repo and releases)



How to choose a release (H2 2017)

- If you are on 0.98.x or earlier, time to upgrade!
- 1.0 is EOL'ed. Use 1.1.x at least
- Both 1.1 and 1.2 are pretty stable
- Starting from scratch, use 1.2 or 1.3
- Moving between minor versions is easy for 1.x
- Start testing out 2.0 with alpha and beta releases

Outline

Recent releases

Versioning / Compatibility

Changes in behavior

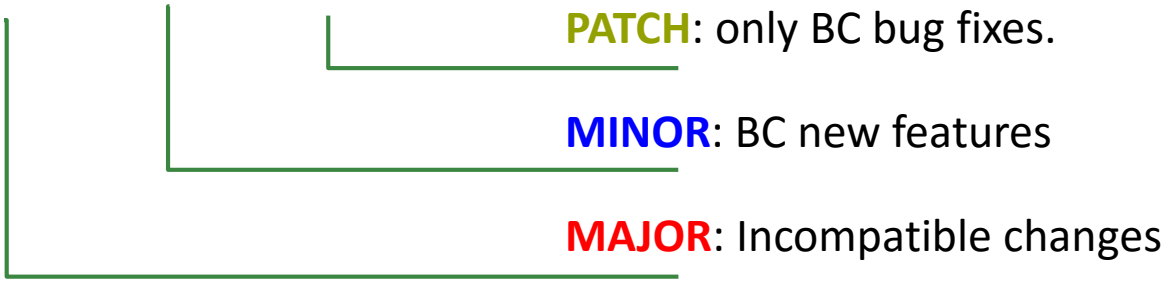
Major Features in HBase-2.0

Phoenix- Latest Features

Semantic Versioning

Starting with the 1.0 release, HBase works toward Semantic Versioning

MAJOR.MINOR.PATCH[-identifiers]



Compatibility

- Compatibility is NOT a simple yes or no
- Many dimensions
  - source, binary, wire, command line, dependencies etc
- Read https://hbase.apache.org/book.html#upgrading
- What is client interface?

HBase API surface

Client API

Explicitly marked with **`InterfaceAudience.Public`**

Get/Put/Table/Connection, etc

LimitedPrivate API

Explicitly marked with **`InterfaceAudience.LimitedPrivate`**

Coprocessors, replication APIs

Private API

Explicitly marked with **`InterfaceAudience.Private`**

All other classes not marked

Also `InterfaceStability.{Stable,Evolving,Unstable}`

	Major	Minor	Patch
Client-Server Wire Compatibility	✗	✓	✓
Server-Server Compatibility	✗	✓	✓
File Format Compatibility	✗*	✓	✓
Client API Compatibility	✗	✓	✓
Client Binary Compatibility	✗	✗	✓
Server Side Limited API Compatibility	✗	✗*/✓*	✓
Dependency Compatibility	✗	✓	✓
Operation Compatibility	✗	✗	✓

Outline

Recent releases
Versioning / Compatibility
Changes in behavior
Major Features in HBase-2.0
Phoenix-Latest Features

Changes in behavior – HBase-2.0

- JDK-8+ only
- Likely Hadoop-2.7+ and Hadoop-3 support
- Filter and Coprocessor changes
- Managed connections are gone.
- Target is “Rolling upgradable” from 1.x
- Updated dependencies
- Deprecated client interfaces are gone
- More info at [1]

- [1] https://docs.google.com/document/d/1WCsVlnHjJeKUcl7wHwqb4z9iEu_ktczrlKHK8N4SZzs/edit#heading=h.v21r9nz8g01j

Client API Overview

HBase 0.98 Name(s)	HBase 1.x + 2.x name(s)
HConnectionManager, ConnectionManager	ConnectionFactory
HConnection, ClusterConnection	Connection
HBaseAdmin	Admin
HTable	Table RegionLocator BufferedMutator

How to prepare for HBase-2.0

- 2.0 contains more API clean up
- Cleanup ProtoBuf and guava “leaks” into the API
- Some deprecated APIs (HConnection, HTable, HBaseAdmin, etc) going away
- Start using JDK-8 (and G1GC). You will like it.
- 1.x client should be able to do read / write / scan against 2.0 clusters
- Some DDL / Admin operations may not work

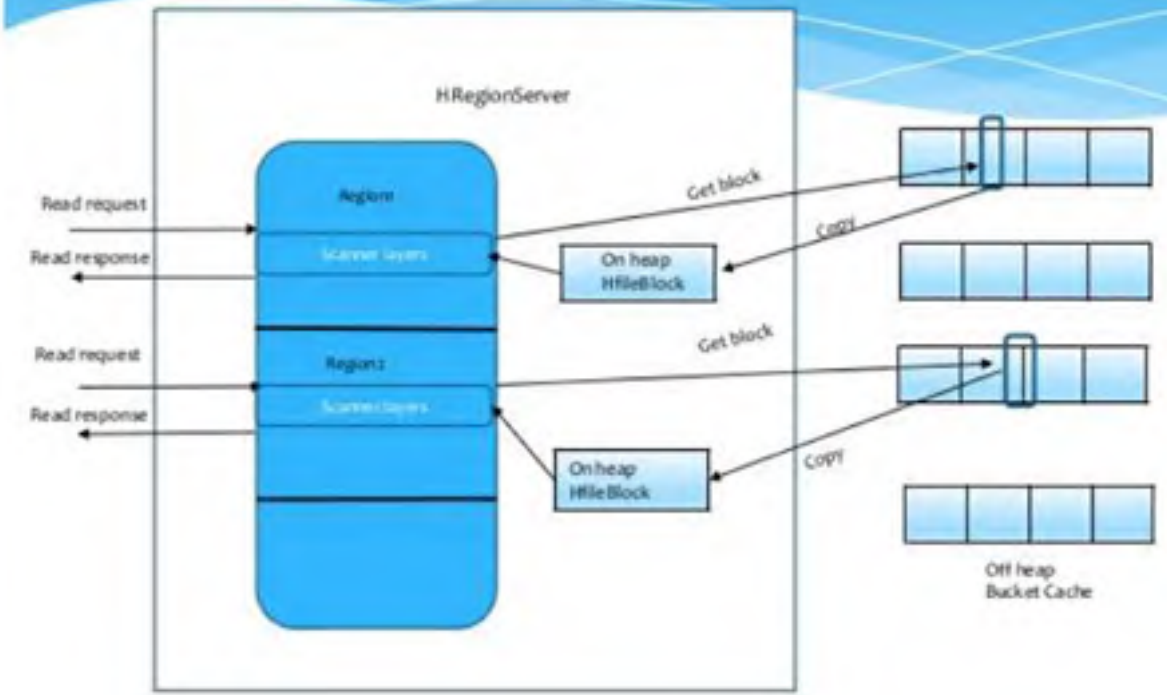
Outline

Recent releases
Versioning / Compatibility
Changes in behavior
Major Features in HBase-2.0
Phoenix- Latest Features

Offheaping Read Path

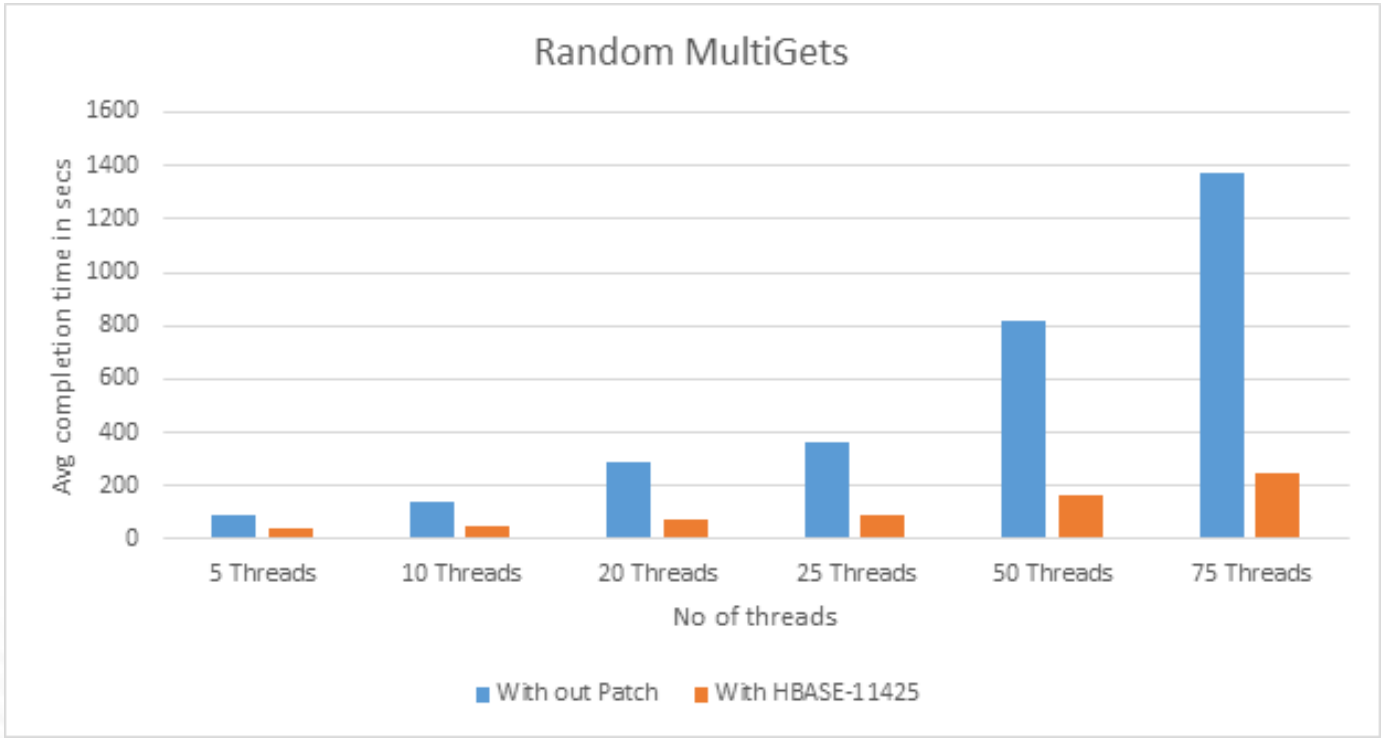
- Goals
  - Make use of all available RAM
  - Reduce temporary garbage for reading
  - Make bucket cache as fast as LRU cache
- HBase block cache can be configured as L1 / L2
- “Bucket cache” can be backed by onheap, off-heap or SSD
- Different sized buckets in 4MB slices
- HFile blocks placed in different buckets
- Cell comparison only used to deal with byte[]
- Copy the whole block to heap

Read from Bucket Cache



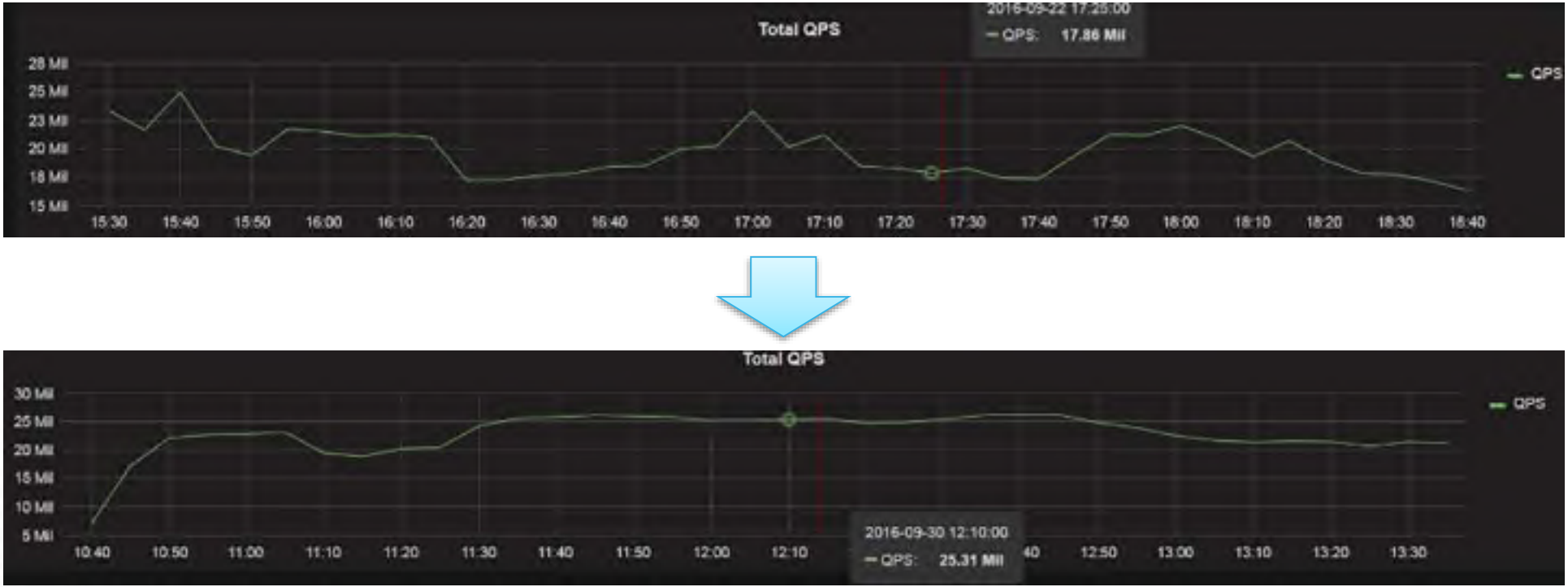
https://www.slideshare.net/HBaseCon/offheaping-the-apache-hbase-read-path

Offheaping Read Path



https://blogs.apache.org/hbase/entry/offheaping_the_read_path_in1

Offheaping Read Path



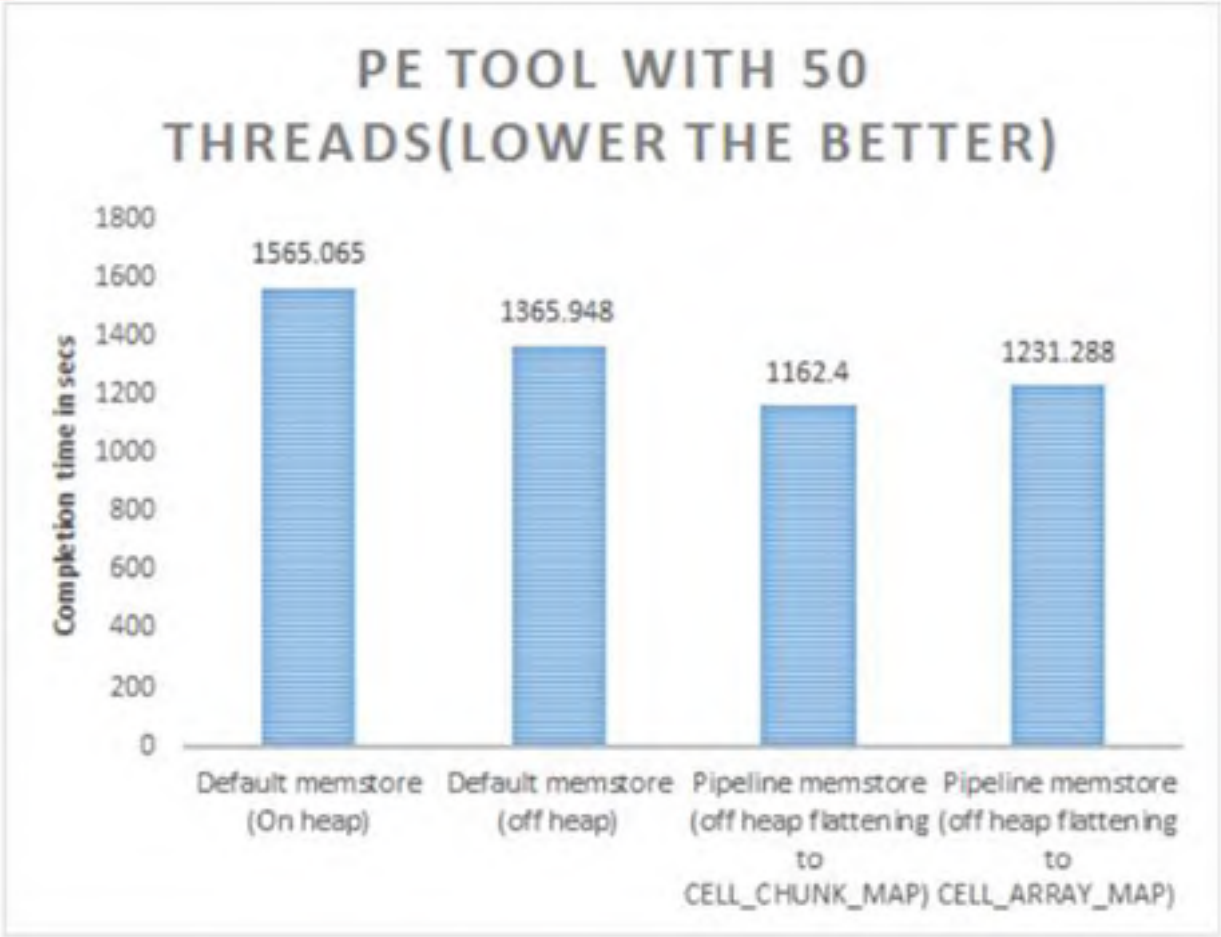
Offheap Read-Path in Production - The Alibaba story (https://blogs.apache.org/hbase/)

Offheaping Write Path

- Reduce GC, improve throughput predictability
- Use off-heap buffer pools to read RPC requests
- Remove garbage created in Codec parsing, MSLAB copying, etc
- MSLAB is used for preventing heap fragmentation
- Now MSLAB can allocate offheap pooled buffers [2MB]

Offheaping Write Path

- Allows bigger total memstore space
- Cell data goes off-heap
- Cell metadata (and CSLM#Entry, etc) is on-heap (~100 byte overhead)
- Async WAL is used to avoid copying Cells onto heap
- Works with new “Compacting Memstore”

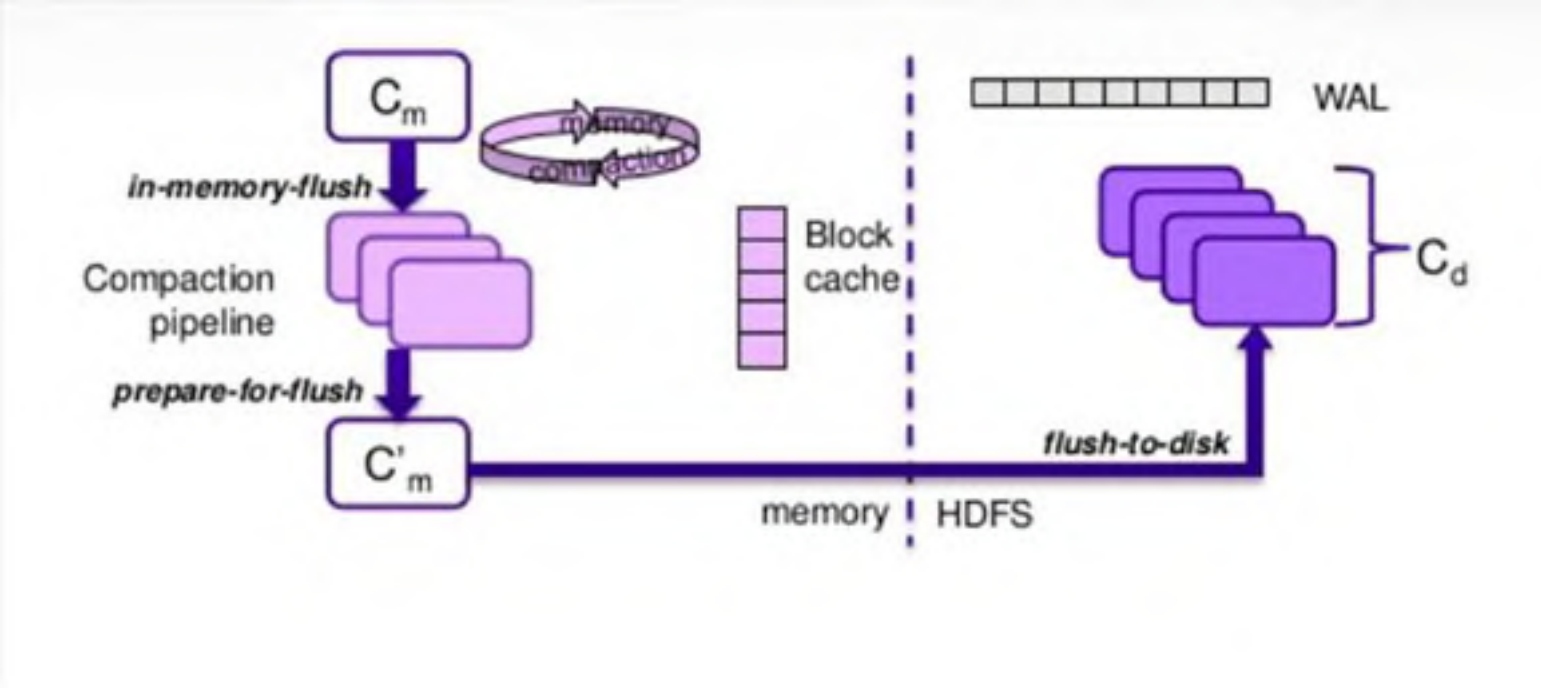


https://docs.google.com/document/d/1fj5P8JeutQ-Uadb29ChDscMuMaJqaMNRI86C4k5S1rQ/edit#

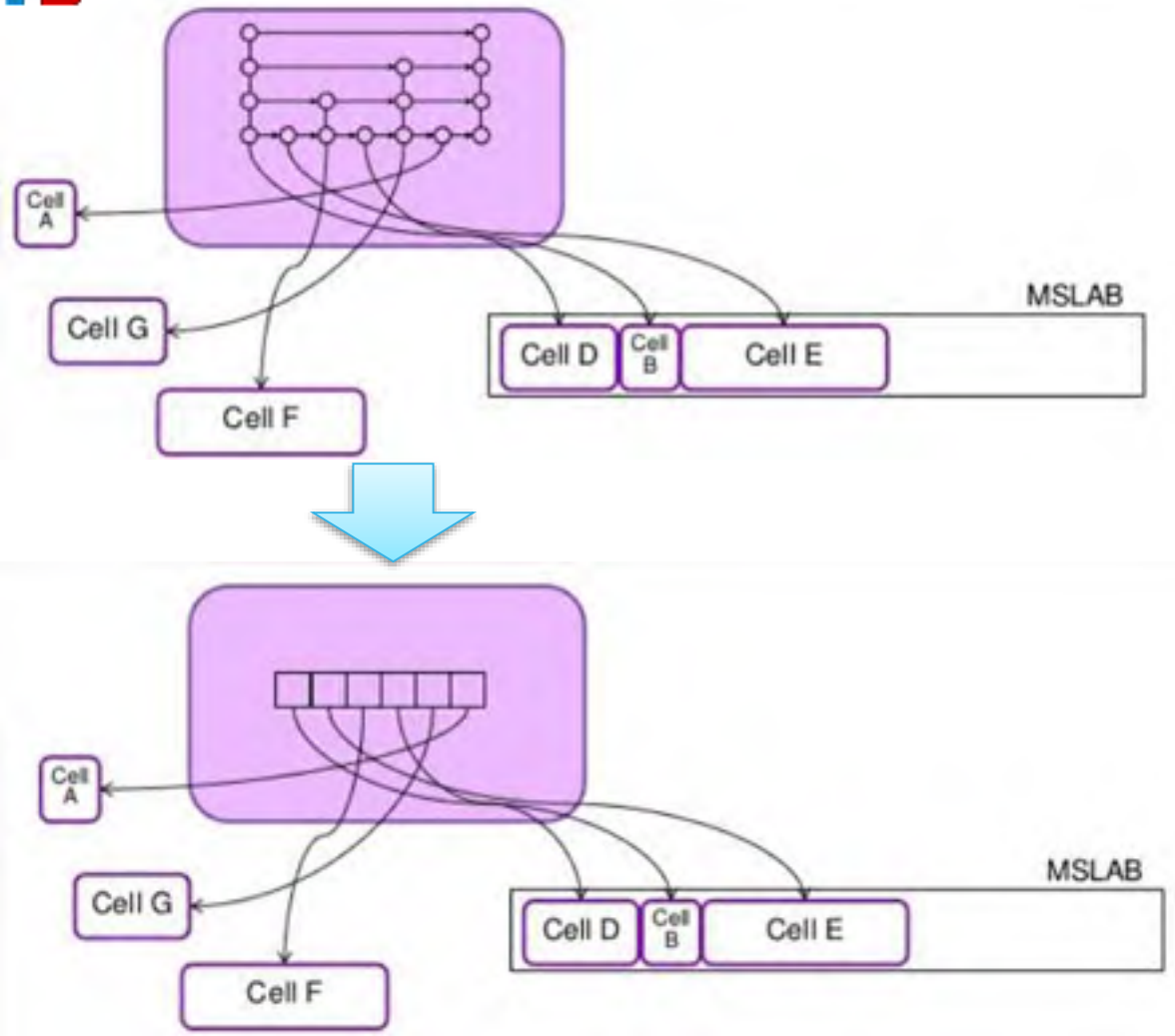
Compacting Memstore

- In memory flushes
  - Reduce index overhead per cell
  - Gains are proportional to cell size
- In memory compaction
  - Eliminate duplicates
  - Gains are proportional to duplicates
- Reduce flushes to disk
  - Reduce file creation
  - Reduce write amplification

Compacting Memstore



https://www.slideshare.net/HBaseCon/apache-hbase-accelerated-inmemory-flush-and-compaction



https://www.slideshare.net/HBaseCon/apache-hbase-accelerated-inmemory-flush-and-compaction

Region Assignment

- Region assignment is one of the pain points
- Complete overhaul of region assignment
- Based on the internal “Procedure V2” framework
- Series of operations are broken down into states in State Machine
- State’s are serialized to durable storage
- Uses a WAL for Procedure V2 on HDFS
- Backup master can recover the state from where left off

Async Client

- A complete new client maintained in HBase
- Different than OpenTSDB/asynchbase
- Fully async end to end
- Based on Netty and JDK-8 CompletableFutures
- Includes most critical functionality including Admin / DDL
- Some advanced functionality (read replicas, etc) are not done yet
- Have both sync and async client.

Async Client

```
AsyncConnection connection = ConnectionFactory.createAsyncConnection(conf).get();
AsyncTable table = connection.getTable(TABLE_NAME, ForkJoinPool.commonPool());
CountDownLatch putLatch = new CountDownLatch(count);
IntStream.range(0, count).forEach(
  i -> table.put(new Put(concat(row, i)).addColumn(FAMILY, QUALIFIER, concat(VALUE, i)))
      .thenAccept(x -> putLatch.countDown()));
putLatch.await();
BlockingQueue<Boolean> existsResp = new ArrayBlockingQueue<>(count);
IntStream.range(0, count)
    .forEach(i -> table.exists(new Get(concat(row, i)).addColumn(FAMILY, QUALIFIER))
        .thenAccept(x -> existsResp.add(x)));
```

C++ Client

- `-std=c++14`
- Synchronous client built fully on async client
- Uses folly and wangle from FB (like Netty, JDK-8 Futures)
- Follow the same architecture as the Java async client
- Gets / Puts / Multi-Gets / Scan are working
- load-client practices above operations
- Supports connection to secure cluster by using Cyrus lib

C++ Client

```
int main(int argc, char *argv[]) {
  HBaseConfigurationLoader loader;
  hbase::optional<Configuration> conf = loader.LoadDefaultResources();

  auto tn = std::make_shared<TableName>(folly::to<TableName>(FLAGS_table));
  auto num_puts = FLAGS_num_rows;

  // Create Client and Table
  Client client = std::make_unique<Client>(*conf);
  Table table = client->Table(*tn);

  // Do the Put requests
  for (uint64_t i = 0; i < num_puts; i++) {
    table->Put(*MakePut(Row(row, i)));
  }

  // Do the Get requests
  for (uint64_t i = 0; i < num_puts; i++) {
    auto result = table->Get(Get{Row(row, i)});
  }
}
```

Backup / Restore

- Need a reliable native Backup mechanism

Provides	Snapshots	Replication	Backup
Protection Against Hardware Failure	No(1)	Yes	Yes
Protection Against User / Application Error	Yes	No	Yes
Availability During Full Site Outage	No	Yes	No
Retention / Audit	No	No	Yes

1) Snapshots combined with exports protect against hardware failure, but are usually impractical because they require full data copies each time.

Backup / Restore

- Full backups and Incremental backups
- Backup destination to a remote FS
- HDFS, S3, ADLS, WASB, and any Hadoop-compatible FS
- Full backup based on “snapshots”
- Incremental backup ships Write-Ahead-Logs
- Backup + Replication are complimentary (for a full DR solution)

Backup / Restore

- Flexible restore options

Approach	Overview	Pros / Cons	Use When
Overwrite		**Pros:** Simple. CLI-driven. Easy to understand final state. **Cons:** Requires database / app downtime. 0 RPO not possible.	Use when you want the simplest possible backup process and you can accept some data loss.
Restore From Staging		**Pros:** Easy, CLI driven approach to re-stage data. 0 RPO Possible. **Cons:** User must identify and extract important data.	Use when data loss is not acceptable and you can build procedures to extract and re-apply data.
Retrieve		**Pros:** No downtime. 0 RPO possible. **Cons:** Scripting tools or software must be used to extract old values.	Use when data loss is not acceptable and re-staging old data is not practical, for example due to size constraints.

FileSystem Quotas

- Request count or size per sec,min,hour,day (per server)
- Number of tables / number of regions in namespace
- New work enables quotas for disk space usage
- Master periodically gathers info about disk space usage
- Regionservers enforce limit when notified from master
- Limits per table or per namespace (not per user)
- Violation policy: {DISABLE, NO_WRITES_COMPACTIONS, NO_WRITES, NO_INSERTS}

FileSystem Quotas

- Much more complex when snapshots are in the picture (similar to quotas with hard links in FS)
- Snapshot usage is counted against the originating table
- A File is only counted once (even original table, or snapshot or restored table refers to it)

```
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY =>DELETES_WITH_COMPACTIONS,
NAMESPACE => 'prod_website', LIMIT => '7125GB'
```

Will be released for the first time

- These will be released for the first time in Apache (although other backports exist)
- Medium Object Support
- Region Server groups
- Favored Nodes enhancements
- WAL's and data files in separate FileSystems

Misc

- API Cleanup
- Replication changes
- New metrics API
- Metrics API for Coprocessors
- Tons of other improvements

Outline

Recent releases
Versioning / Compatibility
Changes in behavior
Major Features in HBase-2.0
Phoenix- Latest Features

Apache Phoenix-Latest features

Below features are targeted for 4.12.0

- Global Index optimization and improved index rebuilding
- HLL for approximate distinct count.
- Index Scrutiny Tool
- Support of running queries with table sampling
- Load balancer for PQS
- ACLs for Phoenix DDLs

Apache Phoenix-Latest releases

- Reduce on disk footprint through column encoding and optimized storage format for write-once data
- Support for HBase 1.3.1
- Local index hardening and performance improvements
- Use of snapshots for MR-based queries and async index building.
- Support of cursors(Forward moving)
- Support Apache Spark 2.0 in Phoenix/Spark integration
- Improved Hive integration

Thank you!