HBase for real-time data Session 2

Ronan Stokes(rstokes@cloudera.com)
Solutions Architect
Cloudera Inc.

HBase application design

HBase applications – great fit …

- Great fit indicators …
  - Need near real-time latency for data acquisition or response times
  - Need capability for large volume random write, random read, or both
  - Need to do many thousands of operations per second on large volumes of data
  - Can optimize for access patterns
  - Use of fuzzy, unstructured or time series data
  - Just in time index / corpus preparation
    - Preparation for ingest to other system such as Lucene/Solr

Copyright 2014 Cloudera Inc. All rights reserved

HBase applications – consider tradeoffs…

- Consider trade offs if application meets none of previous criteria but …
    - Structure is evolving or undefined
    - NOSQL model fits well for application
        - Operations such as large volume multiple cross joins / reordering on large data sets are expensive computationally in Hive / Pig
    - You're already using Hadoop and need key-value store
    - Disjoint processes making frequent updates to same set of data
        - For example digital goods asset ingest vs. metadata
    - Data volume is small
    - Data access pattern is largely read entire dataset sequentially and process it

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Application considerations

- What are key SLAs?
  - Low latency response to data requests
  - Near real time ingest of data
  - Ability to calculate metrics quickly
  - Ability to isolate changes
- Consider also
  - Storage requirements
  - Integration requirements
  - Ingestion style – live, batch, or both
  - Analytics SLAs

Copyright 2014 Cloudera Inc. All rights reserved

Application considerations-2

- Consider applications
  - Portal – low latency access, batch or real time ingest
  - Match of ads or coupons to pages – near real-time response, batch ingest
  - Time series analytics – real time data but not necessarily near real-time access or ingest
  - Production monitoring – low latency access but data may be batched or live (consider monitoring mix / max production levels in hour)
  - Open TSDB – production monitoring of compute processes

Copyright 2014 Cloudera Inc. All rights reserved

Key design decision points - 1

- General requirements
- Performance / SLAs
  - Performance & SLAs
- Data volume, handling and growth
  - Types of sources and transformations required
  - How much will be stored in HBase ? Will it be stored in HDFS as raw data also ?
  - How will it grow
  - Will it need to be aged out
  - Does compression make sense ?

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Key design decision points -2

- Data schema layout
  - Choice of row keys
  - Choice of column families
  - Access patterns
  - Column design
  - Use of versions, TTL, counters
  - Use of pre-splits for tables
  - Wide versus narrow rows
  - Use of versions, TTL, counters
  - Use of special features – bloom filters etc.

Copyright 2014 Cloudera Inc. All rights reserved

Key design decision points - 3

- Data ingestion / acquisition mechanisms
  - Flume, Sqoop, MR, Hive, Pig, other …
  - Transforms to be performed during ingest
  - Automating ingest
  - Isolating updates for incremental processing
- Orchestration of tasks – Oozie or other
- Data access
- Analytics, secondary indexes etc.

Copyright 2014 Cloudera Inc. All rights reserved

Key Design - 4

- Query access patterns
  - Filtering
  - Need for secondary indexes
  - Pre-computing results
  - Background processes
- Data integration techniques
- Special considerations
  - Use of hybrid model – HDFS and HBase
- Data access will determine key design

Copyright 2014 Cloudera Inc. All rights reserved

Real-time considerations

- Responsiveness
- Handling time data
  - Time slices – wide rows vs. narrow rows
  - Multiplexing observations onto broad time slots
  - Increasing time stamp keys
  - Transpose to time based indexing
  - Computing intervals
  - Using HBase's version (timestamp) information
    - Can quickly get all recent updates

Copyright 2014 Cloudera Inc. All rights reserved

Remember

- All rows in table do not have to be same format
- Can add summary rows into live data
- Computing intervals
  - In Hive – requires use of cross joins (costly in compute time)
  - In MR –
  - Iteratively
- Wide rows take same space as narrow rows
  - Each column stored as key, family, qualifier, timestamp, value
  - But have index overhead for narrow rows
  - Against more incremental operations with wide rows to iterate columns

Copyright 2014 Cloudera Inc. All rights reserved

Storage considerations

- Compression
- Regions
- Splitting
- Compaction
  - Usually want to disable automatic major compaction
  - But control it via a cron job or similar

Copyright 2014 Cloudera Inc. All rights reserved

Data modelling for realtime data

Key design

- Goals:
  - Facilitate fast retrieval of data
  - Avoid region server "hotspotting"
- Rows are distributed across regions according to key values
- If all keys fall into 1 region, all i/o will be to one server
  - Strict increasing keys such as successive timestamps will fall into this pattern
  - Time series data is very prone to this
- Avoid hotspotting by randomly distributing keys

Copyright 2014 Cloudera Inc. All rights reserved

Presplitting tables

- By default splits occur at mid point of key range
- Can compute pre-splits to force splits at other points
  - Can define split points for keys in table at creation time
  - HBase shell:
  - `create ’table', {NAME => 'f'},{SPLITS => ['g', 'm', 'r', 'w']}`
- Can use to avoid issues with limited range of row keys
  - Text keys with ‘a’ – ‘z’ => byte codes 97 – 122
  - Could normally fall into same region
  - But use of pre-splits will force them into separate regions

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Key design – alleviating hotspots

- Prefix or 'salt' keys with evenly distributed value
- Prefix keys with natural field value
  - Prefix timestamp with natural id – use id of source site / device
  - Known as field value promotion
  - For string ids, will have limited set of bytes ('a' is ASCII 94)
  - => use md5 hash of device id
  - => alternatively or in conjunction with, explicitly choose splits
- Prefix keys with shard number
  - Use timestamp or random number % number of region servers
  - Will need <number of region servers> x searches to find rows
  - Makes searches more cumbersome but facilitates multithreaded searches

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Key design – use of reverse timestamps

- Often want to find newest time slice first
  - Use reverse timestamp to ensure newest row occurs first in order
  - Use Long.MaxValue – timestamp
  - By itself will not avoid hotspotting – results in strictly decreasing keys instead of increasing
- Use with prefix to distribute across regions
  - E.g. <device_id> +'.'+<reverse timestamp>
- Combine prefix, reverse time stamp and presplitting of tables to optimize

Copyright 2014 Cloudera Inc. All rights reserved

Key design

- Generally use composite keys to search by multiple data items
  - Eg device_id.reverse_timestamp
- Ordering of fields dictates search capability
  - For example if device_id is first, harder to search across all devices for particular timeslice
  - But you can have multiple row layouts in same table – use column families if not needed for same queries
    - => Can use multiple key orders to allow indexing on multiple items
    - => application can perform multiple inserts
    - => or compute as separate map/reduce job or with coprocessor

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Row layout for time data

- Consider growth in rows:
  - 1 row per ms means 86400 * 1000 rows per day
  - Results in billions of rows per year
- Alternatives
  - Use row to represent time slice – 1 minute, 5 minutes, 1 hour etc.
  - Will use same storage but much less index space

Copyright 2014 Cloudera Inc. All rights reserved
cloudera® Ask Bigger Questions

Secondary indexes

- May want secondary index to perform lookups
  - Create second table / column family
  - Populate with key and reference information
- Use other rows /cf /table to build up index of raw data
  - Compute using scheduled mr job or coprocessor
  - Alternatively ingestion process can perform multiple inserts

Copyright 2014 Cloudera Inc. All rights reserved

HBase and schemas …

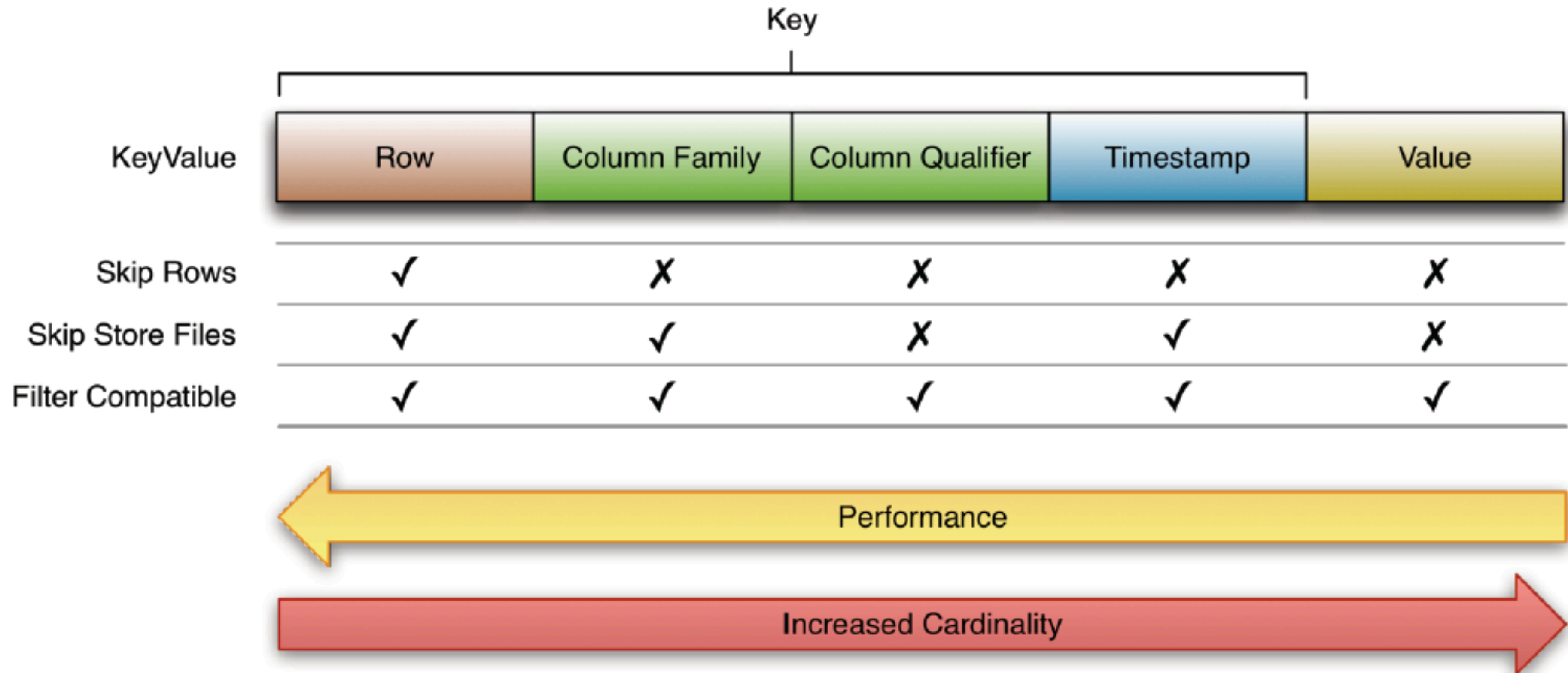
- Unlike relational databases …
  - Typically there are only a few, large (denormalized) tables
  - Each table will have a small number of Column Families
    - Within each CF you may have hundreds or thousands of columns

- Think about your access patterns . . .
  - Columns that are accessed together should be assigned to the same Column Family
  - Row Keys determine how closely on disk commonly access rows are stored
    - Recall that data is assigned to Regions according to Row Keys

- **Design for locality**

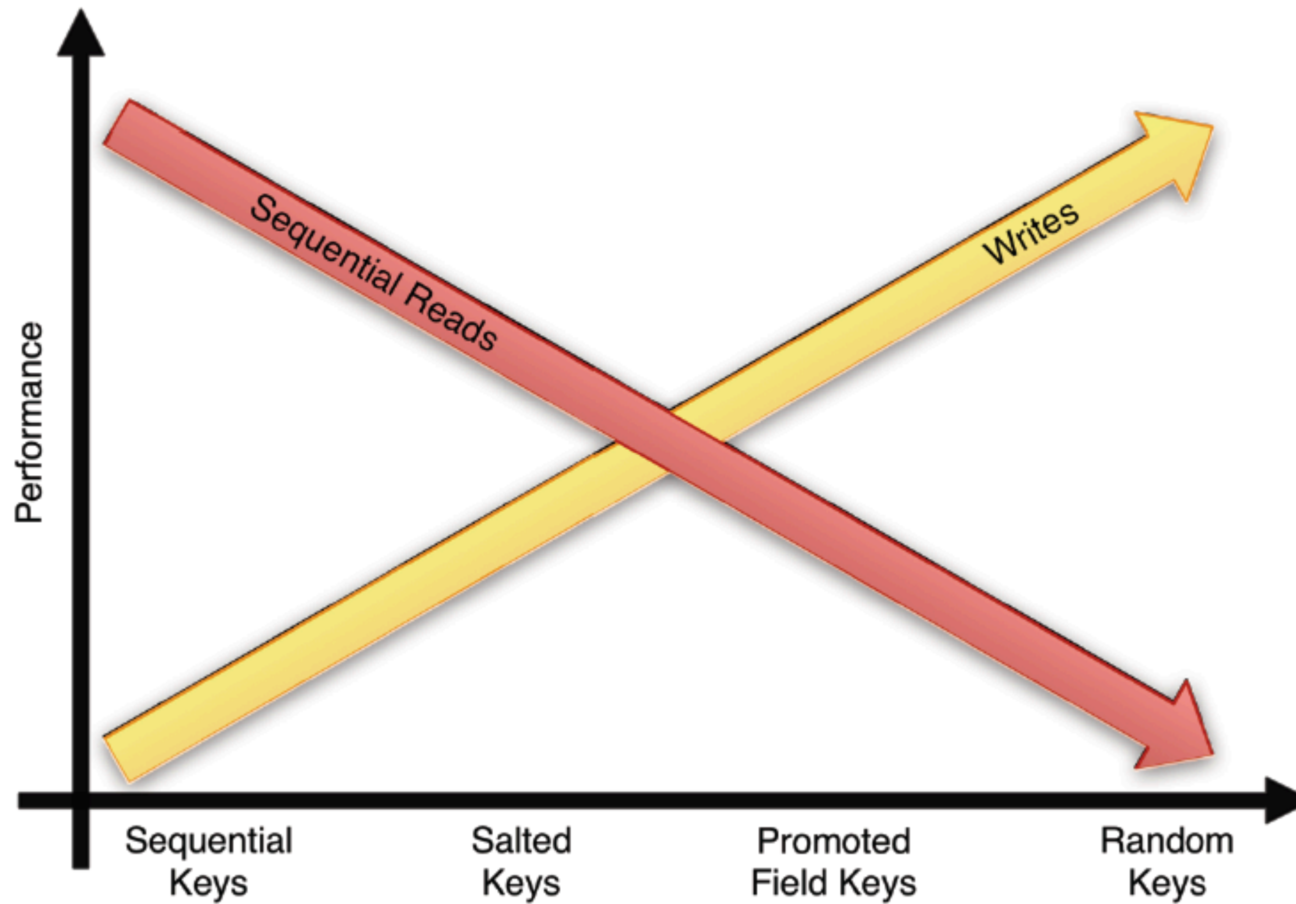
Copyright 2014 Cloudera Inc. All rights reserved

HBase addressing modes



Copyright 2014 Cloudera Inc. All rights reserved

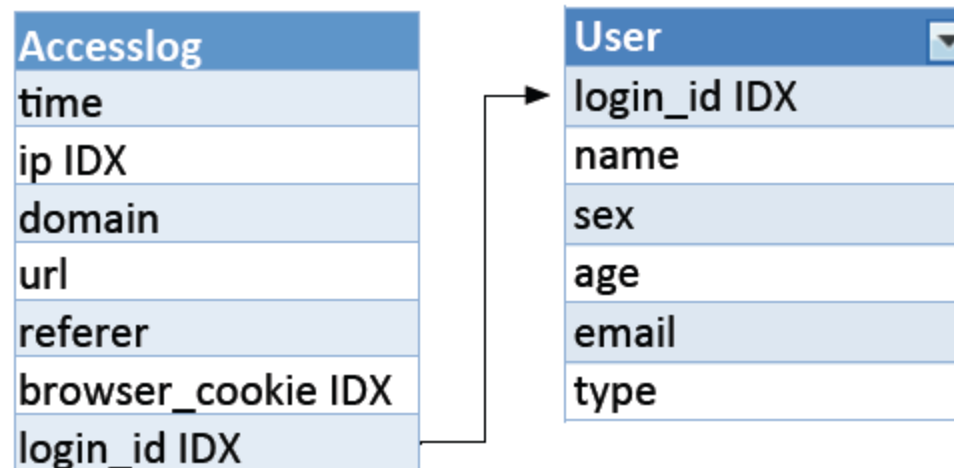
HBase performance characteristics



Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

An example – weblogs …

- User access log
  - Stores web clicks, session info, etc.
- Separate table for user metainfo
  - Stores user name, age, email, etc.
- Lookups by user_id
  - Indexes slow down inserts



Copyright 2014 Cloudera Inc. All rights reserved
cloudera Ask Bigger Questions

An example – weblogs : 2

- User info not frequently retrieved with access log data
  - If data retrieved together then use a single Column Family

	Column Families	
Row	http:column_name	user:column_name
<login_id>	http:ip	User:browser_cookie
	http:domain	user:name
	http:url	user:sex
	http:referer	user:age
	http:time	user:email
		user:type

Copyright 2014 Cloudera Inc. All rights reserved

Weblogs 3 …

- Challenge:
  - What if user_id's are assigned sequentially?
  - Creates Region "hot spots"
- Solution:
  - Prefix the row key with a sharding qualifier

	Column Families	
Row	http:column_name	user:column_name
<type><login_id>	http:ip	user: browser_cookie
	http:domain	user:name
	http:url	user:sex
	http:referer	user:age
	http:time	user:email

Copyright 2014 Cloudera Inc. All rights reserved

Weblogs 4 …

- Challenge:
  - What if we want to retrieve most recent data for a given user_id
- Solution:
  - Add a reverse timestamp
  - Data will be ordered by most recent insert

	Column Families	
Row	http:column_name	user:column_name
<type><login_id> <Long.MAX_VALUE-System.currentTimeMillis()>	http:ip	user: browser_cookie
	http:domain	user:name
	http:url	user:sex
	http:referer	user:age
	http:time	user:email

Copyright 2014 Cloudera Inc. All rights reserved

Ingesting data into HBase

Ingestion

- Typical data sources
  - Tailing logs
  - Streamed live data – sensors, market quotes , …
  - Incremental Batches – hourly updates, aggregates from other systems
  - Initial load of batch data
  - Imports from databases
- Delivery mechanisms
  - File system, cloud storage, HTTP, Thrift, JMS etc.
- Watch for limits of transmission mechanism
  - May need to batch due to transmission limits – e.g. weather station at remote location transmitting over 3g

Copyright 2014 Cloudera Inc. All rights reserved

Ingestion - approaches

- Ingest to HDFS and upload to HBase
  - Can use Hive, Pig, MR for ETL & index calculation
  - HDFS serves as backup
  - HDFS faster for SQL style queries with Impala, Hive
- Use HBase for period x and use HDFS as long term archive
  - Use partitions for optimal retrieval and storage
  - Use compression
- Stream direct to HBase
- Combination of above

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Ingestion mechanisms

- HDFS file operations
- Hive
- Sqoop
- Pig
- Flume
- Java MR
- Thrift + other languages
- Bulk import

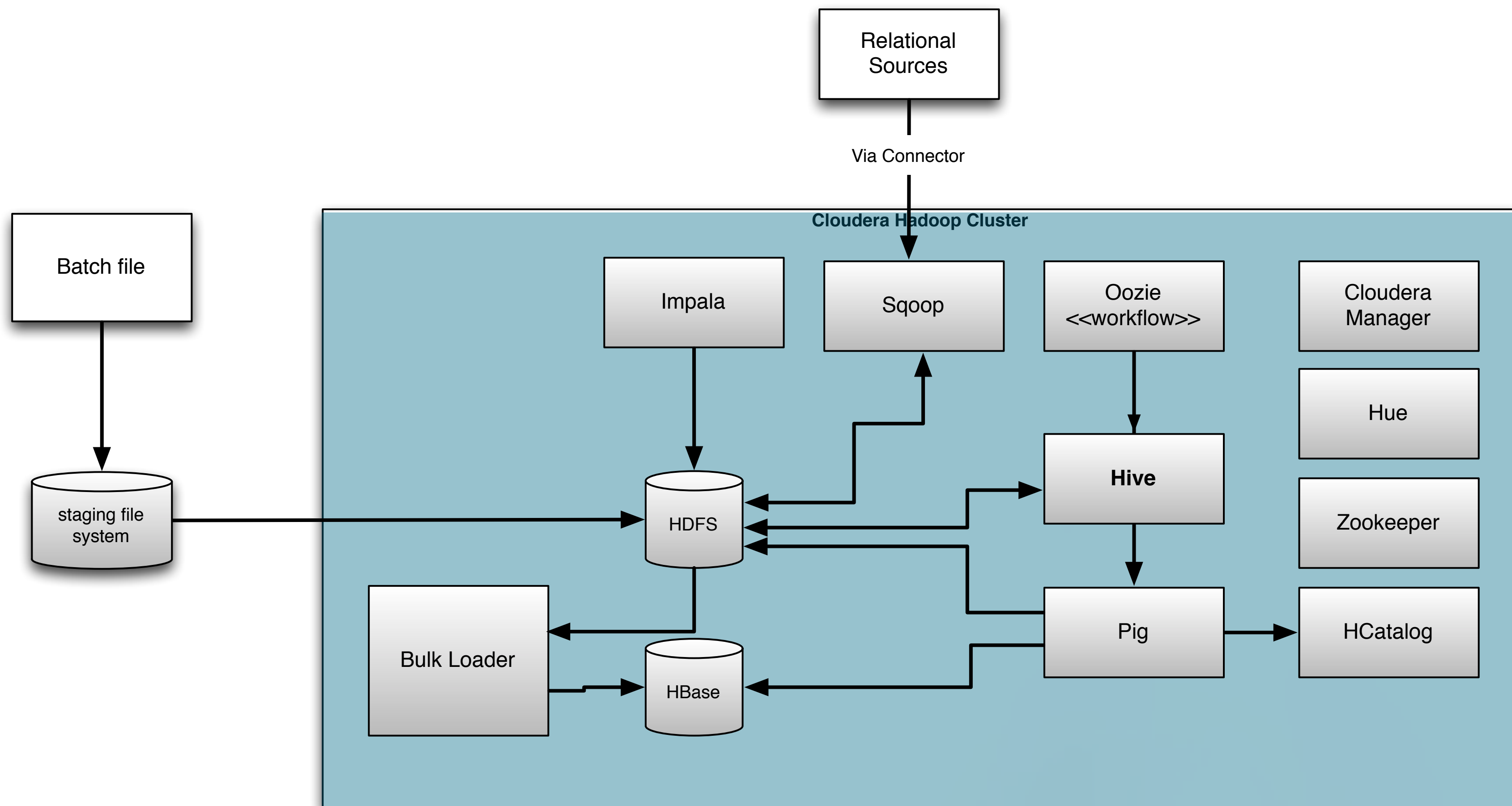
Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Ingestion mechanisms – file operations

- Puts from file system
  - Option 1 – download from remote system & perform put
  - Option 2 – mount remote system as nfs mount and perform put from nfs directly
  - Remote mount avoids extra copy
- Considerations
  - Usually stage in HDFS before upload to HBase
  - Use distcp for very large copy
- Compression
  - Consider use of partitioning and compression for long term archive storage
  - Can compress before upload for improve storage and throughput

Copyright 2014 Cloudera Inc. All rights reserved

Batch data ingestion



Copyright 2014 Cloudera Inc. All rights reserved

Compression considerations

- Compression in HDFS
  - Reduces size of data – compression factors 0.4 – 0.7
  - Choice of codes – bz2, gzip, snappy etc
  - Some are not splittable – MR does not use parallelism
- Recommendations
  - Use splittable compression – bz2, snappy, LZO
  - Time series data compresses very highly – especially in columnar file formats

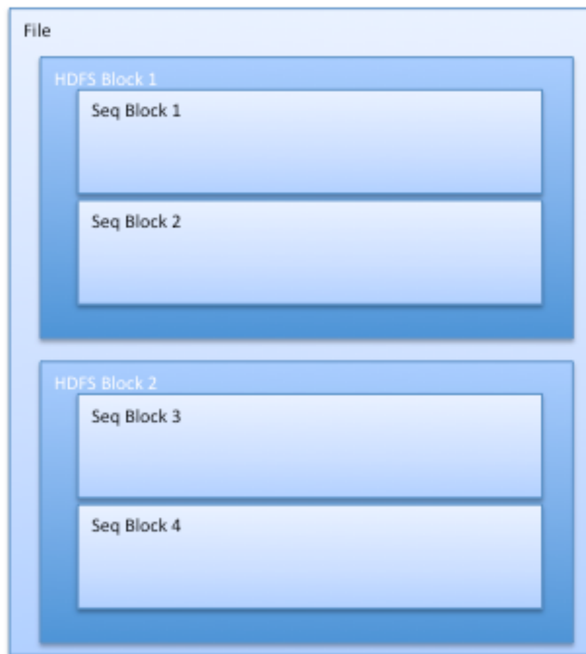
Copyright 2014 Cloudera Inc. All rights reserved

Compression considerations

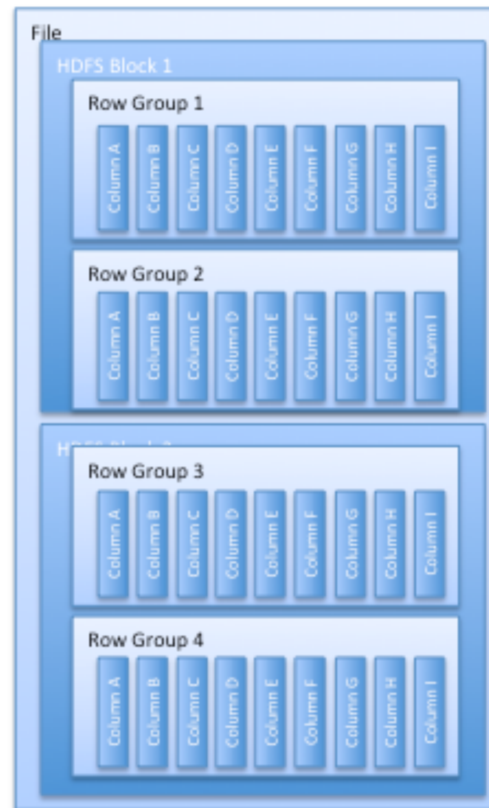
- Compression in HDFS
  - Reduces size of data – compression factors 0.4 – 0.7
  - Choice of codes – bz2, gzip, snappy etc
  - Some are not splittable – MR does not use parallelism
- Recommendations
  - Use splittable compression – bz2, snappy, LZO
  - Time series data compresses very highly – especially in columnar file formats

Copyright 2014 Cloudera Inc. All rights reserved

File formats and compression



Sequence file



RC file

- On ingest only some file formats splittable – bz2, lzo, snappy
- When converted to block oriented file format, can be compressed on block basis – each split is accessible independent of compression
- Use block compression for long term archive purposes

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Ingestion - Sqoop

- Import from database
  - Direct to impala/hive table
  - To HDFS files
  - To HBase directly
- Key advantage – can invoke multiple importers in parallel
- Example:
  - **$SQOOP_HOME/bin/sqoop import --hbase-create-table --hbase-table signaldata --column-family info --hbase-row-key sensorid --connect jdbc:mysql://host-ip/testhadoop --table sensor_batch_data -m 1**

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Ingestion – Hive / Pig /MR

- Use Hive / Pig / MR to join and merge data before upload to HBase
  - Computing row keys
  - Transpose of data to time slices
  - Computing of intervals
  - Preparing for long term storage
- Great for batch and incremental batch upload and processing
- Can optimize loading by disabling write to WAL for put
  - Loose recovery but if file is already in HDFS …
  - Bulk import disables write to WAL for speed

Copyright 2014 Cloudera Inc. All rights reserved
cloudera® Ask Bigger Questions

Ingestion – Java and other languages

- Use Java, Python etc. to write data directly to HBase
- Can run as MR job or purely as client
- For Non Java languages will need to use Thrift or Rest gate
  - Generate using
  - Thrift –gen py <path to thrift file>
- Many optimizations and fine control available to Java
  - If using non java, then use java for custom extensions

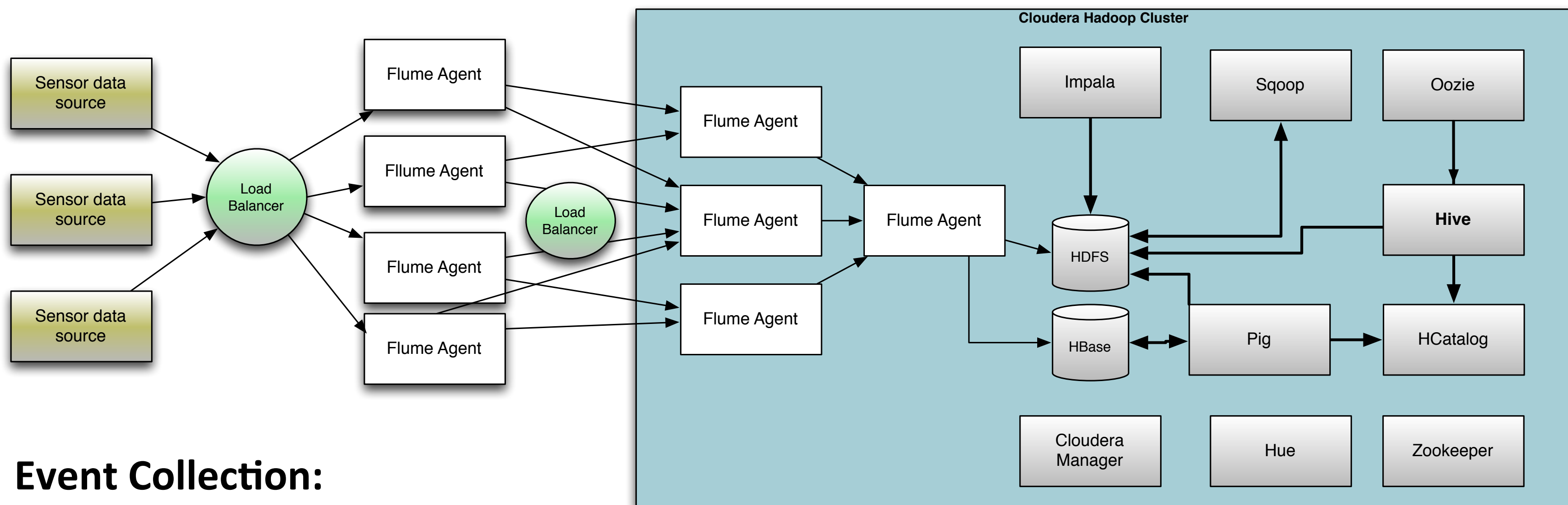
Copyright 2014 Cloudera Inc. All rights reserved

Ingestion – Flume

- Capture live data from streaming source
- Upload to HBase directly
- Lower overall throughput
  - But data is available quicker
- Flume provides collection and aggregation of streaming Event Data
  - Data is considered to be Events
  - Canonical use case is collecting log data

Copyright 2014 Cloudera Inc. All rights reserved
cloudera®
Ask Bigger Questions

Flume based sensor data ingestion



Event Collection:

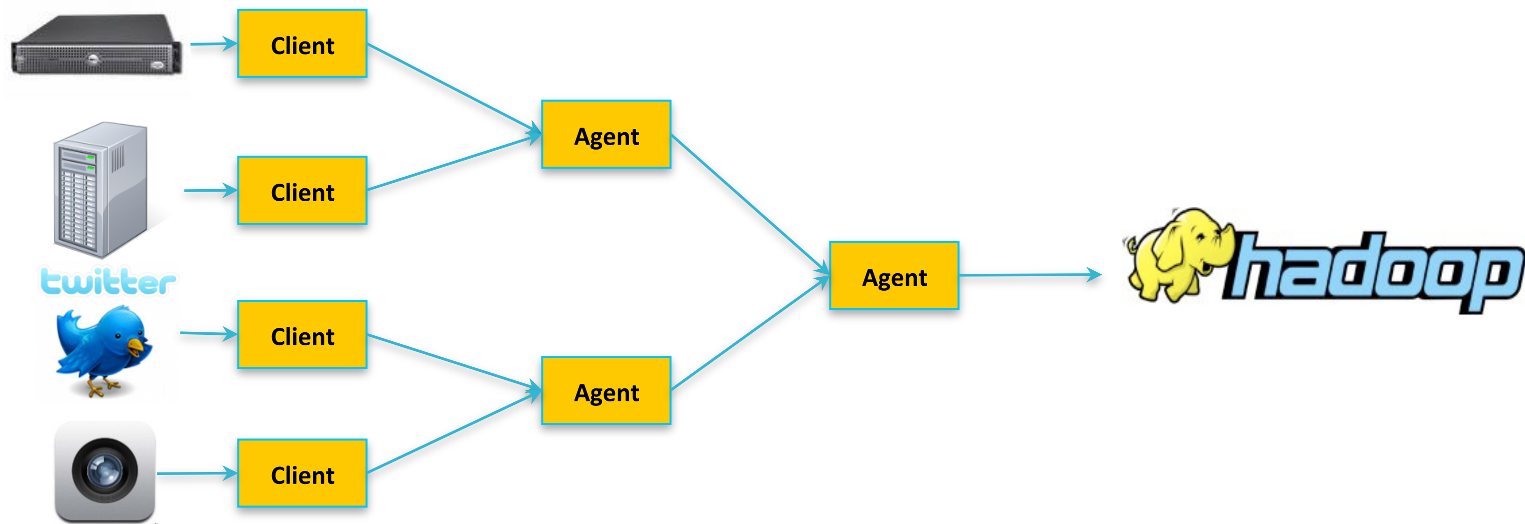
Sources: Avro, netcat, exec

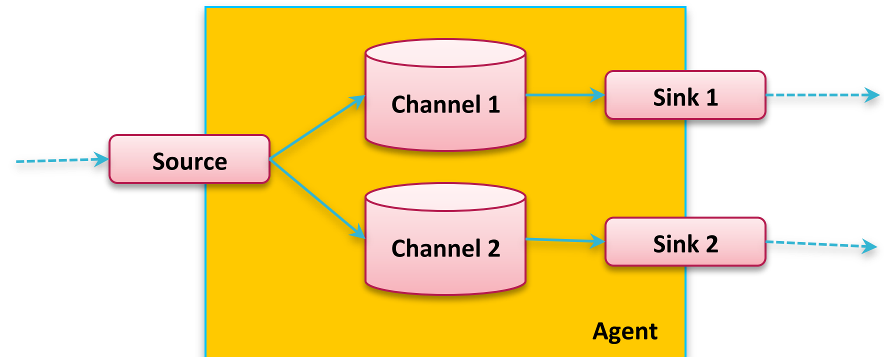
Channels: memory, JDBC

Sink: HDFS, Avro

Copyright 2014 Cloudera Inc. All rights reserved
cloudera Ask Bigger Questions

FlumeNG: High-level Architecture





Event Collection:

Sources: Avro, netcat, exec

Channels: memory, JDBC

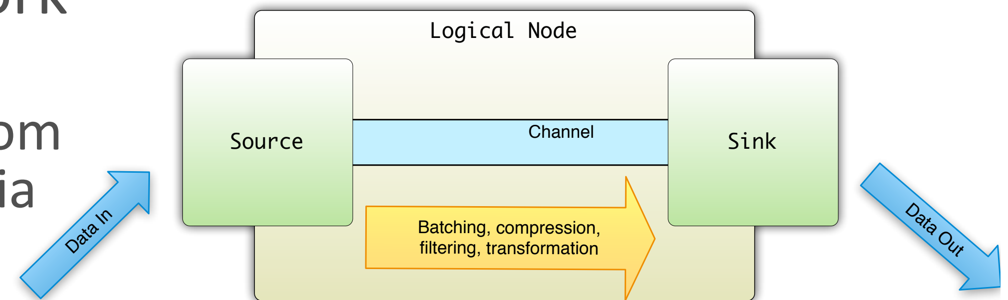
Sink: HDFS, Avro

Copyright 2014 Cloudera Inc. All rights reserved

cloudera
Ask Bigger Questions

Flume architecture

- Data passes through network of logical nodes
    - One function – read data from source and write it to sink via channel
    - Source can be file, process output or other flume logical node
    - Sink can be file, hdfs, S3, IRC, email, Flume logical node



- Chain of logical nodes is known as flow
    - First node in chain is *agent*
    - Last node in chain is *collector*

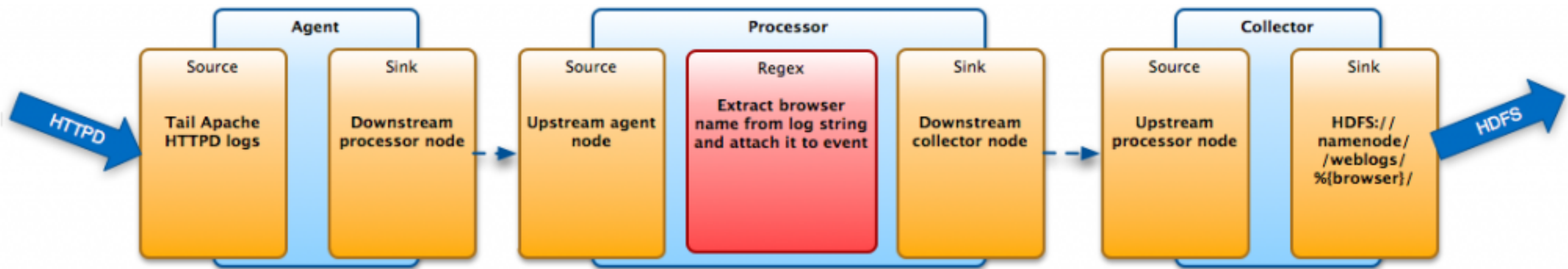
Copyright 2014 Cloudera Inc. All rights reserved

Flume architecture - 2

- Configuration controlled by Flume master distributed process
  - Single flume process houses many logical nodes
  - Logical nodes can be created and configured without restarting flume process
  - Flume master distributes changes and co-ordinates monitoring

Copyright 2014 Cloudera Inc. All rights reserved

Example Web Server Log Flow



- Agent monitors tail of web log
  - Each line of file captured as event
- Processor uses regex to extract data from event
- Collector writes events to HDFS
  - Could be HBase or other

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Basic Concepts - Agents

- Agents are logical node in logical node chain
- Agents have:
  - Source
  - Channel
  - Sink
- Valid Configuration
  - Must have at least one Channel
  - Must have at least one Source or Sink
  - Any number of Sources
  - Any number of Channels
  - Any number of Sinks

Copyright 2014 Cloudera Inc. All rights reserved

Agent Components - Sources

- Event Driven
- Supports Batch Processing
- Source Types:
  - **SYSLOGTCP, SYSLOGUDP**
  - **NETCAT**
  - **EXEC**
  - **AVRO (IPC Source)**
  - **SEQ (For Testing Purposes)**
- Can replicate or multiplex events to multiple channels

Copyright 2014 Cloudera Inc. All rights reserved

Agent Components - Channels

- Passive Component
- Determines the reliability of a flow
- Specialized Channels
  - **FILE**
  - **MEMORY**
  - **JDBC**

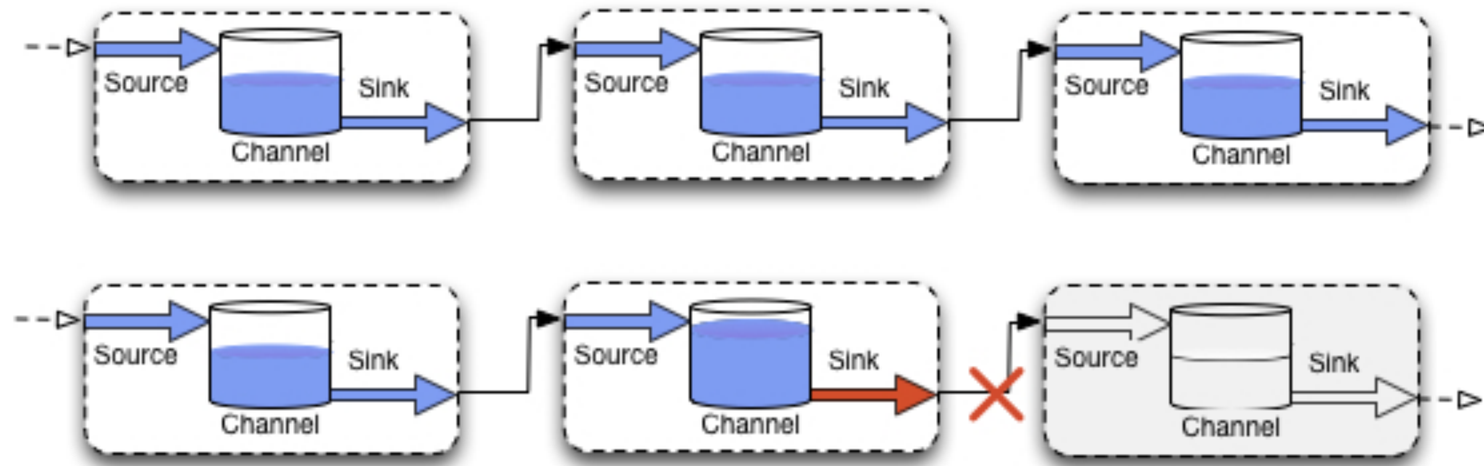
Copyright 2014 Cloudera Inc. All rights reserved
cloudera®
Ask Bigger Questions

Agent Components - Sinks

- Supports Batch Processing on size or time
- Specialized Sinks
  - **HDFS**
  - **HBASE, ASYNC-HBASE**
  - **FILE_ROLL**
  - **AVRO** (IPC Sink)
- Integration of MorphLines
- Sink of one flow be source of events for other flows

Copyright 2014 Cloudera Inc. All rights reserved
cloudera® Ask Bigger Questions

Concepts in Action



- Source: Puts events into the Channel
- Sink: Drains events from the Channel
- Channel: Store the events until drained

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

More Agent Component Concepts

- Client
- Source
- Channel Processor
  - Interceptor
  - Channel Selector
- Channel
- Sink Processor
- Sink
- Terminal Sink

Copyright 2014 Cloudera Inc. All rights reserved

More Agent Component Concepts

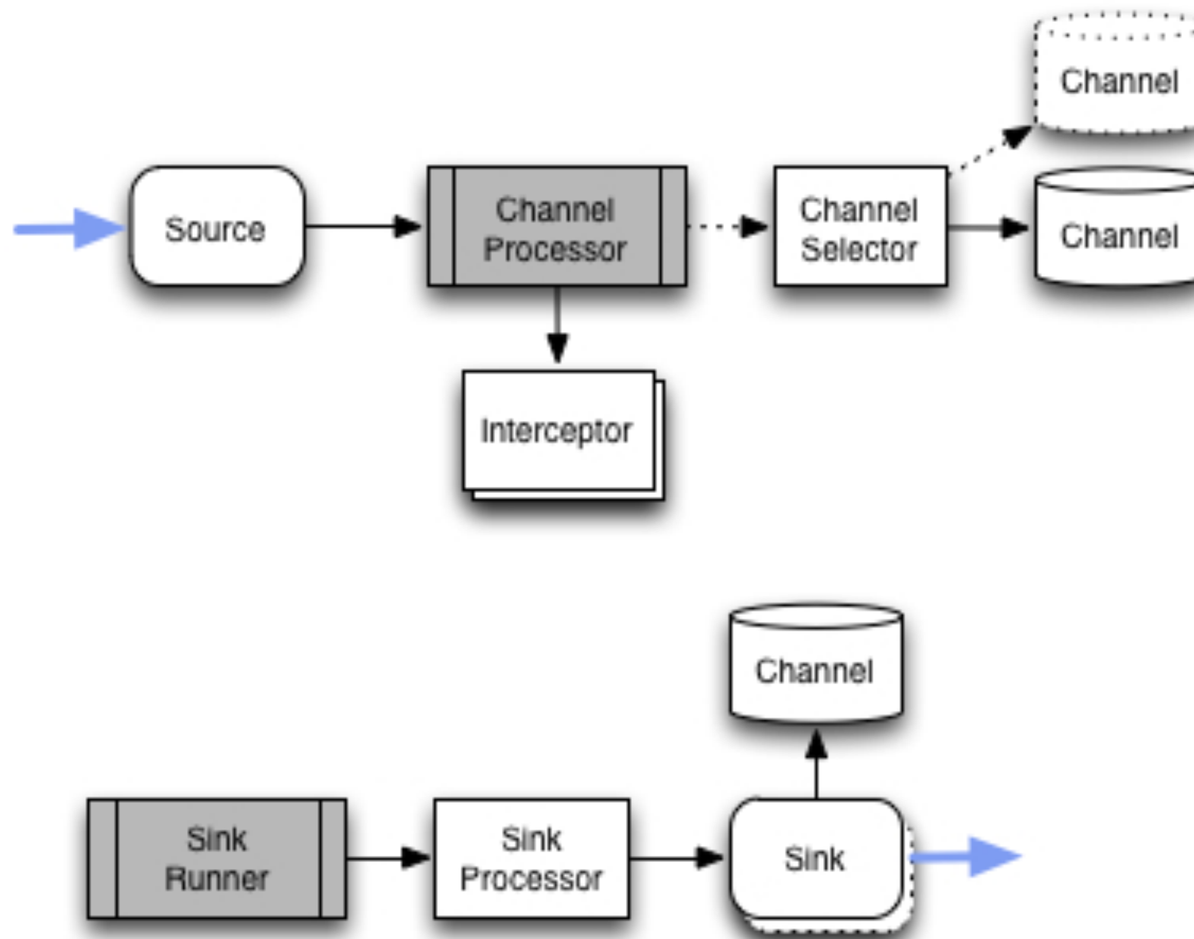
- Client
- Source
- Channel Processor
  - Interceptor
  - Channel Selector
- Channel
- Sink Processor
- Sink
- Terminal Sink

Event manipulation

Event routing

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

More Concepts in Action



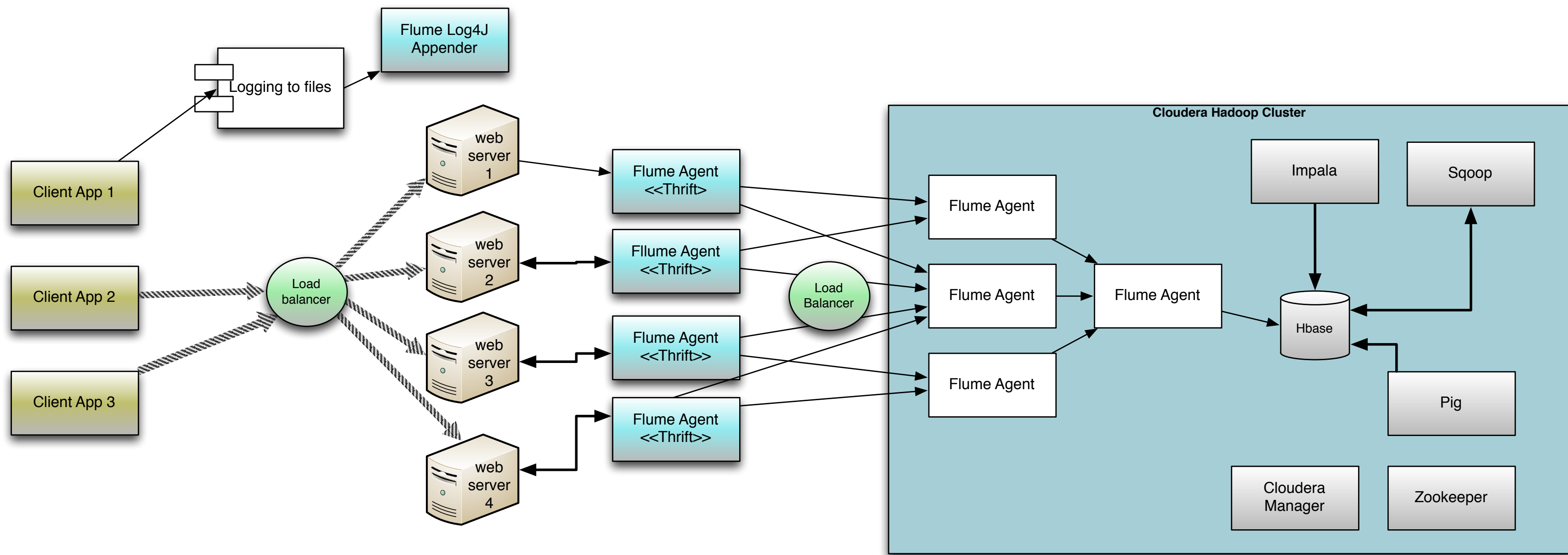
Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Interceptor (Event Manipulation)

- Applied to Source
- One source can have many interceptors
- Chain-of-responsibility
- Can be used for tagging, filtering
- Built-in interceptors:
  - **TIMESTAMP**
  - **HOST**
  - **STATIC**

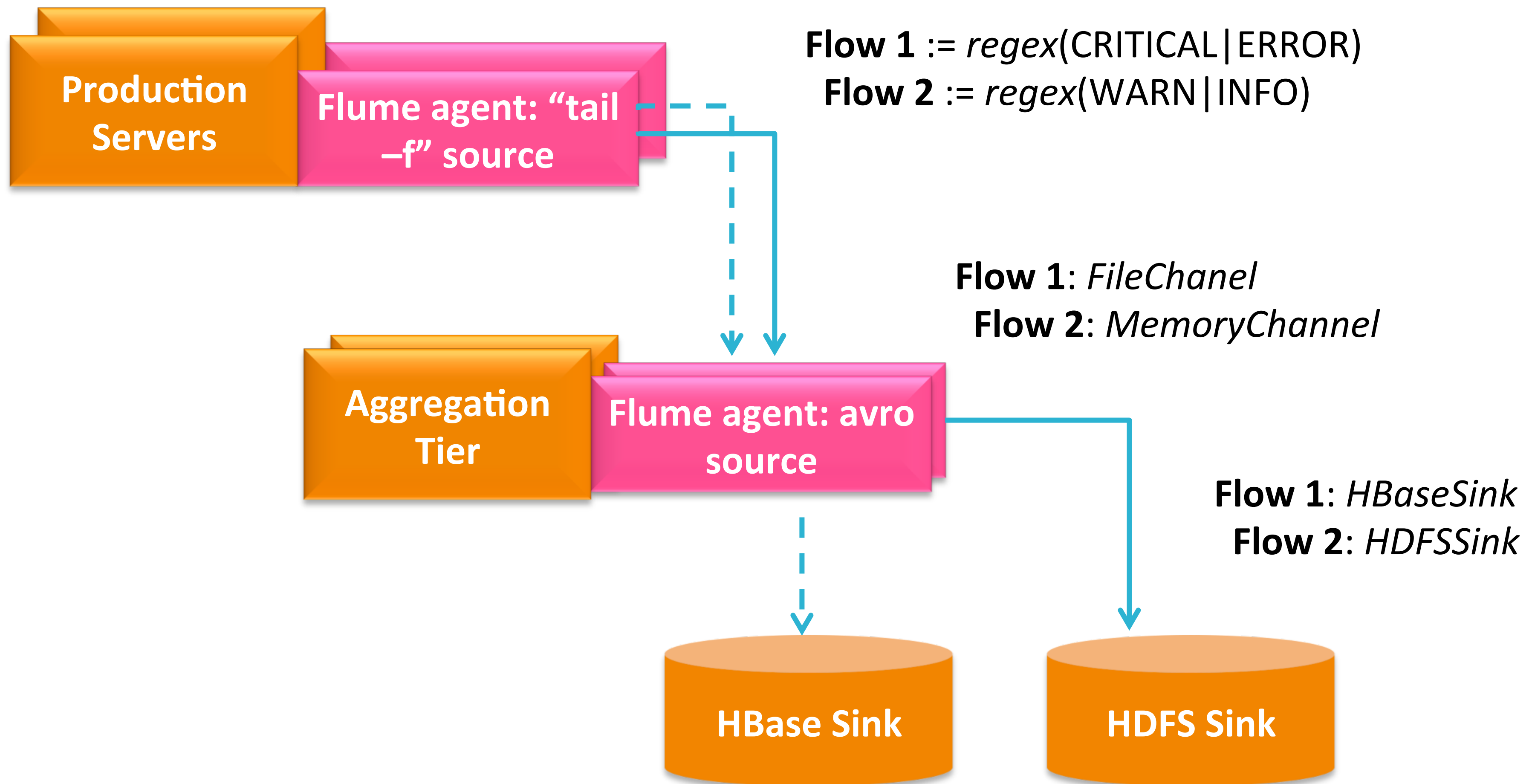
Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Flume based log ingestion



Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Example Deployment: Server Logs



Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

Flume design considerations

- Capacity planning
  - Disk channel can handle 10s or MB/s
  - Memory channel can handle 100s of MB/s
- Starting point
  - Look at throughput for each tier
    - Bytes per second and events per second
    - Start by getting performance for sample and synthetic data loads

Copyright 2014 Cloudera Inc. All rights reserved
cloudera
Ask Bigger Questions

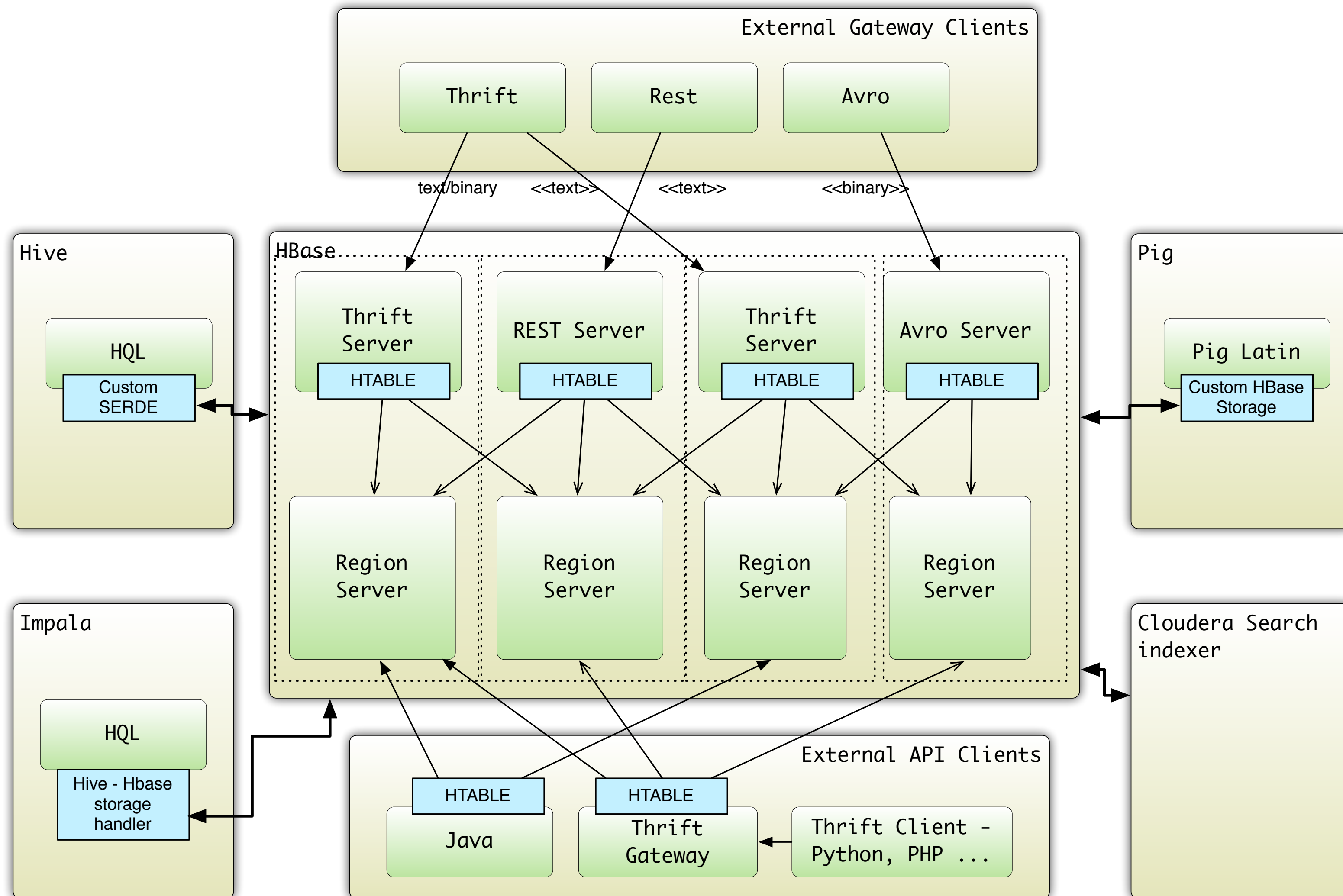
Accessing and manipulating HBase data

Access to HBase data

- Via Impala / Pig / Hive
- Via integration with search
- Via client applications
  - Java
  - Other languages via gateways

Copyright 2014 Cloudera Inc. All rights reserved

Integration options



Copyright 2014 Cloudera Inc. All rights reserved
cloudera Ask Bigger Questions

Advanced Concepts – Bulk Loading

Bullk loading in HBase

- Very efficient load mechanism in HBase
- Cost of puts in HBase due to
  - Write to WAL
  - Write to Memstore
  - Write to HFILE
- Bulk loading takes HFILE and uploads to memstore
  - Bypassing write to WAL
  - If region server fails, source data is available in source file but no WAL recovery
- Bulk loading is preparing HFiles and loading them directly

Copyright 2014 Cloudera Inc. All rights reserved

Bulk loading

- Preparing HFILEs
  - Use built in tools
    - ImportTSV
  - Or, write Map Reduce job
- Load the bulk load files using complete bulk load tool

```
Hadoop jar hbase-VERSION.jar
completebulkload
[-c path to config]
/user/todd/myoutput mytable
```


Copyright 2014 Cloudera Inc. All rights reserved
cloudera®
Ask Bigger Questions

Preparing HFiles with ImportTSV

- Preparing files
  - Load csv or tsv into HFiles
  - Specify columns
  - Tab separated by default
  - Need to specify one of TSV fields as row key
- Example:

```
bin/hbase org.apache.Hadoop.hbase.mapreduce.ImportTsv
  -Dimporttsv.columns=a,b,c
  -Dimporttsv.bulk.output=hdfs://storefile-outputdir
<tablename> <hdfs-data-inputdir>
```

- Options
  - -Dimporttsv.bulk.output=/path/for/output
  - -Dimporttsv.skip.bad.lines=false
    - fail if encountering an invalid line
  - '-Dimporttsv.separator=|’
  - -Dimporttsv.timestamp=currentTimeAsLong
  - -Dimporttsv.mapper.class=my.Mapper

Copyright 2014 Cloudera Inc. All rights reserved

Prepare with Map Reduce job

- Generate HFile with HFileOutputFormat
- Most efficient way
  - Generate HFiles that fit in regions
  - Need to compute splits via TotalOrderPartitioner to split data correctly
- But can use larger HFILE and loader will split anyway
  - Doesn't allow for fine control of split points unless using computed splits
  - But can define split points for keys in table at creation time
  - HBase shell:
  - `create 'table', {NAME => 'f'}, {SPLITS => ['g', 'm', 'r', 'w']}`

Copyright 2014 Cloudera Inc. All rights reserved

Other topics

Putting it together : Oozie

- Extensible workflow orchestration for Hadoop
- Allows scheduling of actions
  - sequentially or parallel
  - With different paths depending on results
  - Flow defined by workflow.xml
- Parallel actions execute across cluster
  - Make sure dependencies exist on each node of cluster
  - Device drivers , scripts
- Extend with Java code for custom actions

Copyright 2014 Cloudera Inc. All rights reserved

Putting it together : Oozie

- Built in actions
  - Run a shell script (can invoke pig or hbase script in turn)
  - Run a sqoop job
  - Send an email
  - Execute Hive script
  - Distributed Copy
- Can publish notifications via JMS for status
- Trigger launch of action
  - Manually
  - By time
  - By presence of resource (… batch file available in monitored folder)

Copyright 2014 Cloudera Inc. All rights reserved

Putting it together : Oozie

- Processing batch files – typical sequence
  - Assign batch id
  - Move to processing folder to avoid double processing
  - Create partition location if using
  - Put to HDFS landing directory
  - Add to hive raw data table
  - Process to hive compressed table
  - Load batch to HBase
  - If any step fails move source file to “failed” subdirectory
  - On success move file to “success directory”

Copyright 2014 Cloudera Inc. All rights reserved

End of session 2