HBase

Introduction and New Developments

Andrew Purtell
andrew_purtell@trendmicro.com
apurtell@apache.org

Outline

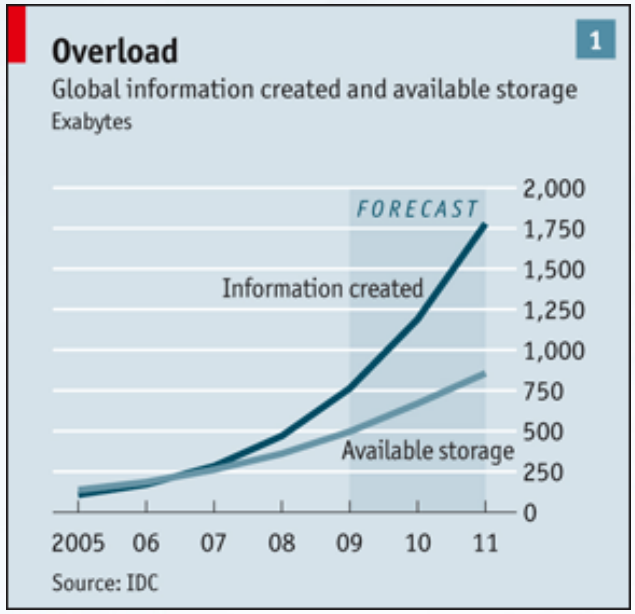
- Big Data and Cloud Computing
- HBase Introduction
- New Features
  - ACID Guarantees
  - Multi Data Center Replication
  - Security
  - Coprocessors
- Wrap Up

Big Data and Cloud Computing

Big Data

Big Data defined

- Scale beyond the limits of conventional data storage solutions today either in terms of capacity or of the sustainable cost of the solution
  - Trillions (10^12) of data items
  - Petabytes (10^15 bytes) of data volume
- Google encountered Big Data in their operations and devised architectural solutions for it, including BigTable
- BigTable is the inspiration for HBase



The Big Data Age

- According to one estimate, globally the world created 150 exabytes of data in 2005
- This year, the world may create more than 1,500 exabytes of data

Medium Data

- "*What about Medium Data? We like to say that Facebook doesn't run Hadoop because it has a lot of data, but that Facebook has a lot of data because it runs Hadoop. Businesses that use Hadoop find that keeping data is worthwhile because Hadoop helps them process it in new ways.*"

  Mike Olson, CEO, Cloudera

  http://www.cloudera.com/
- Many business, even small ones, during the course of their normal operations can generate petabytes of data per year
- If they retain it, they can mine it and gain insights
- Open source analytics enablers like the Hadoop software ecosystem – of which HBase is a part – make this an emerging reality

Cloud Computing

- "Internet-based access to highly scalable pay-per-use IT capabilities"

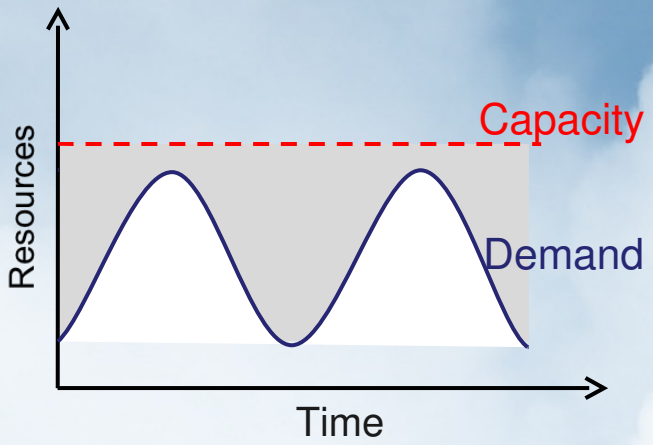
  - Ynema Mangum, SUN Microsystems

  - Describes infrastructure-as-a-service (IaaS) pretty welll

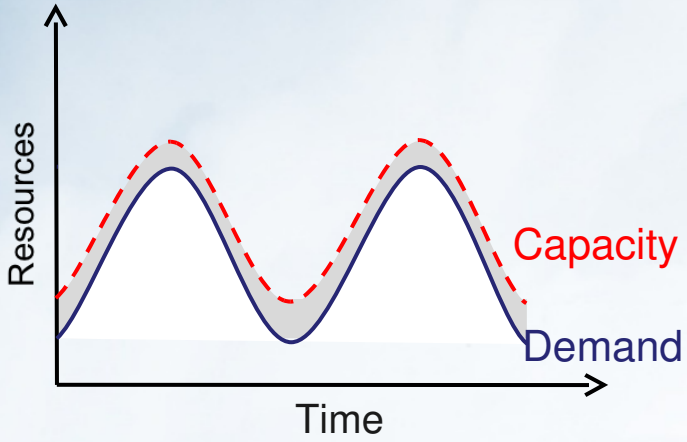
- An evolution of network computing
  - Workstation → Network → Grid → Cloud
  - Cloud computing is client-server computing that abstracs the details of the server away
  - Scale free
  - Resources anywhere/everywhere
  - Loosely coupled computing
  - Decentralized, open standards
  - Open technologies
  - New ownership model

Cloud Computing

- Scale free computing
- A limitless pool of on demand resources is a game changer
  - Pay by use instead of provisioning for peak
  - Elastic scaling up and down, scale up very large
  - Optimize costs to actual service demand
  - Worry only about the application or service, not about infrastructure



Static data center



Data center in the cloud

Unused resources

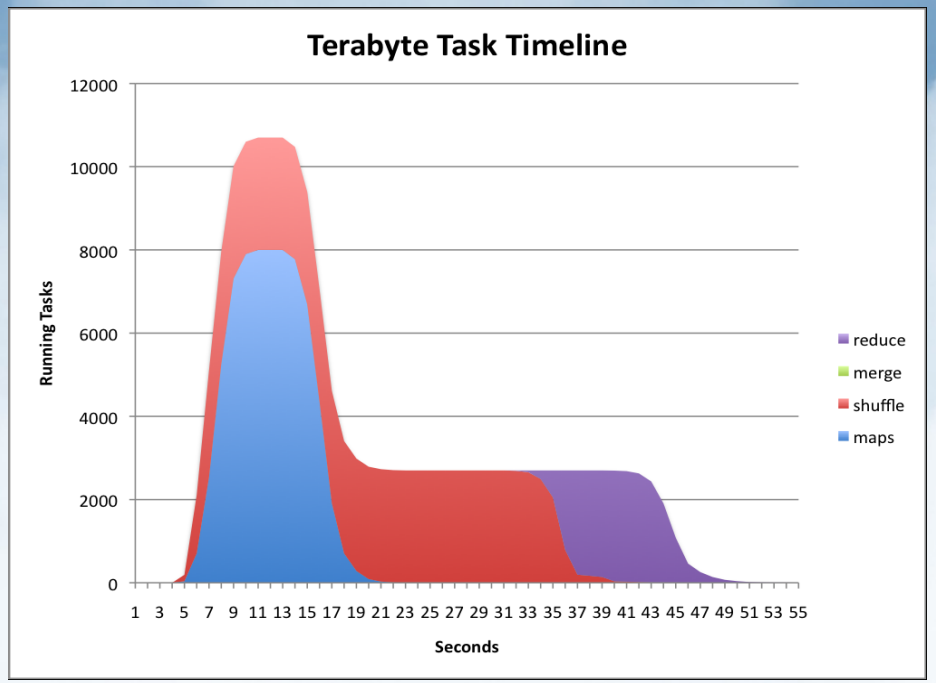
Convergence

- As we enter the age of Big Data we have the scale (and scale-free computational nature) of the Cloud to manage it
- The Cloud is a driver of Big Data even as it is a means to deal with it
- Hadoop and HBase are Cloud scale architectures
  - → Container for Big (and Medium) Data
  - → Scale free computational framework for managing it

HBase (and Hadoop) Introduction

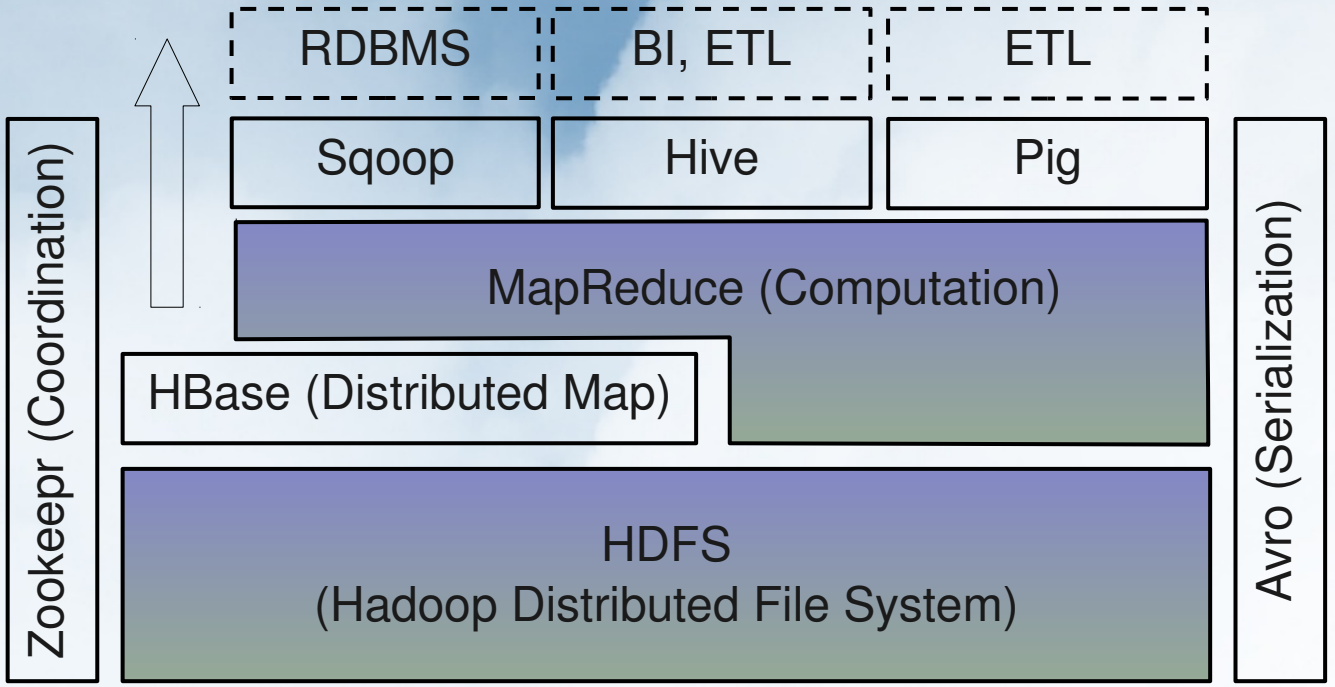
Hadoop – The Platform

- A transparently scalable computing platform
- “Cloud scale” grid data processing
  - 10K nodes, 100 million files, 10 petabytes
- 2009 Gray Sort winner: 0.578 terabytes/minute, a new world record
  - Sort a terabyte (1,000,000,000,000 bytes) in 62 seconds
  - Sort a petabyte (1,000,000,000,000,000 bytes) in 16.25 hours



Hadoop – The Ecosystem

- MapReduce framework (Core)
- Pluggable cluster task scheduler (Core)
- Distributed replicated fault tolerant file system (HDFS)
- Horizontally scalable distributed fault tolerant database (HBase)
- Various value adds: Add on packages (analytics, management), distributions, dashboards, etc.



Seek Versus Sort and Merge

- At scale, disk time dominates storage and computation
  - CPU, RAM, and disk size double every 18-24 months
  - Seek time remains nearly constant (~5% per year)
- Two database paradigms
  - Seek dominant: Indexed (B-Tree) seek and replace (RDBMS)
  - Transfer dominant: sort/merge (MapReduce, Bigtable)
- Seek is inefficient compared to transfer at scale
  - Given:
    - 10 MB/second transfer bandwidth
    - 10 milliseconds disk seek time
    - 100 bytes per entry (10 billion entries)
    - 10 kB per page (1 billion pages)
  - Updating 1% of entries (100,000,000) takes:
    - 1,000 days with random B-Tree updates
    - 100 days with batched B-Tree updates
    - **1 day with sort and merge**

→ Log structured data access on streaming filesystem

HBase – The Hadoop Database

- A persistent distributed hash map

  ... and separate namespaces

  → Tables

  ... and an index

  → Rows

  ... and locality of I/O references

  → Column families

  ... and time ranges

  → Timestamps

HBase – The Hadoop Database

- Google: "*BigTable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers*"
- Goal: Store billions of rows * millions of columns * thousands of versions
- An open source version of BigTable, enhanced with additional features developed by the community
  - Fast fault recovery via Zookeeper
  - Query push down via server side query and scanner filters
  - Optimizations for real time queries
  - Rolling restarts
  - Push metrics to log files or Ganglia (http://ganglia.info/)
  - Value time to lives (TTLs)
  - Administrative GUI and command line shell
- A Hadoop subproject
  - The usual ASF things apply (license, JIRA, etc)

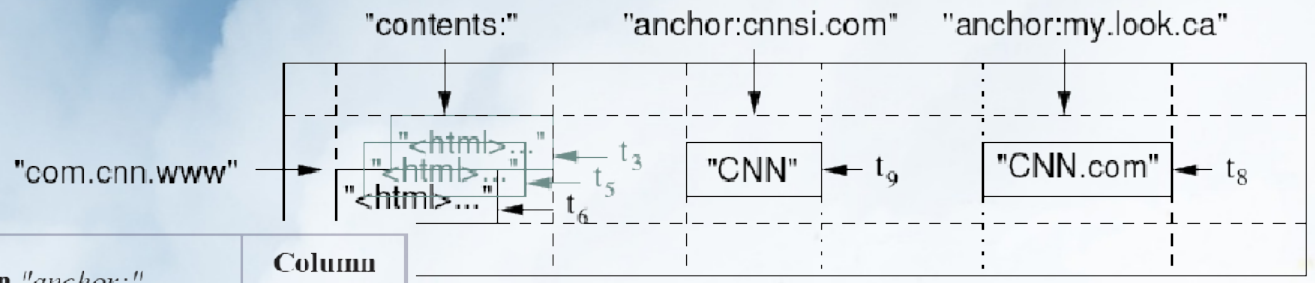
HBase – The Hadoop Database

- Designed, like the Hadoop platform, for Big Data
- To handle Big Data, we discard transactions and relational data models
  - No distributed transactions
  - No complex locking
  - No waits or deadlocks
  - Update through sort and merge instead of seek and replace
- In contrast, RDBMS systems
  - Distributed transactions, complex locking
  - Seek and replace update strategy
  - Waits and deadlocks rise non-linearly with transaction size and concurrency
    - Square of concurrency
    - Third power of transaction size
  - Relational features are abandoned at scale anyway
  - Application level sharding as a last resort is not a solution

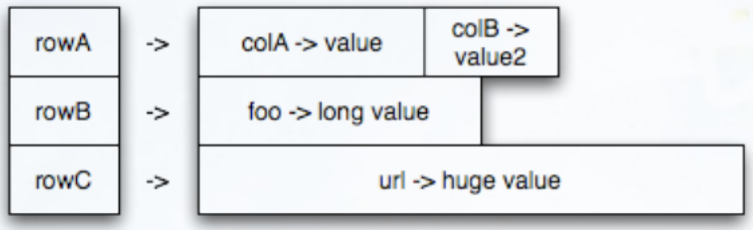
Data Model

- Distributed persistent sparse map
- Multidimensional keys

  <row>, <column>:<qualifier>, <timestamp>
- Keys are arbitrary strings
- Data grouped by columns
- Access to row data is atomic
- Multiversioning and timestamps avoid edit conflicts caused by concurrent decoupled processes



Row Key	Time Stamp	Column *"contents:"*	Column *"anchor:"*		Column *"mime:"*
"com.cnn.www"	t9		"anchor:cnnsi.com"	"CNN"	
	t8		"anchor:my.look.ca"	"CNN.com"	
	t6	"<html>..."			"text/html"
	t5	"<html>..."			
	t3	"<html>..."			

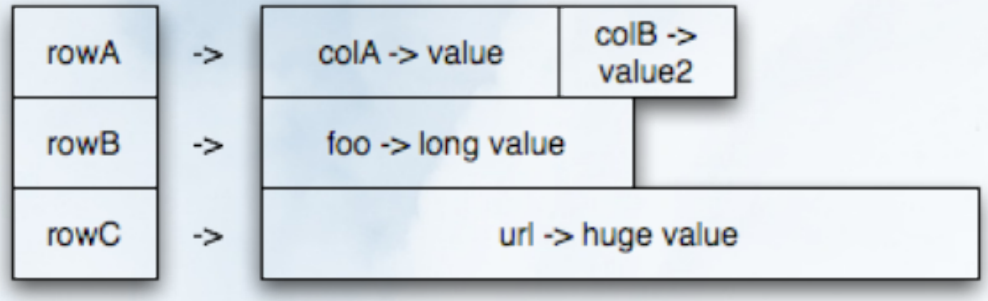


Grouped by Columns?

- Not a spreadsheet

	colA	colB	colC	colD
rowA				
rowB				
rowC			NULL?	
rowD				

- Instead, think of tags

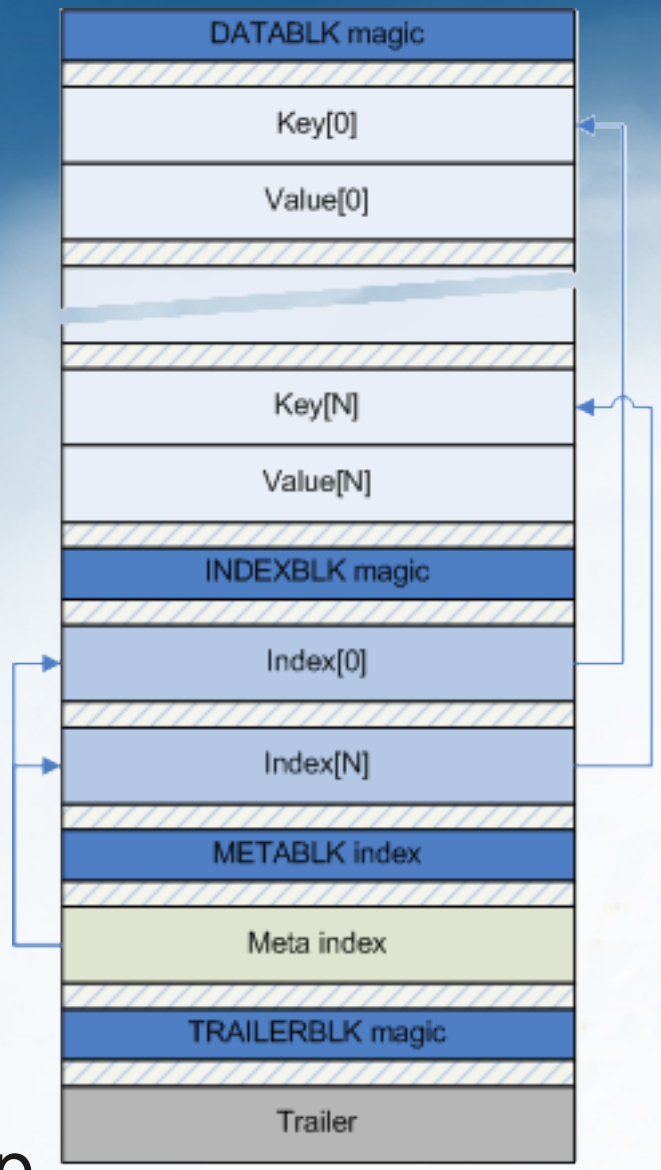


Values of any length, no predefined names or widths

Column Oriented Stores

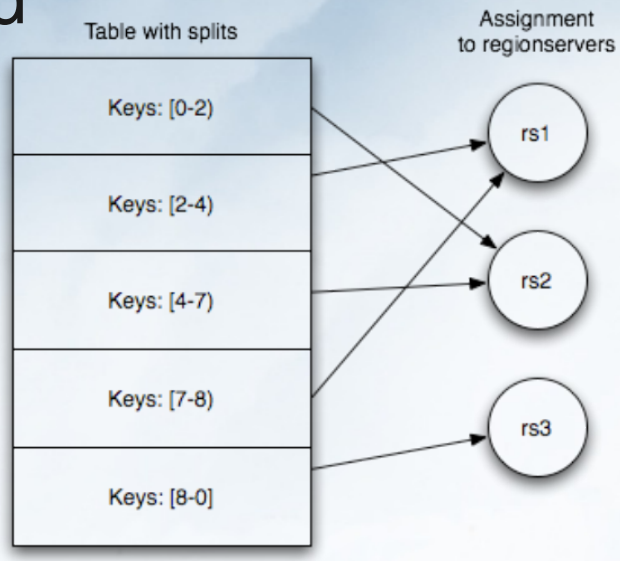
- A table consists of one or more column families
  - Column names may have an optional qualifier

    `family:qualifier`
  - Qualifiers are additional level of indexing
    - First, binary search over index (memory resident) into column store to find any matching row
    - Then scan the block to find values with matching qualifiers
- Each family in separate store files
  - Values are stored in sorted order
  - Optional file level compression (GZIP, LZO)
- Lexiographically similar values are packed adjacent to each other for good locality of I/O; it is fast and cheap to scan adjacent rows and columns



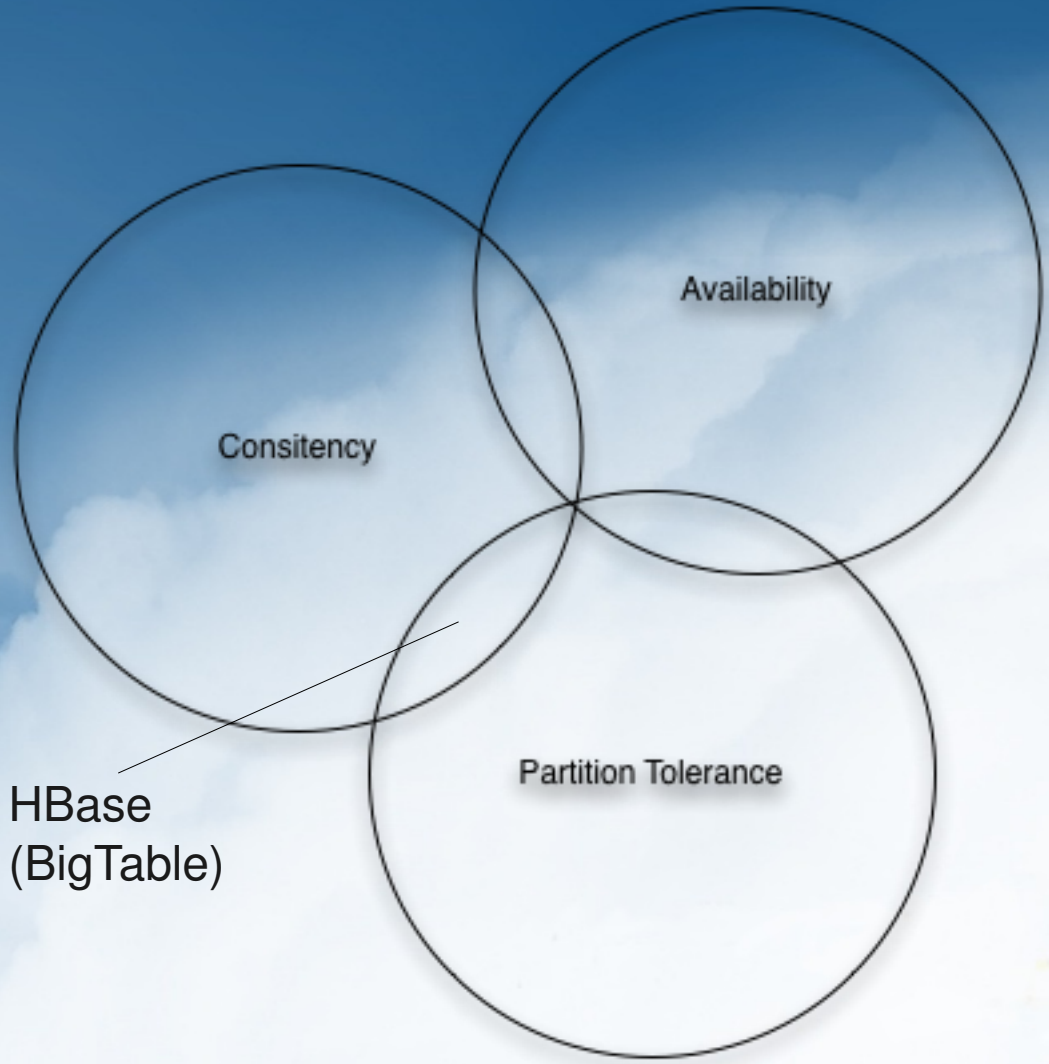
Tables And Regions

- Rows are stored in byte-lexographic sorted order
- Tables are dynamically split into regions
- Regions are hosted on a number of regionservers
- As regions grow, they are split and distributed evenly among the storage cluster to level load
  - Splits are “almost” instantaneous
  - Fine grained load balancing
  - Regions are migrated away from highly loaded nodes
  - Enables fast recovery
  - Master rapidly redeploys regions from failed nodes to others



"Strong C" (Consistency)

- CAP Theorem
  - Brewer, 2000
- Simplified definitions:
  - C: Consistency
    - Writes are visible to all readers at the same time
  - A: Availability
    - The system always returns an answer (immediately)
  - P: Partition Tolerance
    - The system can handle the failure of some nodes and loss of some messages
- You can have only 2 of 3
- BigTable is a "CP" architecture
  - Strong consistency with fault (partition) tolerance
  - Value storage is canonical: Every value appears in one region only and each region is assigned to only one region server at a time

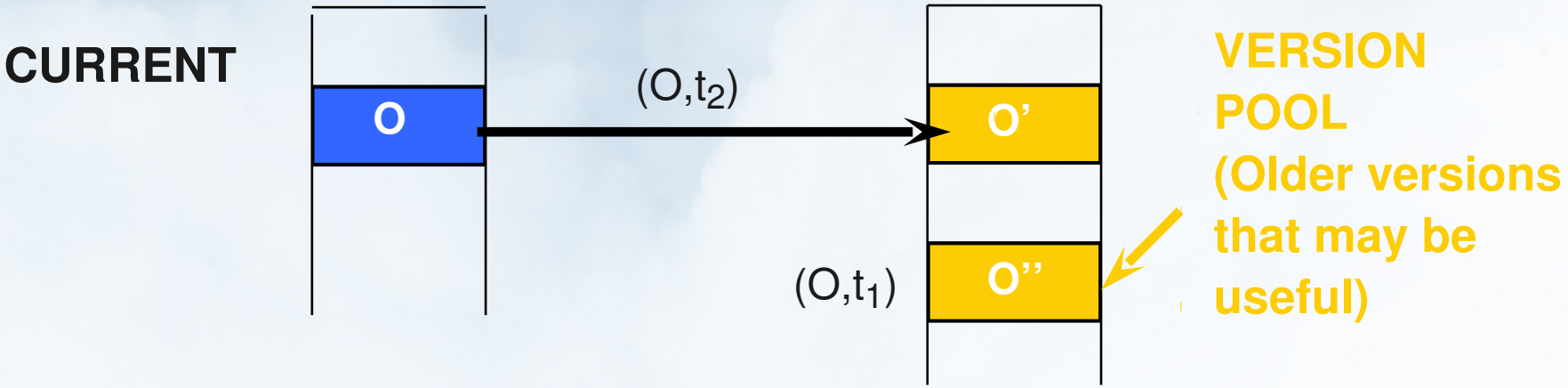


More on Consistency

- HBase is strongly consistent
  - Every value appears in one region only, within the appropriate boundary [startKey,endKey] for its row key
  - Each region is assigned to only one region server at a time
- In comparison with other "NoSQL" systems, HBase has workable atomic primitives
  - Row locks
  - Atomic compareAndSet (CAS) and increment/decrement operators
  - All mutations are atomic in the row
- Multiversioning and timestamps can help with application layer consistency concerns
  - All edits are timestamped and the storage system supports storage and retrieval of multiple versions of a value
  - Applications can query for the latest version according to timestamp, time range, or most recent *N* versions, depending on requirements

More on Consistency

- Version pool
  - Asynchronous processes update table independently
  - Microsecond resolution timestamps make edit conflicts unlikely
    - Used in conjunction with NTP clock synchronization, provides global ordering
    - Cross site replication preserves timestamps
  - Application can query N versions or by timestamp range for independently stored edits and merge or resolve them



Integration with Hadoop

- With HDFS
  - HBase relies on DFS replication for data durability and availability
  - WAL uses append feature
  - Without HDFS, regions could not be migrated
  - HBase compaction interacts faviorably with HDFS block placement
- With ZooKeeper
  - Track cluster membership and detect dead servers
  - Supports master election and recovery in multi-master deployments
  - Automatic Master failover
  - Rolling upgrades of point releases
  - Modify some cluster configuration without full cluster restart

Integration with Hadoop

- With MapReduce
  - TableInputFormat
  - TableOutputFormat
  - Splits correspond to regions for optimal I/O
  - Tasks scheduled on RegionServers hosting the table regions
- This is first class integration into the Hadoop stack

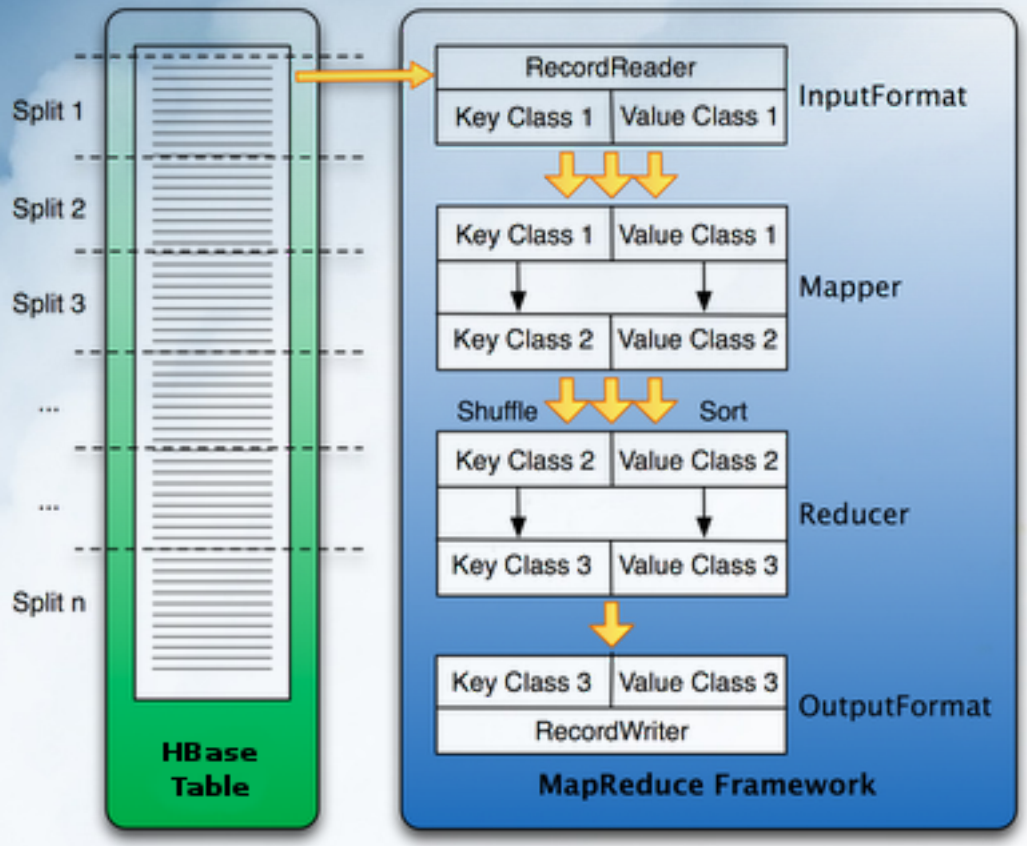


Image credit: Lars George

Integration with Hadoop

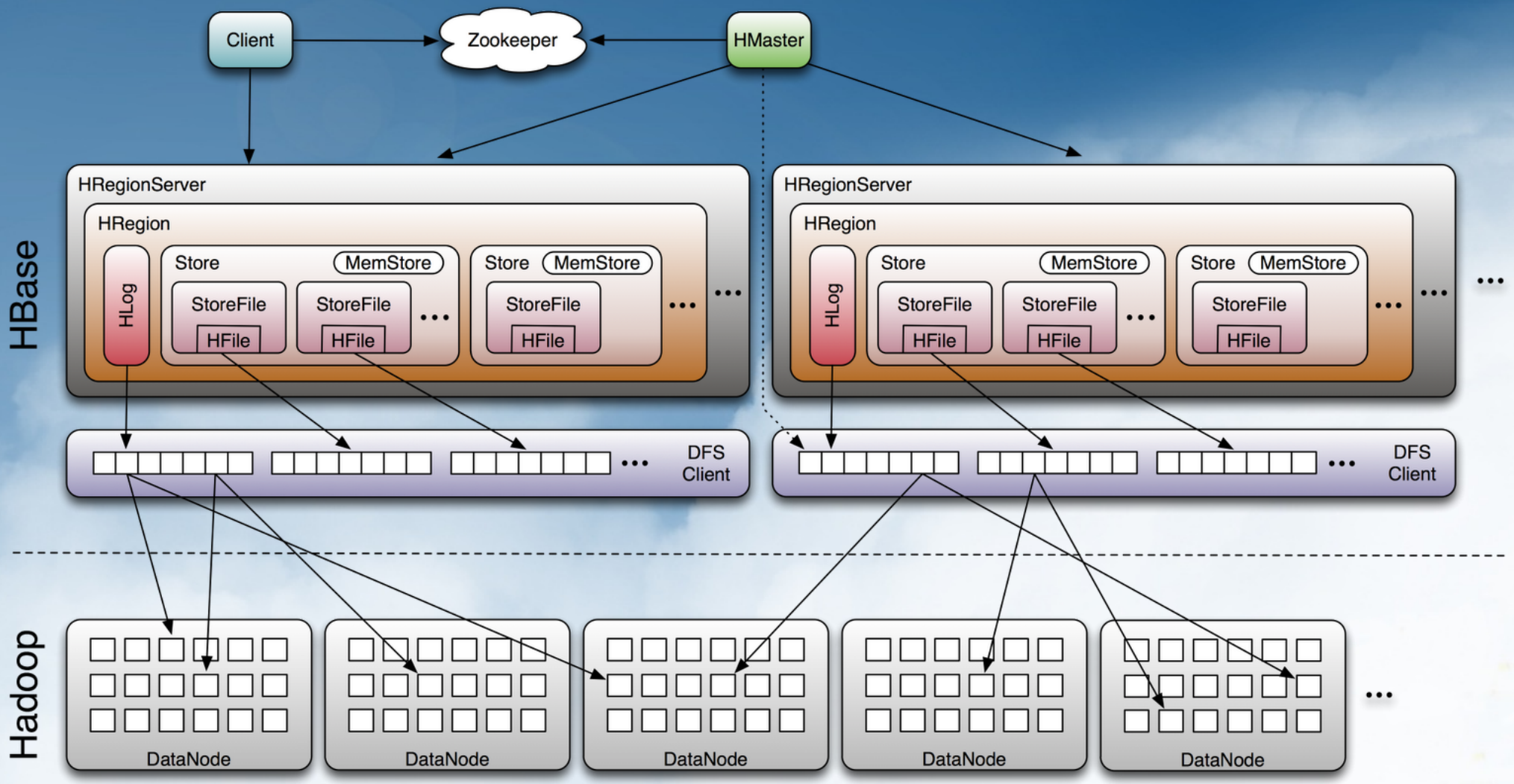
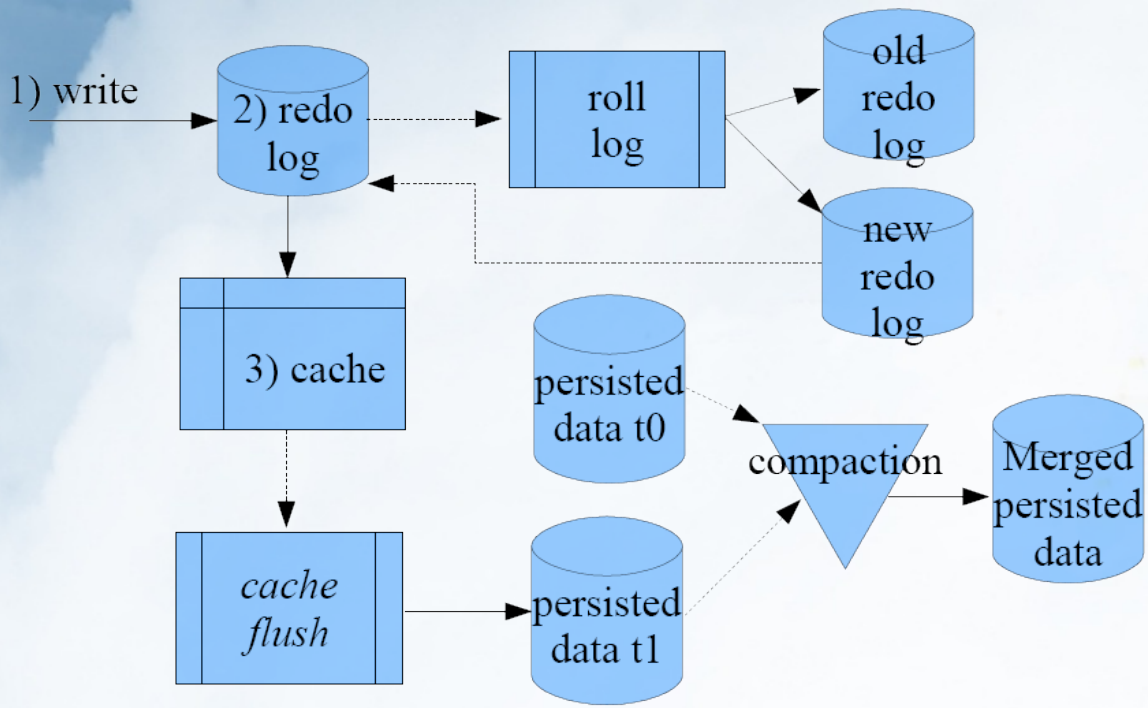


Image credit: Lars George

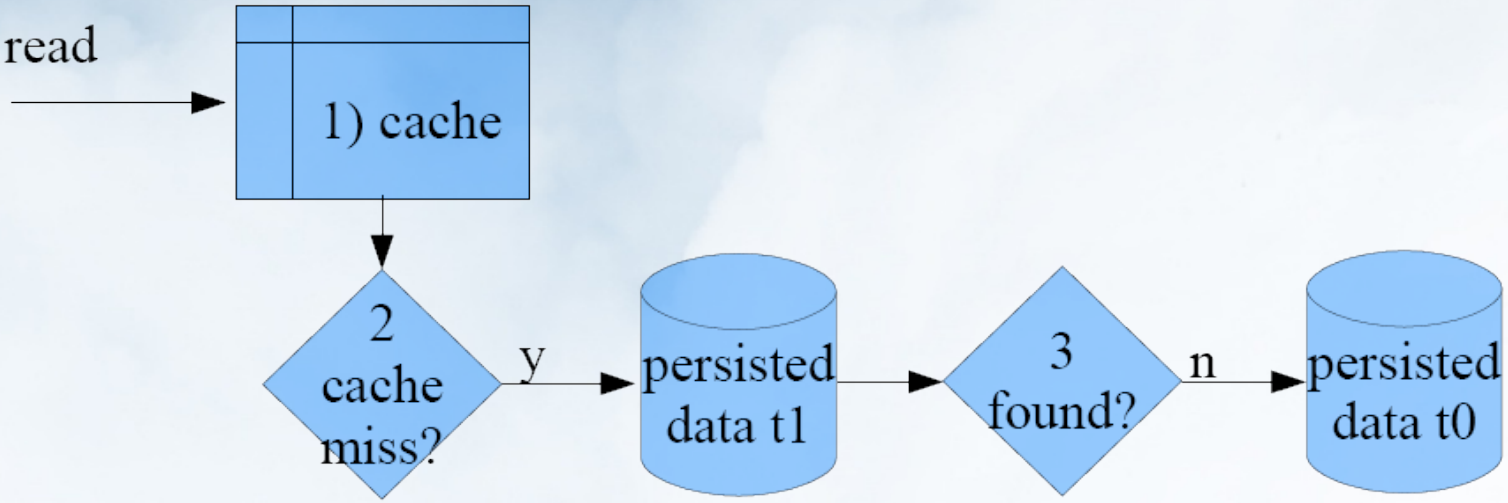
More Detail: Write Path

- When a write occurs
  - It is written to a write ahead log
  - It is buffered in memory ("memstore")
  - Deletes are just another kind of write, a delete marker
  - Periodically, the cache is written to disk, creating a new file in each store for the columns being flushed
  - A compaction occurs when the number of files in an store exceeds a threshold: all stores are merge sorted into a single new store file
  - Deleted and expired values are garbage collected during compactions
  - Compactions are done in the background
  - Periodically, the log file is closed and a new one created
  - Old log files are garbage collected
  - Updating via rewriting is 100x – 1000x faster than update via seek and replace at large scale



More Detail: Read Path

- When a read occurs
  - Check for data in memstore
  - Look for data in persisted data, from newest to oldest
  - Stop backwards search for a given { table, row, column, [timestamp] } coordinate if a delete marker is found
  - Stop search if enough versions have been found to satisfy the search criteria
  - Ignore expired values if TTLs are configured on the relevant column families
  - Expired values will be garbage collected at next compaction, just like deletes



New Features

Data Durability and ACID Guarantees

ACID Guarantees

In 0.20.4

- ACID
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability
- Up to now, the ACID guarantees that HBase has provided have not been well enumerated, but we now have a specification
  - JavaDoc in 0.20.4
  - HBASE-2294

    (https://issues.apache.org/jira/browse/HBASE-2294)
- Many of these guarantees have always been informally provided
- Some have been clarified for this release
- Next: Additions to the unit test suite to continuously validate the implementation and test for regressions

ACID Guarantees

- All mutations are atomic within a row
  - Any put will either totally succeed or totally fail
  - APIs that mutate several rows will not be atomic across the multiple rows
- The checkAndPut API happens atomically like the typical compareAndSet (CAS) operation
- The order of mutations is seen to happen in a well-defined order for each row, with no interleaving
- All rows returned via any access API will consist of a complete row that existed at some point
  - This is true across column families
- A scan is **not** a consistent view of a table; scans do not exhibit snapshot isolation*;* instead:
  - Any row returned by the scan will be a consistent view
  - A scan will always reflect a view of the data at least as new as the beginning of the scan

ACID Guarantees

- When a client receives a success response for any mutation, that mutation is immediately visible to both that client and any other client
- A row will never exhibit "time-travel" properties
  - A series of mutations moves a row sequentially through a series of states
  - Any sequence of concurrent reads will return a subsequence
- Any version of a cell that has been returned to a read operation is guaranteed to be durably stored

Data Durability

In 0.20.5

- HDFS-200
  - https://issues.apache.org/jira/browse/HDFS-200
  - A working append
- This feature is critical for HBase to be able to guarantee to clients that writes have been persisted to disk (in the write ahead log)
- Normally not a problem but there are some narrow failure cases remaining
- HDFS-200 provides working append and HBase has support for it which solves the problem
  - Developed
  - In testing
  - About 1-2 months away
  - HBase 0.20.5 with a Hadoop release including HDFS-200 will insure that if a store succeeds the data is guaranteed to be persisted

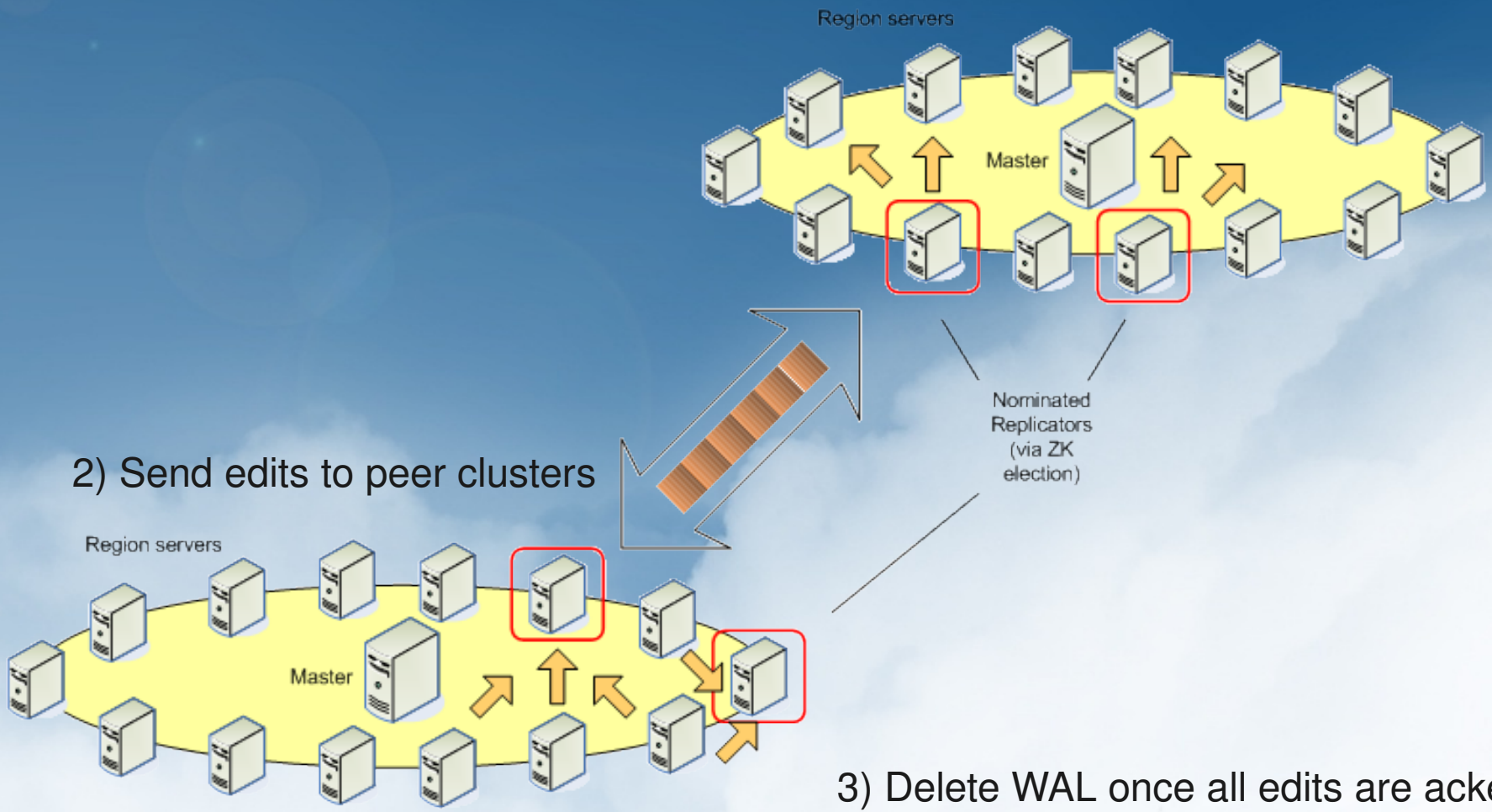
New Features

Multi Data Center Replication

Multi Data Center Replication *In 0.21.0*

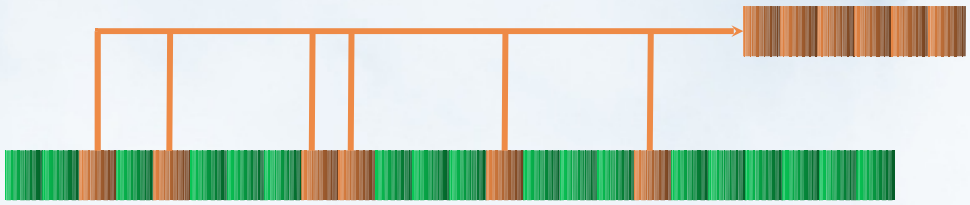
- Log shipping
  - Lazy / asynchronous
  - Update everywhere
  - Convergence with multiversioning instead of ACID
  - Use column and/or table descriptor attributes to specify the scope of data items stored there
    - Local: Do not replicate
    - Global: Replicate everywhere
    - Only replicate globally scoped cells
    - Scope is specified as an integer to enable more complex policies as they are developed in the future
- Replication is peer to peer (cluster)
  - Supports arbitrary topologies: mesh, spoke-and-wheel, tree, pipeline, etc.
- RegionServers do the work
  - A subset of RS nominate themselves via ZK to act as endpoints for inter-cluster replication
  - Ship logs between themselves in the background

Multi Data Center Replication



3) Delete WAL once all edits are acked

1) Collect global edits from newly rolled WAL



Log Backup

New Features

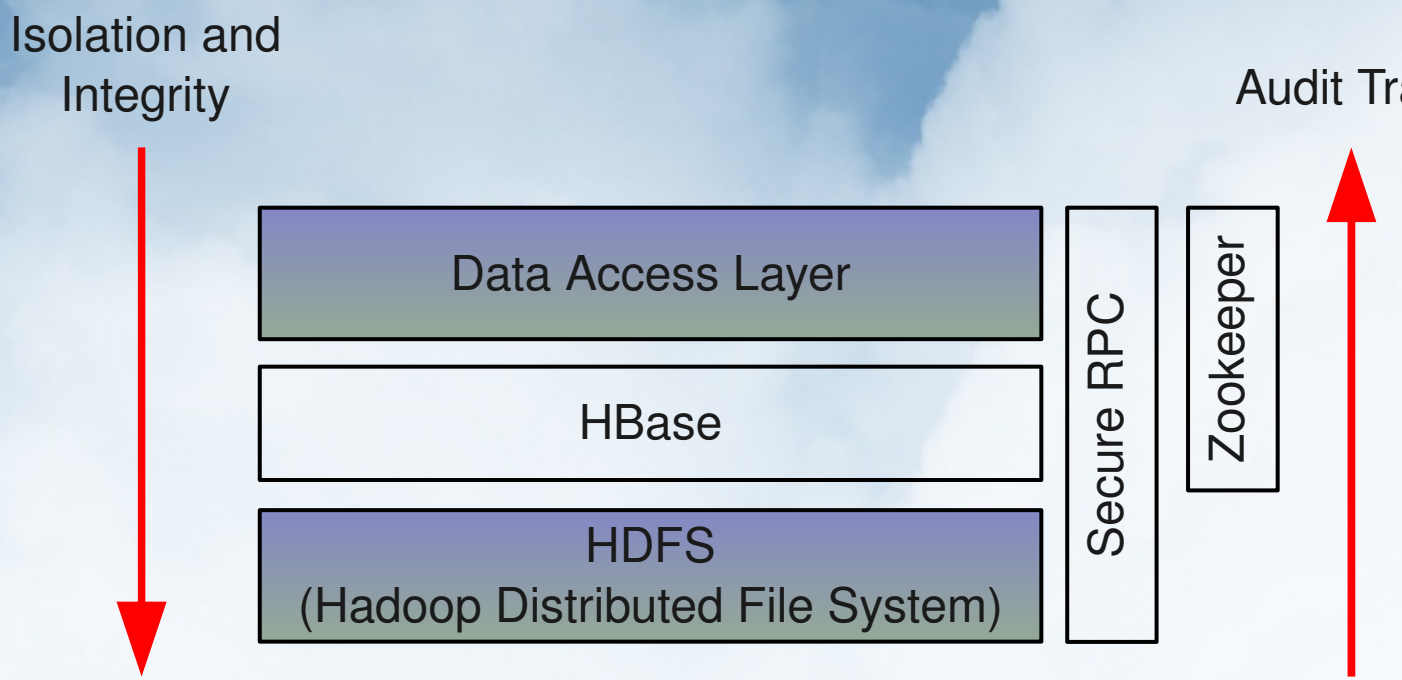
Secure HBase

Secure HBase

- Isolation guarantees
- Audit support, authentication, authorization
- New security features in Hadoop from Y!
  - Kerberos strong authentication
  - Data isolation at the HDFS layer
  - Secure RPC
  - Get HBase working on this substrate bottom up
- Add role based access control model to HBase
  - Leverage Kerberos to establish user identity
  - Manage a meta table that associates users with roles
  - ACLs on tables, possibly also on column families
  - Superuser privilege for administration
  - Integrate this into HBase top down
  - Integrate with HDFS layer security ?
- Meet efficiently and effectively in the middle

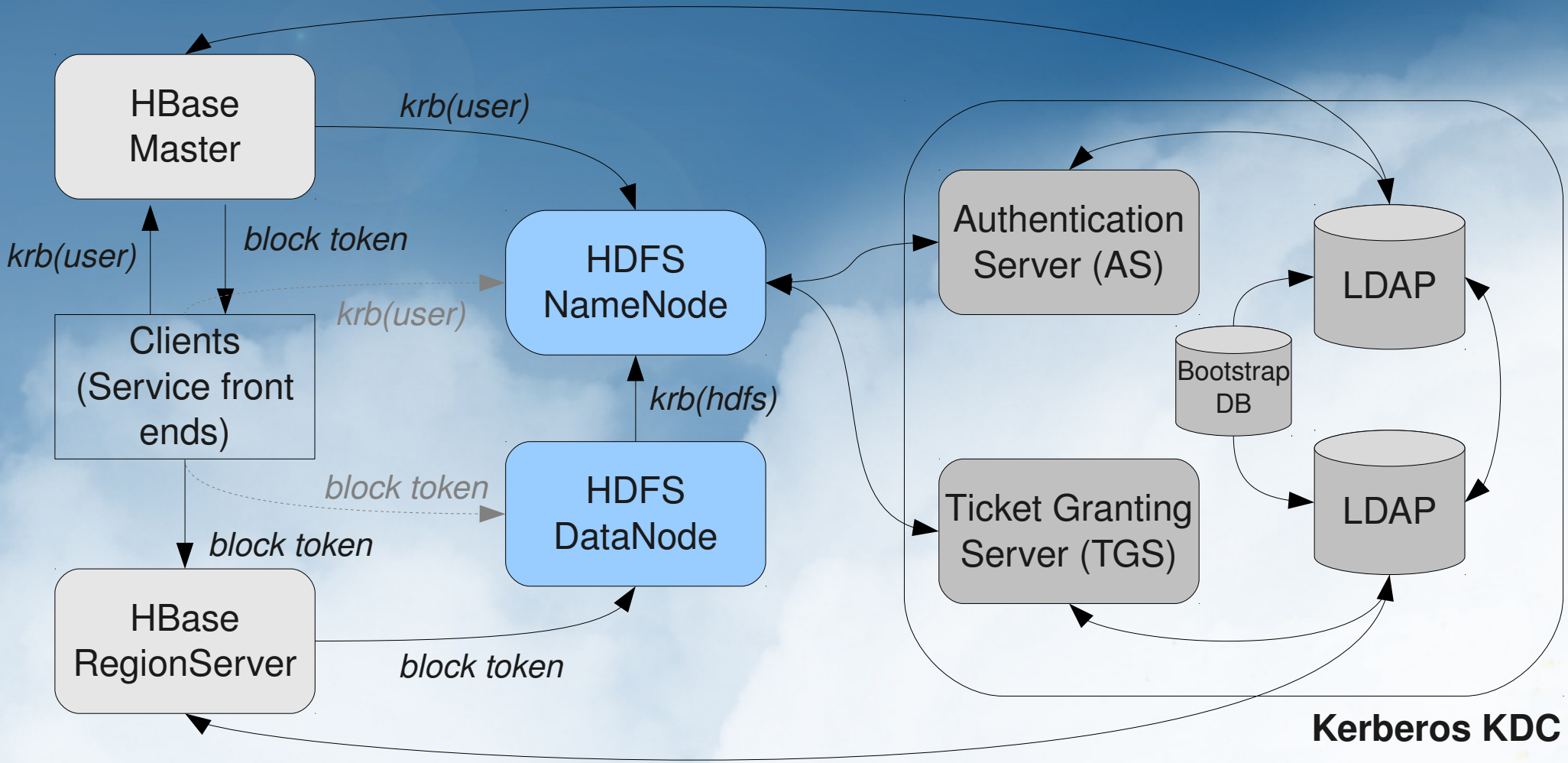
Secure HBase

- Motiviations
  - Multitenant cloud platforms and hosted services based (in part) on HBase and HDFS
  - We need to be able to reason about user isolation and data integrity controls for certification, e.g. SAS-70



Secure HBase

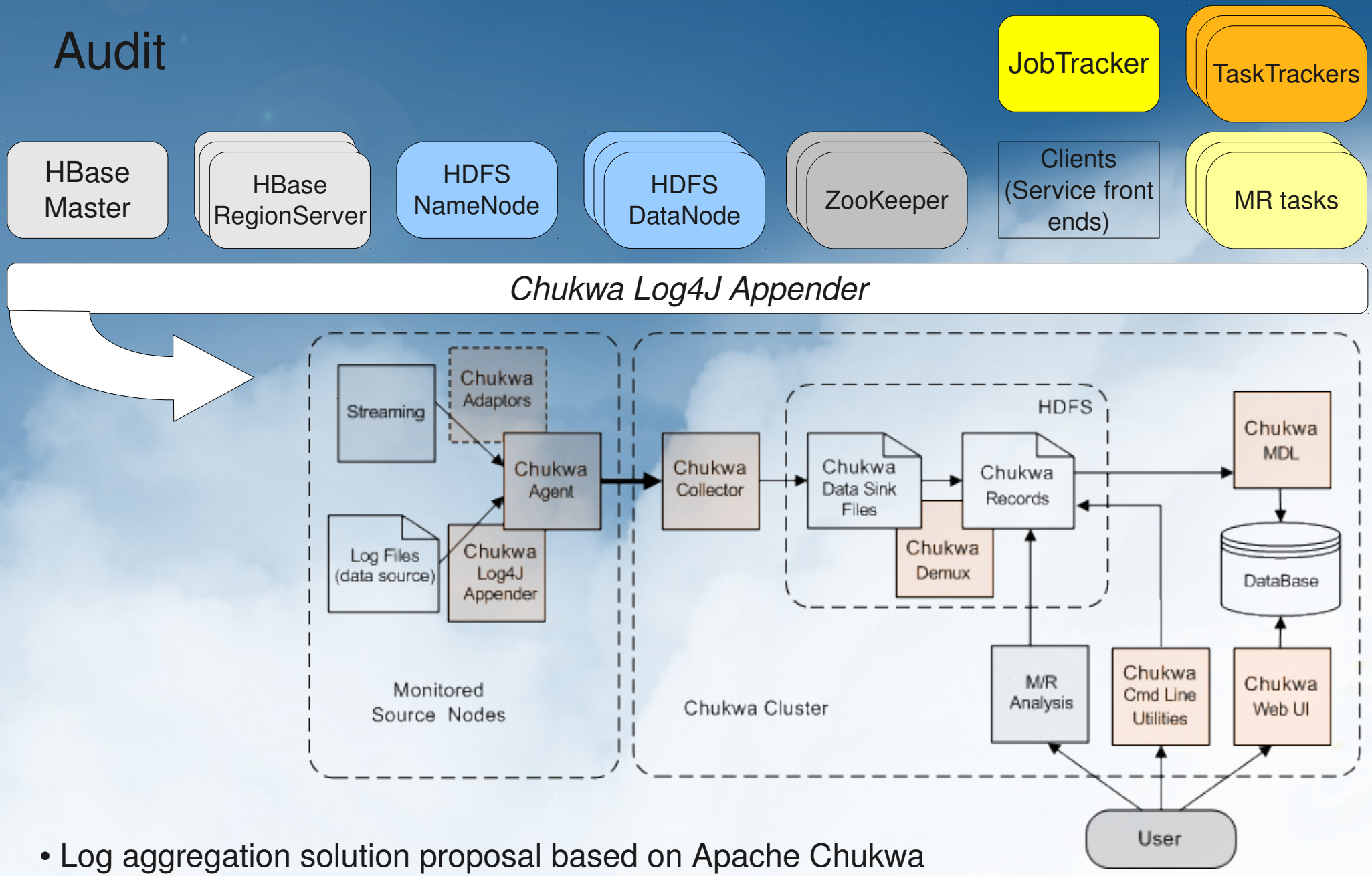
Authentication and Authorization



- Clients access HDFS and HBase services independently
- All actions requires a valid HDFS block token acquired via krb authentication to NameNode
- HDFS DataNodes will not serve reads and writes unless given valid block token for block(s)
- All RPC is secure (SASL + GSSAPI)

Secure HBase

Audit



- Log aggregation solution proposal based on Apache Chukwa
- Subset of platform resources must be reserved for private Chukwa/MR cluster

New Features

Coprocessors

Coprocessors

In 0.21

- A new BigTable Feature
- New Since OSDI'06
- Arbitrary code that runs run next to each tablet in table
  - As tablets split and move, coprocessor code automatically splits/moves too
  - High-level call interface for clients
  - Calls addressed to rows or ranges of rows
    - Coprocessor client library resolves to actual locations
    - Calls across multiple rows automatically split into multiple parallelized RPCs
- "Very flexible model for building distributed services"
- "Automatic scaling, load balancing, request routing"
- Example Coprocessor Uses
  - Scalable metadata management for Colossus
  - Distributed language model serving for machine translation system
  - Distributed query processing for full-text indexing support
  - Regular expression search support for code repository

HBase Coprocessors

- Inspired by BigTable Coprocessors
- Generic extension mechanism
  - Coprocessors are associated with tables via a table attribute
  - Table attribute is a path (e.g. HDFS URI) to jar file
  - Jar is loaded into the regionservers when table regions are opened and new function becomes part of the regionserver
- Motiviation
  - Currently extending HBase means subclassing HRegionServer and HRegionInterface
    - Resulting extensions are mutually exclusive
  - Basic Hadoop architectural principle of colocating computation with data
    - Computation here can be
      - Calculation of aggregates over region data: count(), sum(), etc.
      - Management of secondary indices
      - Dynamic indexing
      - More complex data models layered on HBase for scalability
      - Query push down with arbitrarily complex predicates

HBase Coprocessors

- RegionObserver
  - If the coprocessor implements this interface, it will be interposed in all region actions via upcalls
  - Chaining of multiple observers (by priority)
  - Can mediate (veto) actions
    - Enables security policy extensions
    - Mediators can be chained ahead of observers
  - Many extensions can be built on top of RegionObserver
    - Secondary indexes
    - Filters
    - Propagation constraints ?
- CommandTarget
  - If the coprocessor implements this interface, it can receive arbitrary method invocations from clients
- Combine RegionObserver and CommandTarget to extend HBase in arbitrary ways
  - Mapping layers, e.g. ORMs
  - Native RDF tuple store
  - Cloud filesystem

HBase Coprocessors

- MapReduce
  - If the coprocessor implements this interface, clients can call up to it execute parallel MapReduce over the HBase cluster
  - Runs concurrently on all regions of the table
  - Like Hadoop MapReduce
    - Mappers, reducers, partitioners, intermediates
  - Unlike Hadoop MapReduce
    - Not table MapReduce, parallel region MapReduce
    - Shared memory
      - For efficient implementation of aggregating functions
    - Multithreaded (worker pools)
      - Concurrency of mappers and reducers is specified separately
  - Scanner like interface to retrieve results
  - Uses leases
    - A job is only alive as long as it has a lease
    - For long running jobs the client must periodically poll status to keep it alive; jobs without interest will be cancelled
    - Retrieval by "scanner" will also renew the lease

Wrap Up

For More Information

- HBase Website and Wiki
  - http://hbase.org/
- Mailing List
  - http://hadoop.apache.org/hbase/mailing_lists.html
- IRC Channel
  - #hbase on Freenode
  - Committers and core contributors are here on a regular basis
  - More active than the Hadoop forums
- Follow HBase on Twitter!
  - @hbase

The End

- Q&A
- Contact info:

Andrew Purtell
apurtell@apache.org
andrew_purtell@trendmicro.com