HBase

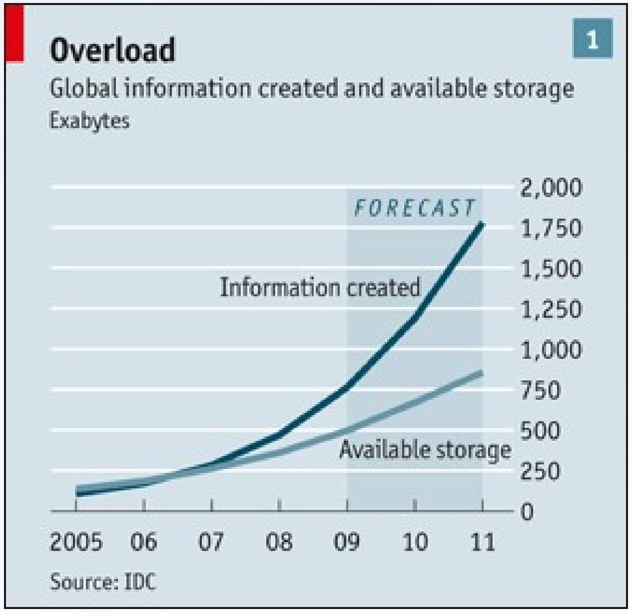
Introduction

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

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

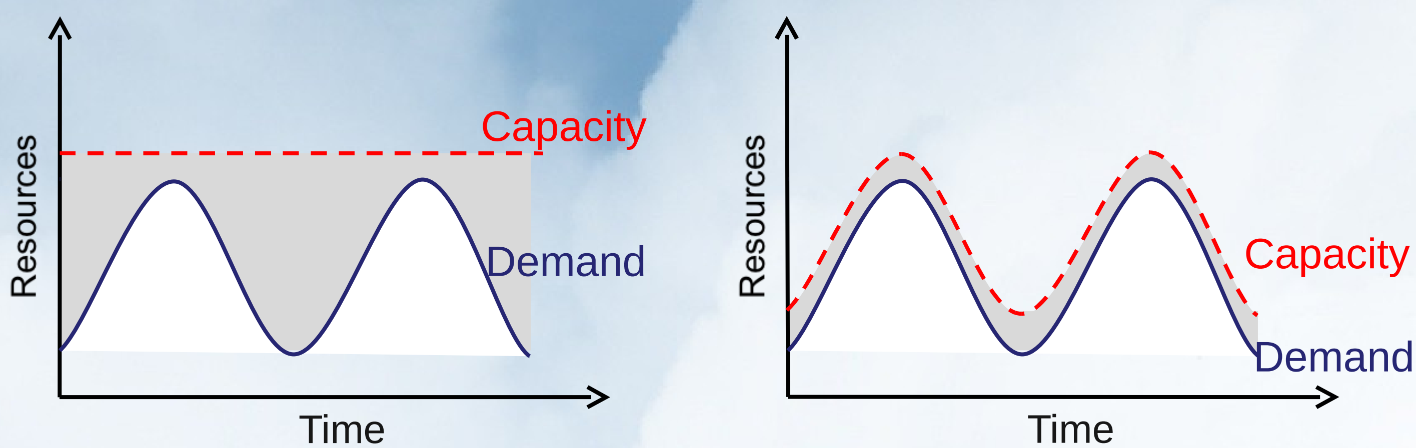
- "Internet-based access to highly scalable pay-per-use IT capabilities"

  - Ynema Mangum, SUN Microsystems

- 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

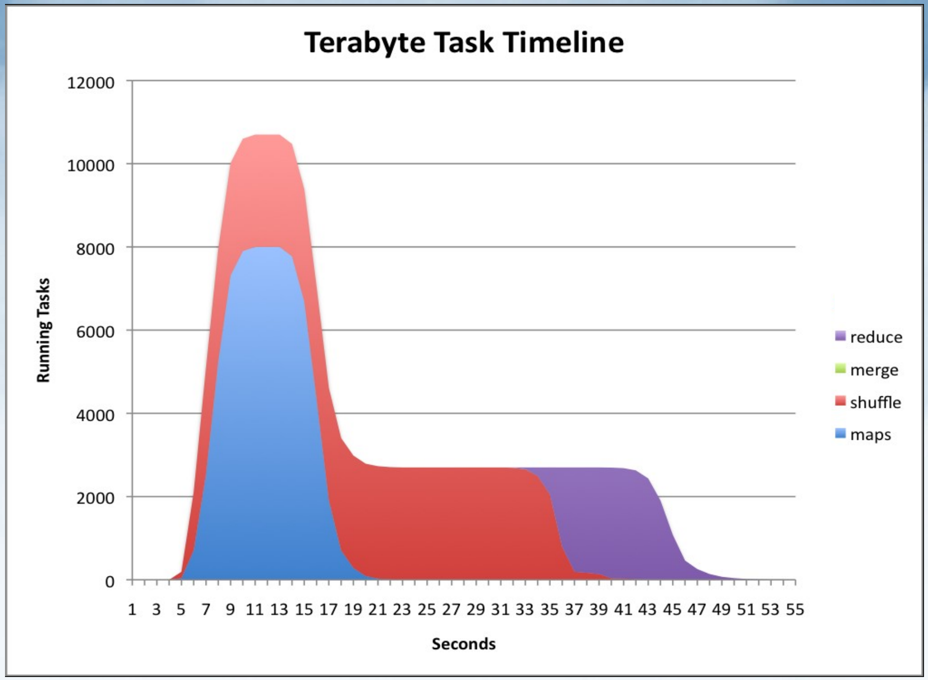
Unused resources

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

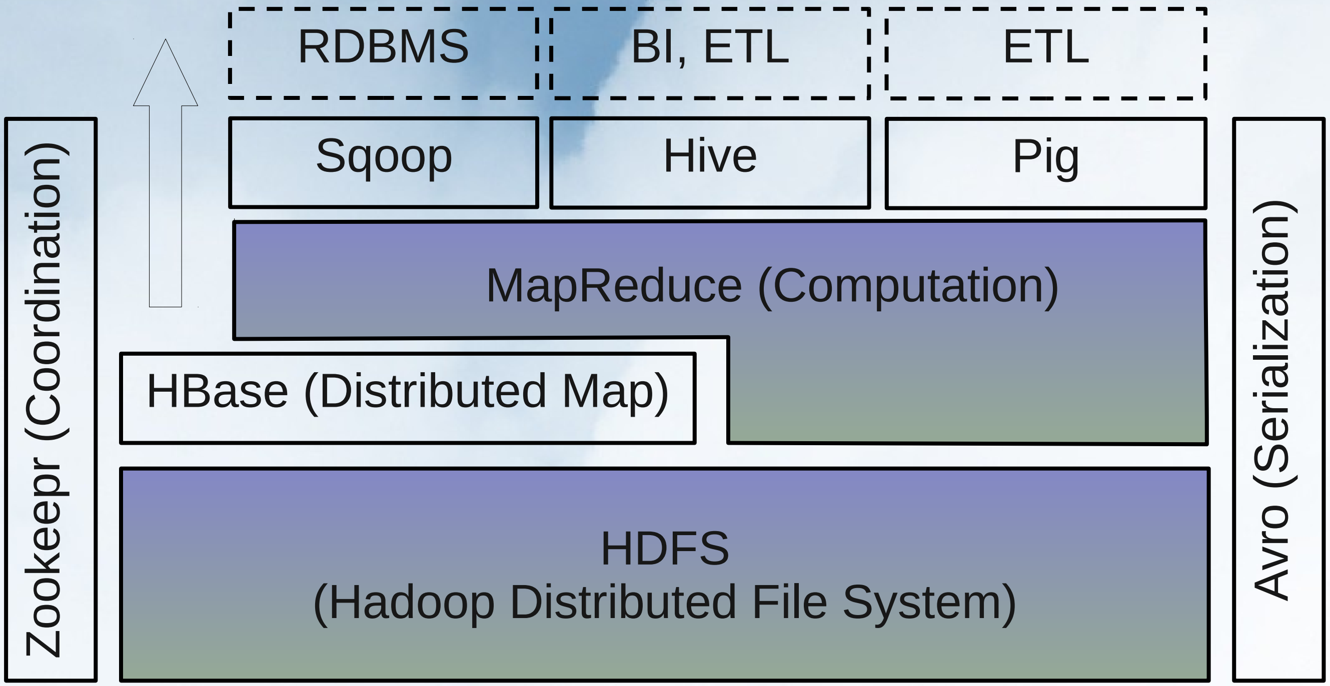
Hadoop – The Platform

- An elastic and 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

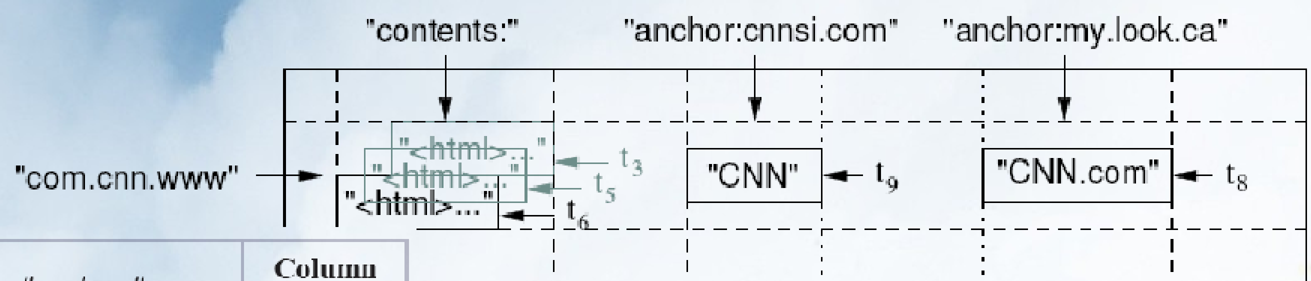
  → Timestamps

HBase – The Hadoop Database

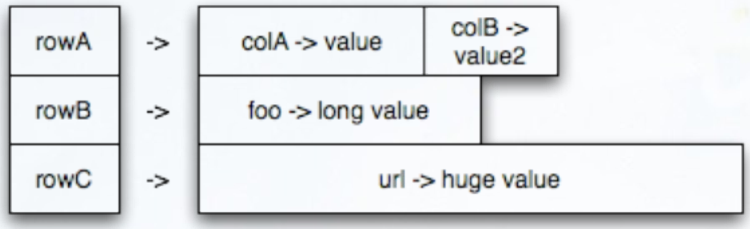
- Google: “*BigTable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers*”
- Goal: Store billions of rows * millions of columns * thousands of versions
- An open source version of BigTable, enhanced with additional features developed by the community
- A Hadoop subproject
  - The usual ASF things apply (license, JIRA, etc)
- 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

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>..."			

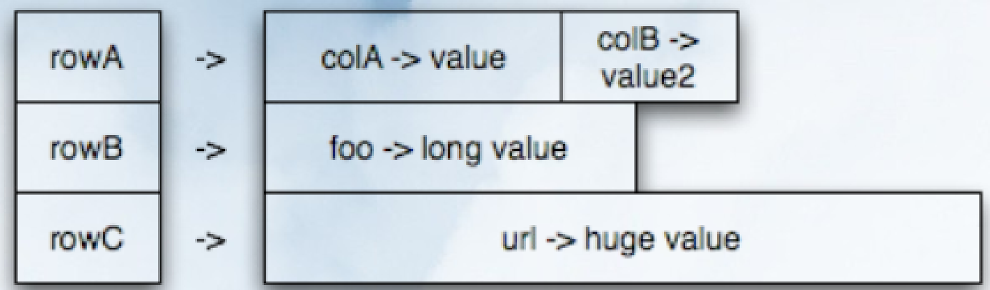


Grouped by Columns?

- Not a spreadsheet

	colA	colB	colC	colD
rowA				
rowB				
rowC			NULL?	
rowD				

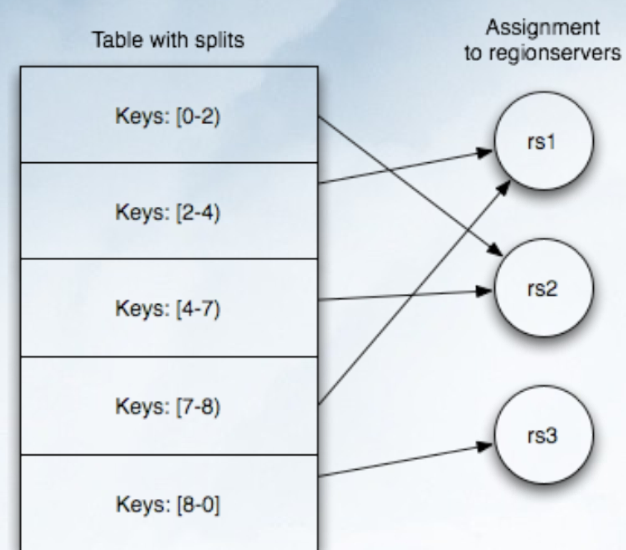
- Instead, think of tags



Values of any length, no predefined names or widths

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



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

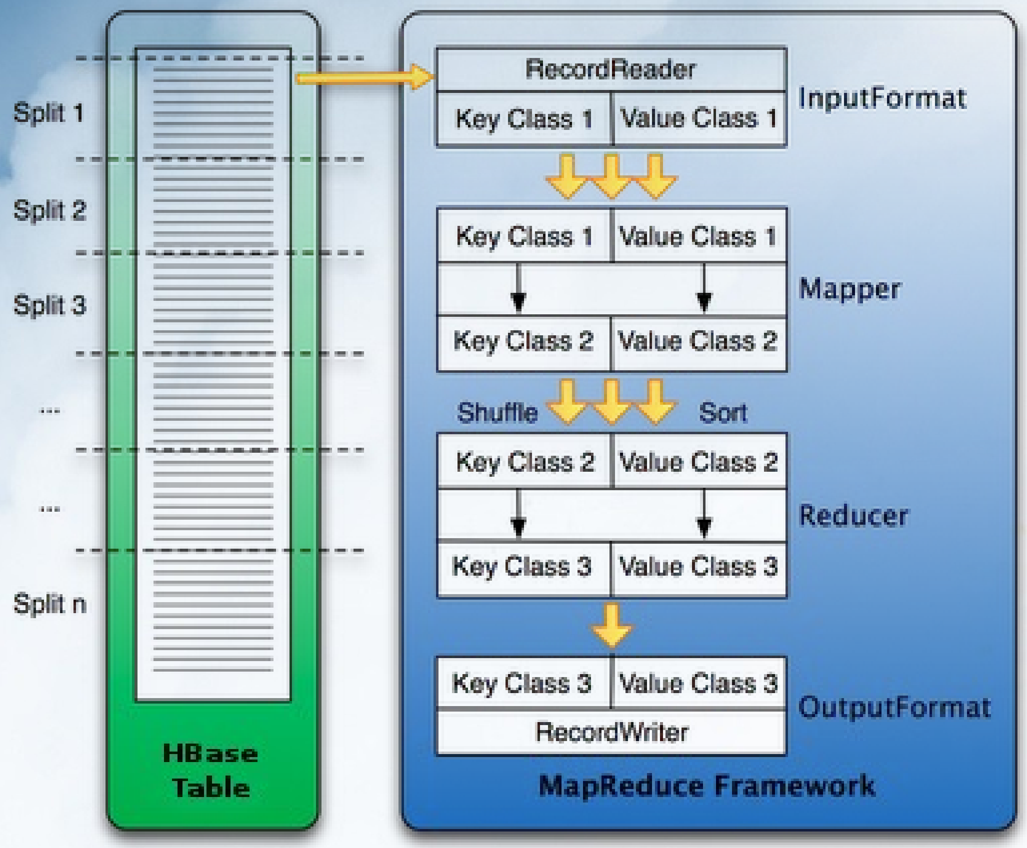


Image credit: Lars George

Integration with Hadoop

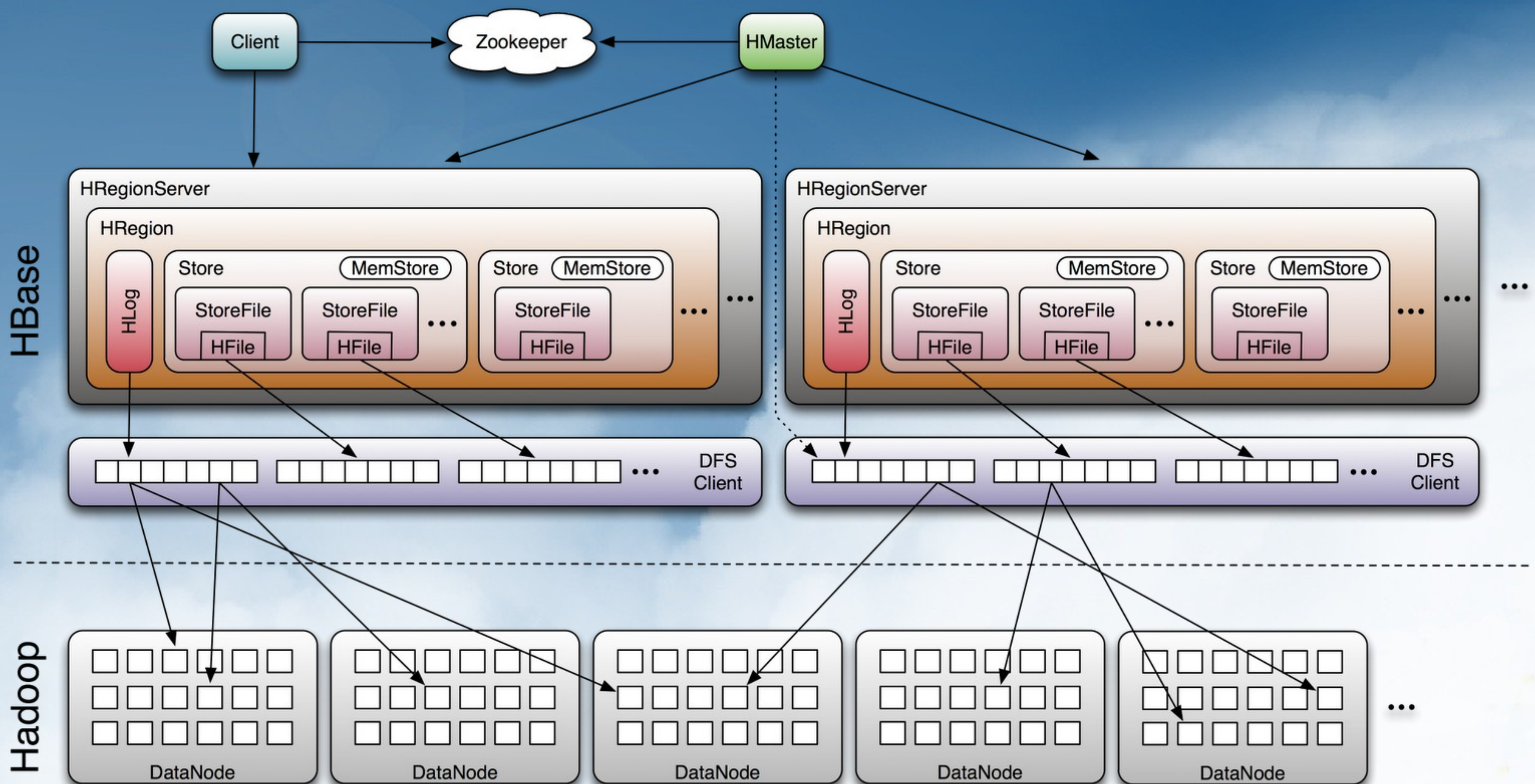


Image credit: Lars George

The End

- Q&A

- Contact info:

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

HBase

Coprocessors

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

TREND MICRO™ TCloud Computing H·BASE

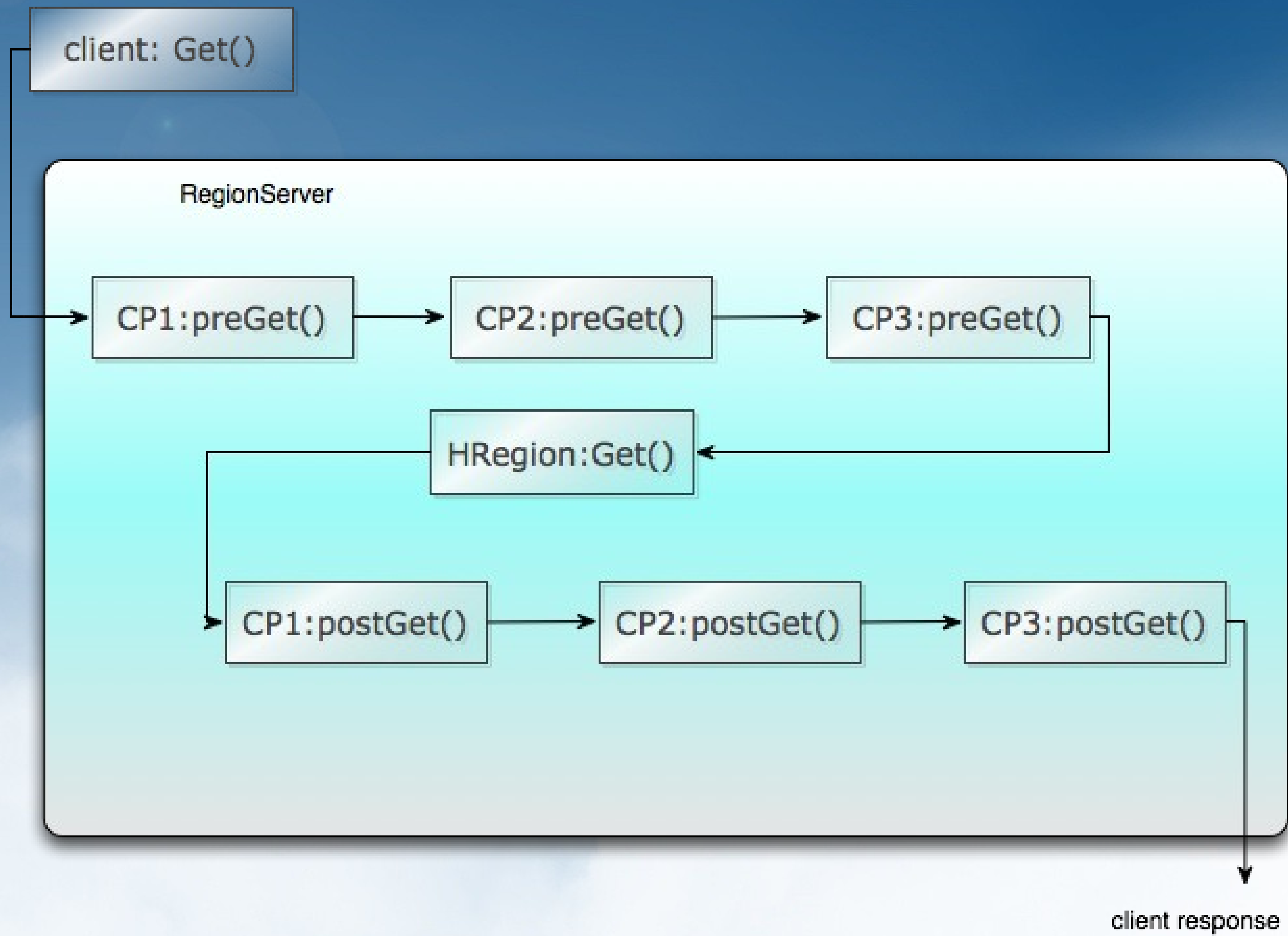
BigTable Coprocessors

- Inspired by Google Bigtable Coprocessors (Jeff Dean's keynote talk at LADIS 09)
  - Arbitrary code that runs at each tablet in table server
  - 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 RPC
- Very flexible model for building distributed services
  - Automatic scaling, load balancing, request routing for app
- Example use cases
  - Scalable metadata management for Colossus
  - Distributed language model serving for machine translation system
  - Distributed query processing for fulltext indexing support
  - Regular expression search support for code repository

HBase Coprocessors

- Inspired by BigTable Coprocessors
  - Also a generic extension mechanism
  - But reimagined as a server extension framework
- Coprocessors extend base HBase function
  - In the site configuration (*system coprocessors*)
  - Using a table attribute (*table coprocessors*)
    - Table attribute is a path (e.g. HDFS URI) to jar file
    - Jar is loaded into the regionservers when table regions are opened
  - New functionality becomes part of the regionserver implementation and runs in process
  - Lifecycle methods
  - RegionObserver: Watch or change when clients interact with regions
  - Endpoint: Provide a new service via dynamic RPC
- No more mutually exclusive subclassing of RegionServer implementation and RPC interfaces!

Example



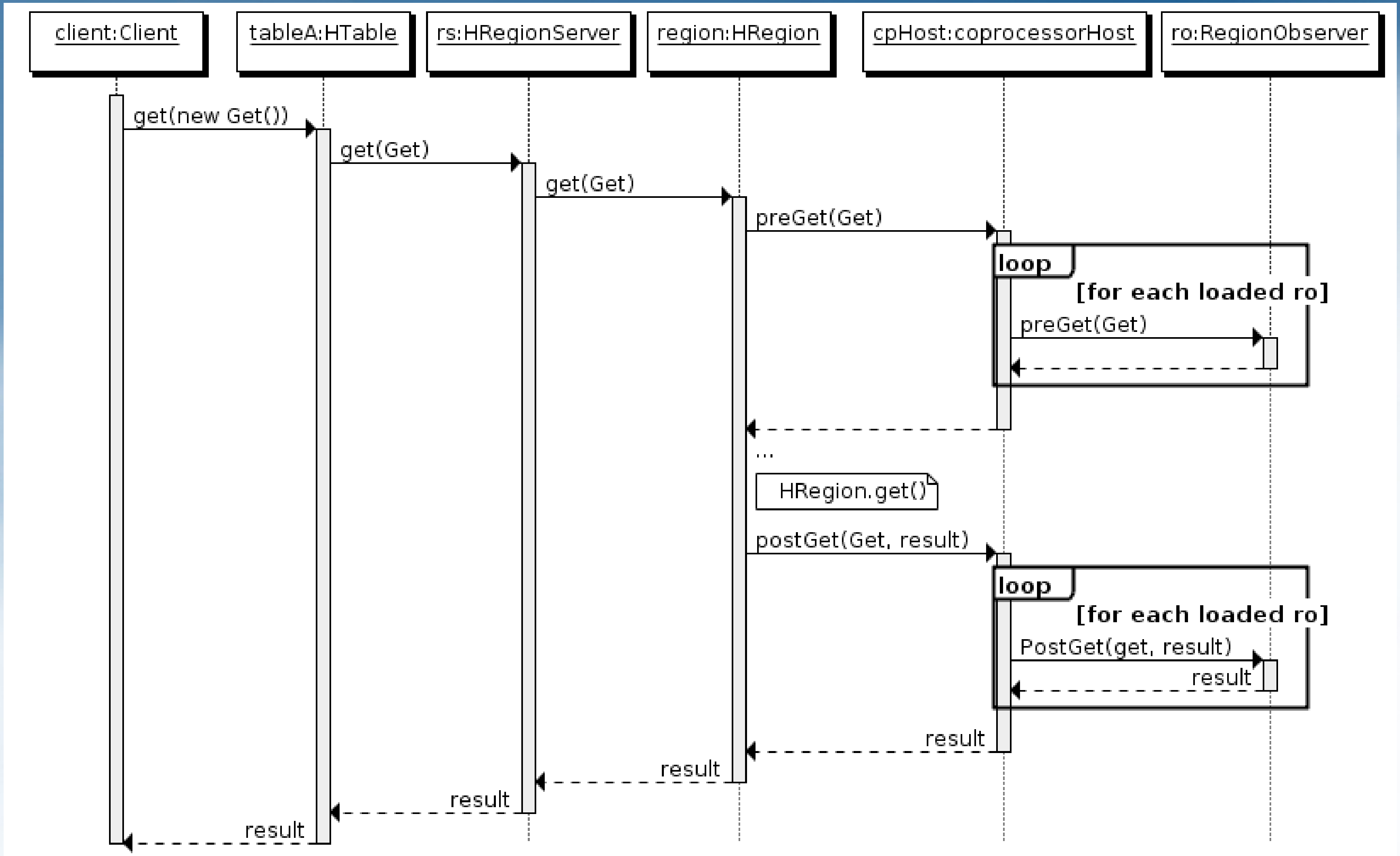
client: Get()
RegionServer
CP1:preGet()
CP2:preGet()
CP3:preGet()
HRegion:Get()
CP1:postGet()
CP2:postGet()
CP3:postGet()
client response

Extension Interfaces

- Coprocessor
  - Basic lifecycle events: Start, stop
  - Region housekeeping: Open, flush, compact, split, close
- Observer
  - RegionObserver
    - If a coprocessor implements this interface, it will be interposed in all region actions via upcalls
      - Provides hooks for client side requests: get, put, delete, etc.
      - Chaining of multiple observers by priority; mediators can be chained ahead of watchers to implement security policy extensions
  - MasterObserver
    - If a coprocessor implements this interface, it can intercept administrative actions taken at the master for a region (load balance, enable/disable, etc.)
  - How to develop an Observer
    - Implement interface and override upcall methods

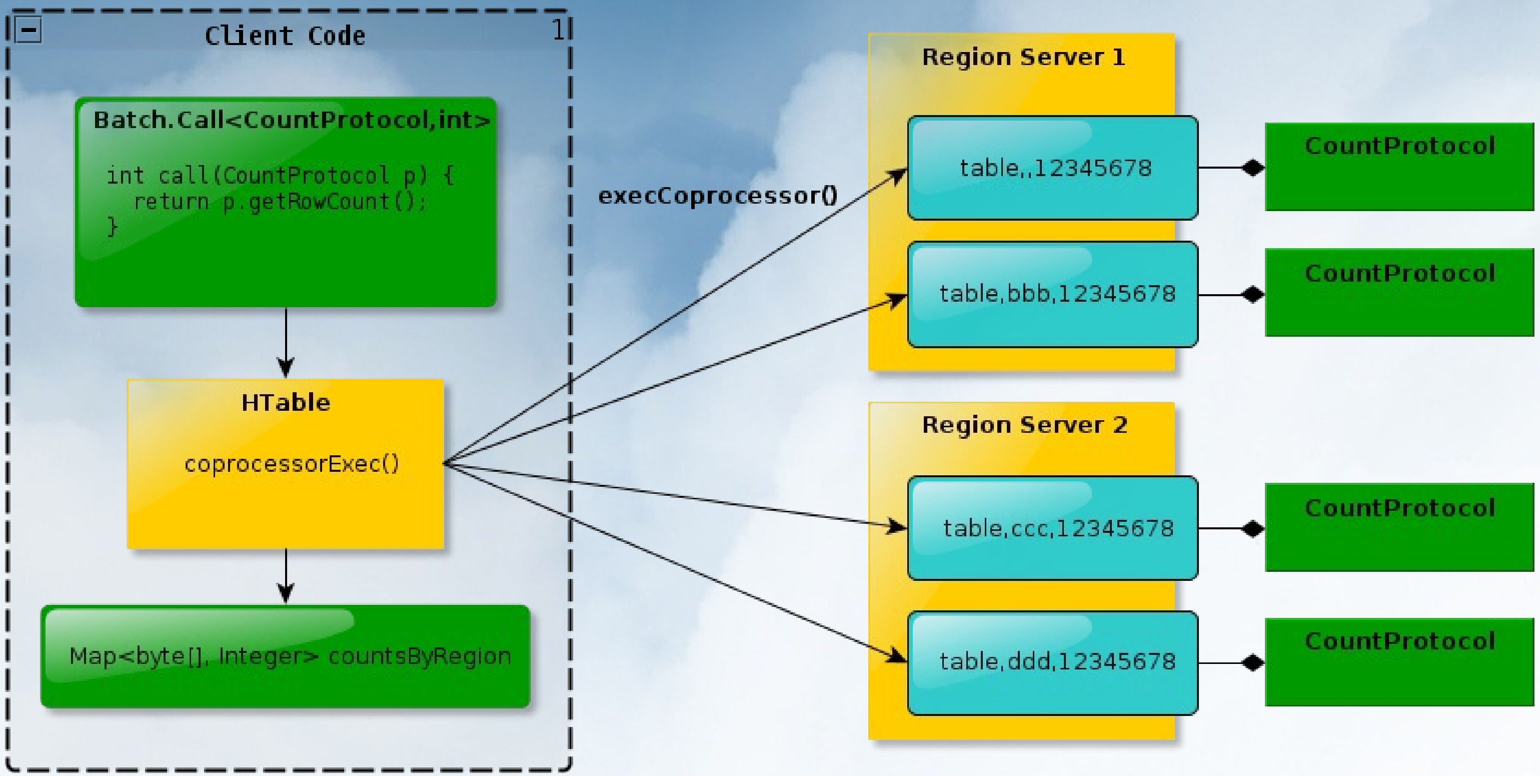
Extension Interfaces (cont.)

- Observer



Extension Interfaces (cont.)

- Endpoint
  - Provides a way to define one's own protocol communicated between client and RegionServer, and execute arbitrary code in the RegionServer process
  - The communication protocol between the HBase client and RegionServer is extended at runtime without recompilation



Extension Interfaces (cont.)

- Endpoint
  - How to develop an Endpoint
    - Define the protocol interface (extends CoprocessorProtocol)
    - Implement this protocol interface
      - Extend BaseCommandTarget so protocol will be automatically registered at coprocessor load
    - On client side, the Endpoint can be triggered by:
      - HTable.proxy() - single region
      - HTable.exec() - region range

TREND MICRO TCloud Computing HBASE

Current Projects

- Access control
  - Build access control into HBase using the Coprocessor framework
    - Fine grained execute permissions by table, role, creating user/role, executing user/role
    - Different security models can be implemented as coprocessors
  - Meta JIRA is HBASE-1697
  - Some early work as HBASE-3025
    - Reject data access by nonauthorized user according to ACL
    - Keep ACLs in META table
    - Use ZooKeeper to propagate ACL changes made to META via put() or delete() to permissions caches on all RegionServers
  - Remaining work
    - Add Kerberos plugin to ZooKeeper so ZK auth and ACLs can be managed in a seamless manner with HBase ones
    - Master side coprocessor hooks to get control over admin ops

Current Projects

- Aggregate operators
  - HBASE-1512
  - Add aggregate table ops as dynamic RPC extensions via Endpoint
    - count(), sum(), average(), etc.
  - Preliminary patch on issue
  - If you have interest, we encourage you to participate in the design of this feature via the JIRA

Interesting Ideas

- Computational frameworks
  - Cascading (cascading.org) execution target: Compiler and runtime support for partitioning work over HBase cluster and executing assemblies in parallel where the data is located
  - Streaming data processing framework
- HDFS-DNN
  - Proof-of-concept code for replacing the HDFS NameNode with HBase
  - This is roughly the architecture of Google's GFS2
  - Scalable nameservice, no more NameNode singleton reliability, availability, and scalability concerns
- FuzzyTable
  - Fuzzy matching against keys that encode high dimensional data
  - Application domain is low latency biometrics search

Future Direction

- Code weaving
  - Start with allowing arbitrary code
  - Use a rewriting framework like ASM to weave in policies at load time
  - Build policies over time which improve fault isolation and system integrity protections
    - Wrap heap allocations to enforce limits
    - Insert monitor code in loop headers to detect and throw execeptions if CPU time limits are exceeded, e.g. Infinite loops
    - Reject APIs considered unsafe
- Parallel computation framework
  - Hadoop MapReduce API (mappers, reducers, partitioners, intermediates) but parallel region MapReduce ?
  - Stream processing paradigm
    - Cascading or S4

The End

- Q&A

- Contact info:

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

TREND MICRO™ TCloud Computing H·BASE

Secure HBase

Hadoop Group @ Trend Micro: *Andrew Purtell, Gary Helmling, Joshua Ho, Eugene Koontz, Mingjie Lai*

TREND MICRO™

H·BASE

Background

- No real security in Hadoop 0.20.x and prior
  - User impersonation trivial
  - No mutual client/server authentication
  - File permission enforcement assumes good actors
  - Instead clusters secured at the perimeter
  - With no FS security, HBase security would make no sense
- Security features in Yahoo Hadoop 0.20.S and ASF Hadoop 0.21+
  - Strong authentication using Kerberos
  - Mutual authentication of RPC connections
  - Data isolation at HDFS level
  - Multiple groups can share the same cluster
  - HBase security now a possibility

TREND MICRO™

Overview

- Why?
  - User isolation (control over your data)
  - Multi-tenancy: private and public cloud
- What is it?
  - Client access to HBase is authenticated
  - User data is private unless access has been granted
  - Access to data can be granted at a table or per column family basis
- What is it not?
  - Row-level or per value (cell)
  - Push down of file ownership to HDFS
  - Full Role Based Access Control

Concepts

- Authentication
  - Who are you?
- Authorization
  - Can user A do action X within context Y?
- Isolation
  - System-wide concern
  - Requires enforcement of authorization internally
  - Authorization useless if system leaks data in other ways
    - Observability of storage files
    - Eavesdropping on data in transit

Authentication

- Need to be able to confirm identities
- Building on Secure Hadoop RPC
  - Client authentication based on Kerberos
    - allows servers to verify client credentials with trusted third party
  - Secure RPC based on SASL
    - can provide confidential communications between HBase clients and servers via GSSAPI/Kerberos
    - also supports DIGEST-MD5 authentication, allowing Hadoop delegation token use for MapReduce

TREND MICRO™

Authorization

- Default deny policy – full user isolation
- Permissions
  - Read, Write, Execute, Create, Admin
- Permission Grants
  - Principal: user or group
  - Scope: table, optional column family
  - Permissions
- Built-in Roles
  - Superuser: full access
  - Table owner: full access to table, plus delegation

Isolation

- Optional RPC encryption with SASL
- Secure Hadoop
  - HDFS permissions to restrict access
  - Table HFiles owned by HBase system user
- HBase mediates access to table data
  - column family granularity
  - -ROOT- and .META. readable by all
    - potential leakage of row key data

TREND MICRO™

How?

- Access Control Lists (ACLs)
  - Store combination of principal, scope, permissions
- Coprocessors
  - Intercept data operations to perform authorization checks
- Permission Synchronization : .META. to ZK to RegionServers
  - Reflects ACL changes across entire cluster

TREND MICRO™

Access Control Lists

Canonical Storage : .META.'s acl: column family

Logical		
Principal	**Table:ColumnFamily**	**Permissions**
humphrey	foo:*	{READ,WRITE}
.META. acl:		
Row	**Qualifier**	**Value**
foo,,1286569...	humphrey	RW

Client interface

- hbase shell 'grant' command:

```
hbase> create 'foo', 'f1'
hbase> grant 'herbert', 'RW', 'foo'
hbase> scan '.META.'
ROW                   COLUMN+CELL
  foo,,1286569...     column=acl:herbert, timestamp=1286569..., value=RW
...
```

TREND MICRO™

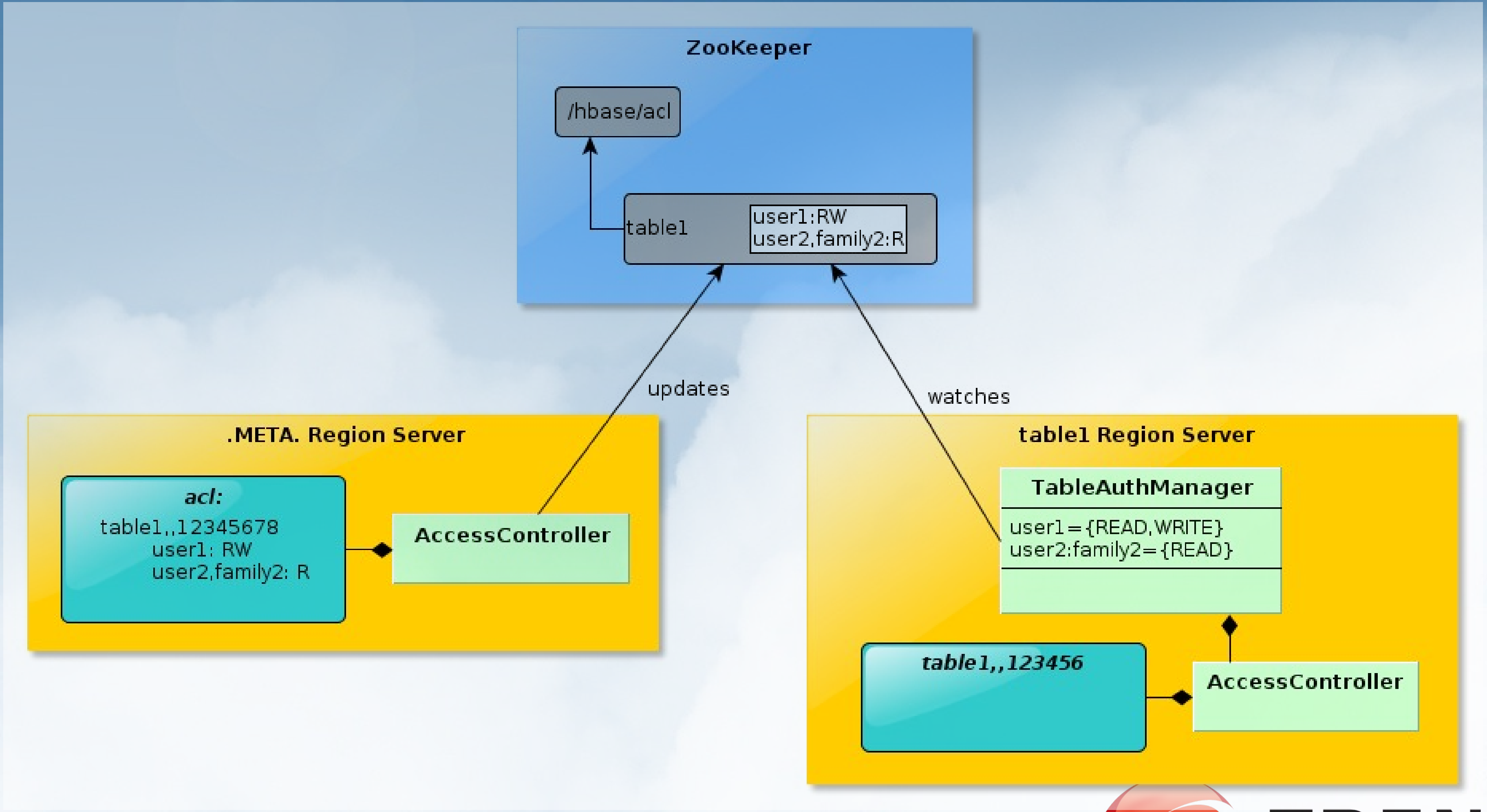
Coprocessors

- AccessController: the "checking" in permissions checking
  - Anatomy of a client request

	Component	User	Action	Class	Method
1	client	U	sends IPC call C: <U,'get',T> to regionserver.	HTable	get()
2	regionserver	S	Receives C:<U,'get',T>.	HBase Server	processData()
3	regionserver	S→U	`U.doAs('get',T)`		run()
4	regionserver	U	Check in-memory Permission Mirror: `UserPerms = getUserPermissions(UserGroupInformation.getCurrentUser(),T)`	AccessController	preGet()
5	regionserver	U	`if (UserPerms imply Get) then return; else throw AccessDeniedException;`		
6	client	U	Receives either get() return value or AccessDeniedException	HTable	get()

Synchronizing ACLs

- .META. -> ZK -> Region servers



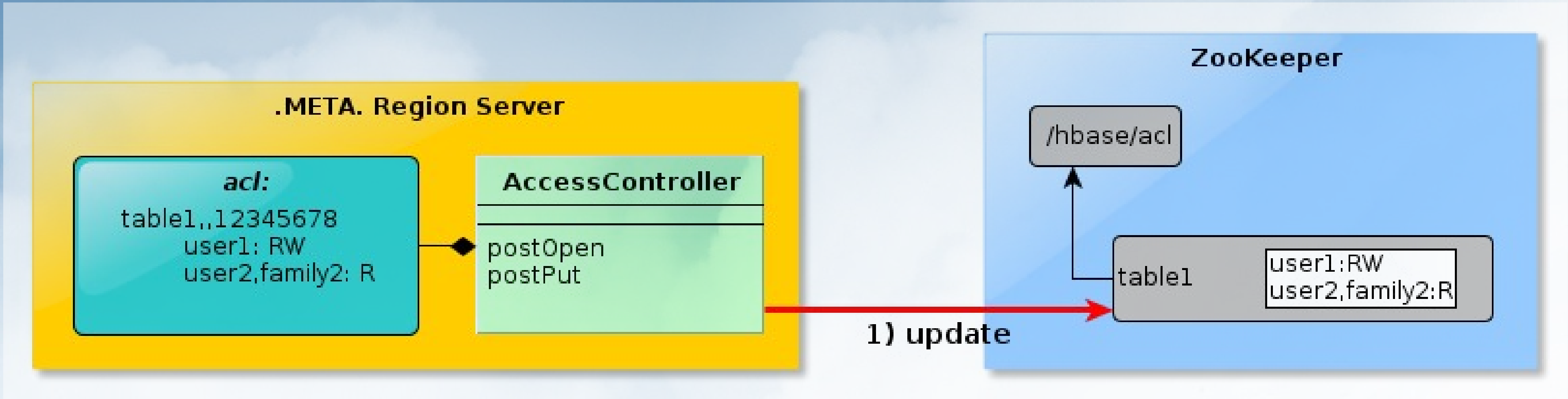
TREND MICRO™

Synchronizing ACLs

1)<u>.META. To ZooKeeper</u>

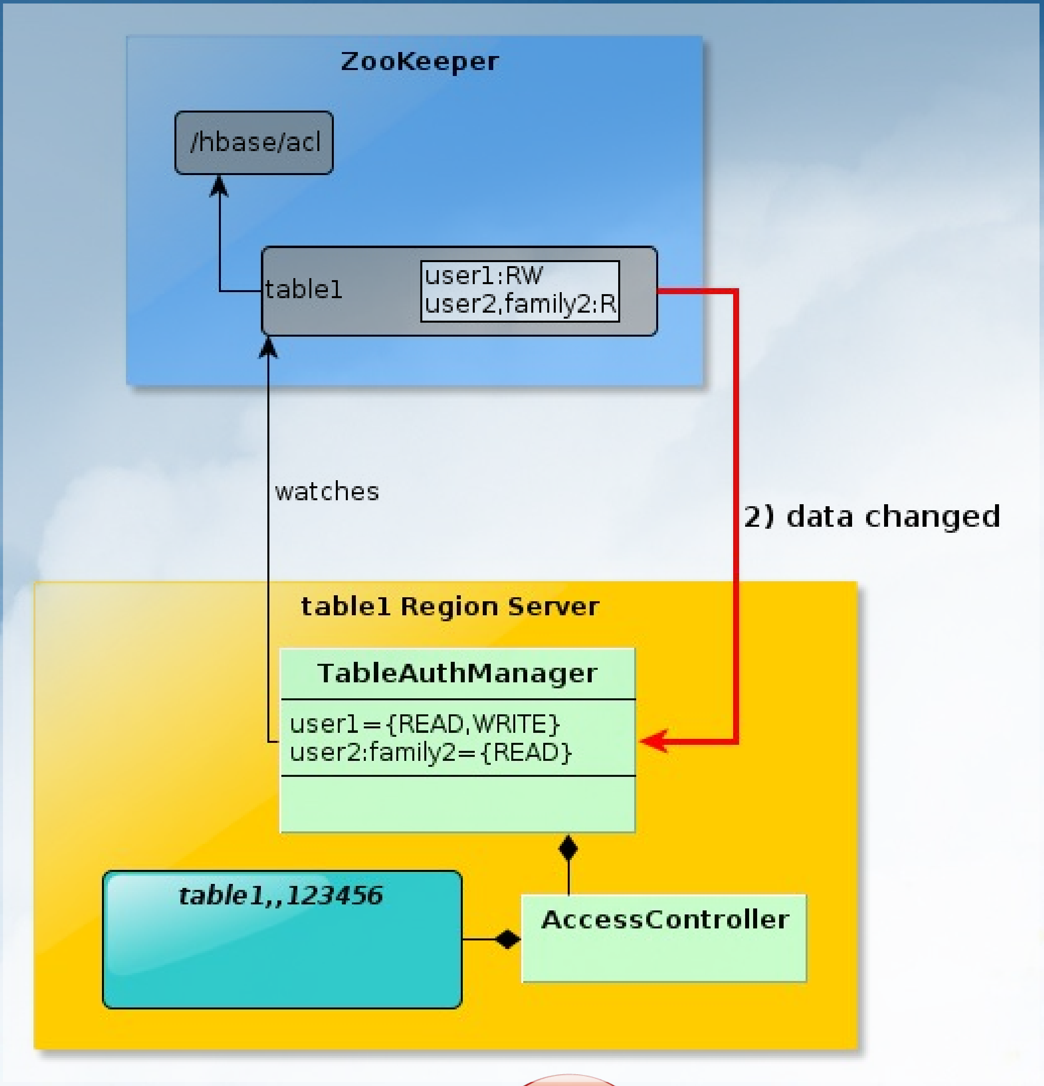
- On initial load populates ZK znode per table
- Grant updates to .META. also update ZK

2)ZooKeeper to RegionServers



Synchronizing ACLs

1).META. To ZooKeeper
2)<u>ZooKeeper to RSs</u>
- Znode watcher notified of data change
- New permissions loaded into local cache



Next Steps

- Handling of Master Operations
  - Create and Admin permissions
- Delegation token authentication for MapReduce
- Additional permissions
  - Execute, Create, Admin
- Roles
  - Current implementation is really ACL not RBAC
- ZooKeeper Kerberos auth plugin
- Audit logging
- More granular access control
  - Per row or per KV?
  - Would be implemented via meta-columns

For More Information

- HBase blog: Secure HBase by Eugene Koontz
- HBase JIRA
  - HBASE-1697: Discrentionary access control (umbrella issue)
  - HBASE-3025: Coprocessor based access control
- Code
  - http://github.com/trendmicro/hbase/tree/security