cloudera®
Ask Bigger Questions

Deploying Apache Flume for Low-Latency Analytics

Mike Percy
Software Engineer, Cloudera
PMC Member & Committer, Apache Flume

About me

- Software engineer @ Cloudera
- Previous life @ Yahoo! in CORE
- PMC member / commiter on Apache Flume
- Part-time MSCS student @ Stanford
- I play the drums (when I can find the time!)
- Twitter: @mike_percy

What do you mean by low-latency?

- Shooting for a 2-minute SLA
  - from time of occurrence
  - to visibility via interactive SQL
- Pretty modest SLA
- The goal is fast access to all of our “big data”
  - from a single SQL interface

Approach

- Stream Avro data into HDFS using Flume
  - Lots of small files
- Convert the Avro data to Parquet in batches
  - Compact into big files
- Efficiently query both data sets using Impala
  - UNION ALL or (soon) VIEW
- Eventually delete the small files

Agenda

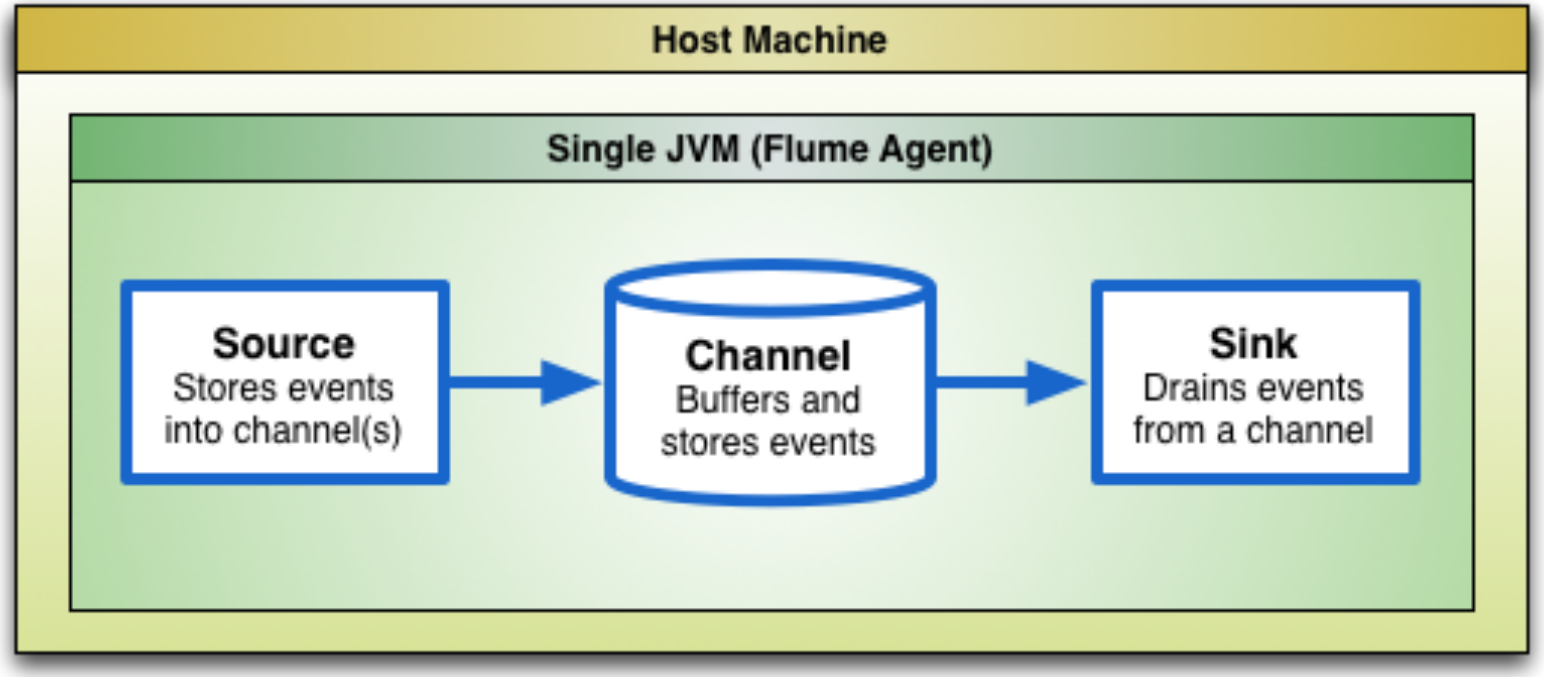
1. Using Flume
   - Streaming Avro into HDFS using Flume
2. Using Hive & Impala
   - Batch-converting older Avro data to Parquet
   - How to query the whole data set (new and old together)
3. Gotchas
4. A little about HBase

github.com/mpercy/flume-rtq-hadoop-summit-2013

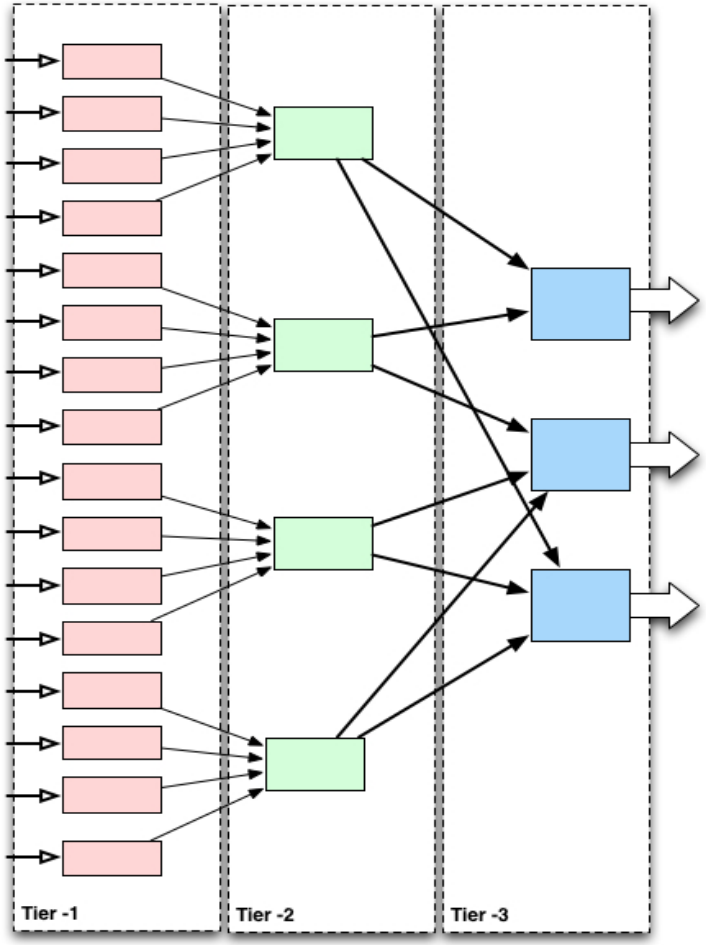
Why Flume event streaming is awesome

- Couldn't I just do this with a shell script?
  - What year is this, **2001**? There is a better way!
- **Scalable** collection, aggregation of events (i.e. logs)
- Dynamic, **contextual** event routing
- **Low latency**, **high throughput**
- **Declarative** configuration
- Productive out of the box, yet powerfully **extensible**
- **Open source software**

Flume Agent design



Multi-agent network architecture



Typical fan-in topology

Shown here:

- Many agents (1 per host) at edge tier
- 4 “collector” agents in that data center
- 3 centralized agents writing to HDFS

Events

Flume's core data movement atom: the **Event**

- An Event has a simple schema
  - **Event header:** Map<String, String>
    - similar in spirit to HTTP headers
  - **Event body:** array of bytes

Channels

- Passive Component
- Channel type determines the reliability guarantees
- Stock channel types:
  - JDBC – has performance issues
  - Memory – lower latency for small writes, but not durable
  - File – provides durability; most people use this

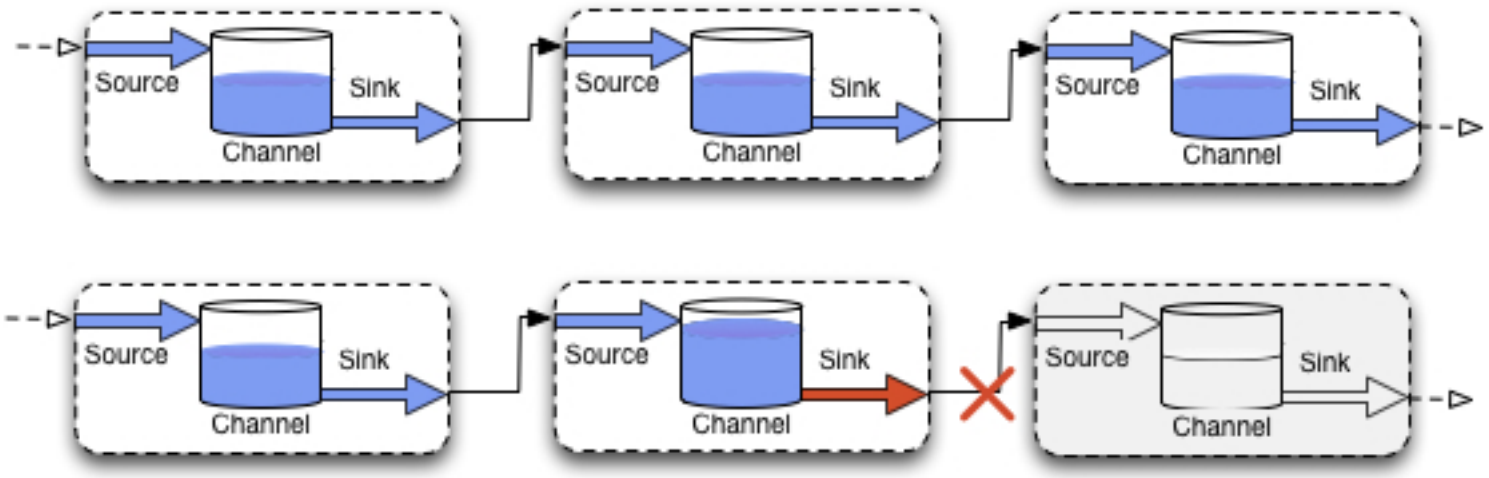
Sources

- Event-driven or polling-based
- Most sources can accept batches of events
- Stock source implementations:
  - Avro-RPC – other Java-based Flume agents can send data to this source port
  - Thrift-RPC – for interop with Thrift clients written in any language
  - HTTP – post via a REST service (extensible)
  - JMS – ingest from Java Message Service
  - Syslog-UDP, Syslog-TCP
  - Netcat
  - Scribe
  - Spooling Directory – parse and ingest **completed** log files
  - Exec – execute shell commands and ingest the output

Sinks

- All sinks are polling-based
- Most sinks can process batches of events at a time
- Stock sink implementations (as of 1.4.0 RC1):
  - HDFS
  - HBase (2 variants)
  - SolrCloud
  - ElasticSearch
  - Avro-RPC, Thrift-RPC – Flume agent inter-connect
  - File Roller – write to local filesystem
  - Null, Logger, Seq, Stress – for testing purposes

Flume component interactions



- Source: Puts events into the local channel
- Channel: Store events until someone takes them
- Sink: Takes events from the local channel
  - On failure, sinks backoff and retry forever until success

Flume configuration example

agent1.properties:

```
# Flume will start these named components
agent1.sources = src1
agent1.channels = ch1
agent1.sinks = sink1

# channel config
agent1.channels.ch1.type = memory

# source config
agent1.sources.src1.type = netcat
agent1.sources.src1.channels = ch1
agent1.sources.src1.bind = 127.0.0.1
agent1.sources.src1.port = 8080

# sink config
agent1.sinks.sink1.type = logger
agent1.sinks.sink1.channel = ch1
```

Our Flume config

```
agent.channels = mem1
agent.sources = netcat
agent.sinks = hdfs1 hdfs2 hdfs3 hdfs4

agent.channels.mem1.type = memory
agent.channels.mem1.capacity = 1000000
agent.channels.mem1.transactionCapacity = 10000

agent.sources.netcat.type = netcat
agent.sources.netcat.channels = mem1
agent.sources.netcat.bind = 0.0.0.0
agent.sources.netcat.port = 8080
agent.sources.netcat.ack-every-event = false
```

```
agent.sinks.hdfs1.type = hdfs
agent.sinks.hdfs1.channel = mem1
agent.sinks.hdfs1.hdfs.useLocalTimeStamp = true
agent.sinks.hdfs1.hdfs.path =
  hdfs:///flume/webdata/avro/year=%Y/month=%m/
day=%d
agent.sinks.hdfs1.hdfs.fileType = DataStream
agent.sinks.hdfs1.hdfs.inUsePrefix = .
agent.sinks.hdfs1.hdfs.filePrefix = log
agent.sinks.hdfs1.hdfs.fileSuffix = .s1.avro
agent.sinks.hdfs1.hdfs.batchSize = 10000
agent.sinks.hdfs1.hdfs.rollInterval = 30
agent.sinks.hdfs1.hdfs.rollCount = 0
agent.sinks.hdfs1.hdfs.rollSize = 0
agent.sinks.hdfs1.hdfs.idleTimeout = 60
agent.sinks.hdfs1.hdfs.callTimeout = 25000
agent.sinks.hdfs1.serializer =
  com.cloudera.flume.demo.CSVAvroSerializer$Builder

agent.sinks.hdfs2.type = hdfs
agent.sinks.hdfs2.channel = mem1
# etc … sink configs 2 thru 4 omitted for brevity
```

Avro schema

```
{
  "type": "record",
  "name": "UserAction",
  "namespace": "com.cloudera.flume.demo",
  "fields": [
    {"name": "txn_id", "type": "long"},
    {"name": ”ts", "type": "long"},
    {"name": "action", "type": "string"},
    {"name": "product_id", "type": "long"},
    {"name": "user_ip", "type": "string"},
    {"name": "user_name", "type": "string"},
    {"name": "path", "type": "string"},
    {"name": "referer", "type": "string", "default":""}
  ]
}
```

Custom Event Parser / Avro Serializer

```
/** converts comma-separated-value (CSV) input
      text into binary Avro output on HDFS */

public class CSVAvroSerializer extends
   AbstractAvroEventSerializer<UserAction> {

 private OutputStream out;
 private CSVAvroSerializer(OutputStream out) {
  this.out = out;
 }

 protected OutputStream getOutputStream() {
  return out;
 }

 protected Schema getSchema() {
  Schema schema = UserAction.SCHEMA$;
  return schema;
 }
```

```
 protected UserAction convert(Event event) {
  String line = new String(event.getBody());
  String[] fields = line.split(",");
  UserAction action = new UserAction();
  action.setTxnId(Long.parseLong(fields[0]));
  action.setTimestamp(Long.parseLong(fields[1]));
  action.setAction(fields[2]);
  action.setProductId(Long.parseLong(fields[3]));
  action.setUserIp(fields[4]);
  action.setUserName(fields[5]);
  action.setPath(fields[6]);
  action.setReferer(fields[7]);
  return action;
 }

 public static class Builder {
  // Builder impl omitted for brevity
 }

}
```

And now our data flows into Hadoop

Send some data to Flume:

```
./generator.pl | nc localhost 8080
```

Flume does the rest:

```
$ hadoop fs -ls /flume/webdata/avro/year=2013/month=06/day=26
Found 16 items
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099633.s1.avro
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099633.s2.avro
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099633.s3.avro
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099633.s4.avro
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099634.s1.avro
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099634.s2.avro
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099634.s3.avro
/flume/webdata/avro/year=2013/month=06/day=26/log.1372244099634.s4.avro
. . .
```

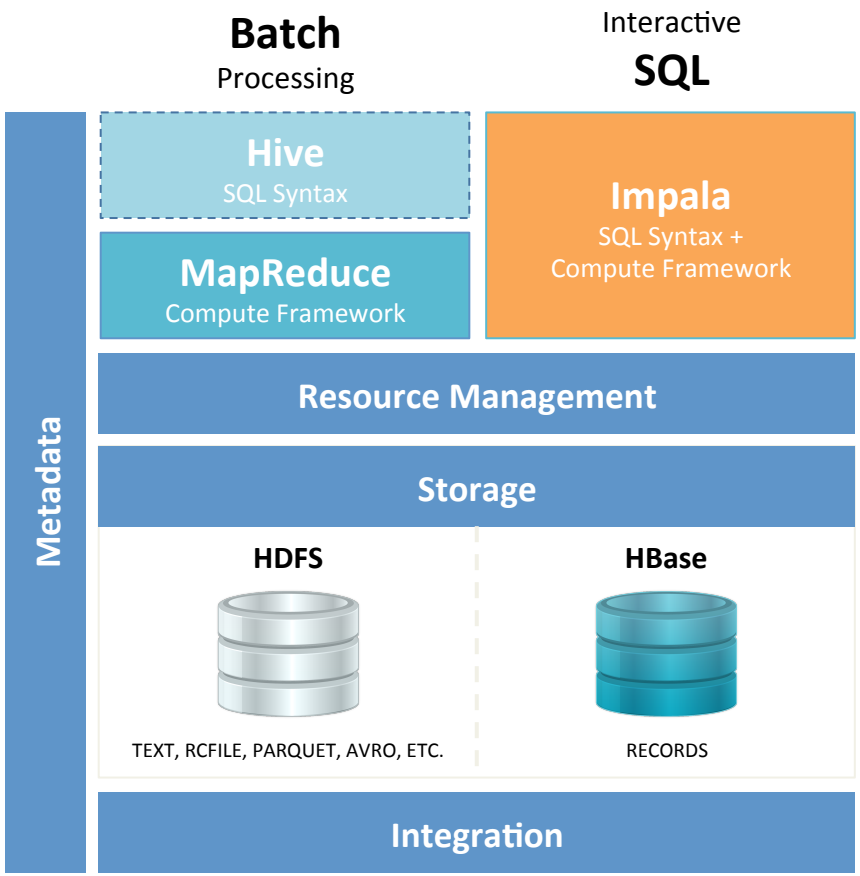
Querying our data via SQL: Impala and Hive

Shares Everything Client-Facing

- Metadata (table definitions)
- ODBC/JDBC drivers
- SQL syntax (Hive SQL)
- Flexible file formats
- Machine pool
- Hue GUI

But Built for Different Purposes

- **Hive:** runs on MapReduce and ideal for batch processing
- **Impala:** native MPP query engine ideal for interactive SQL



Cloudera Impala

Interactive SQL for Hadoop

- Responses in seconds
- Nearly ANSI-92 standard SQL with Hive SQL

Native MPP Query Engine

- Purpose-built for low-latency queries
- Separate runtime from MapReduce
- Designed as part of the Hadoop ecosystem

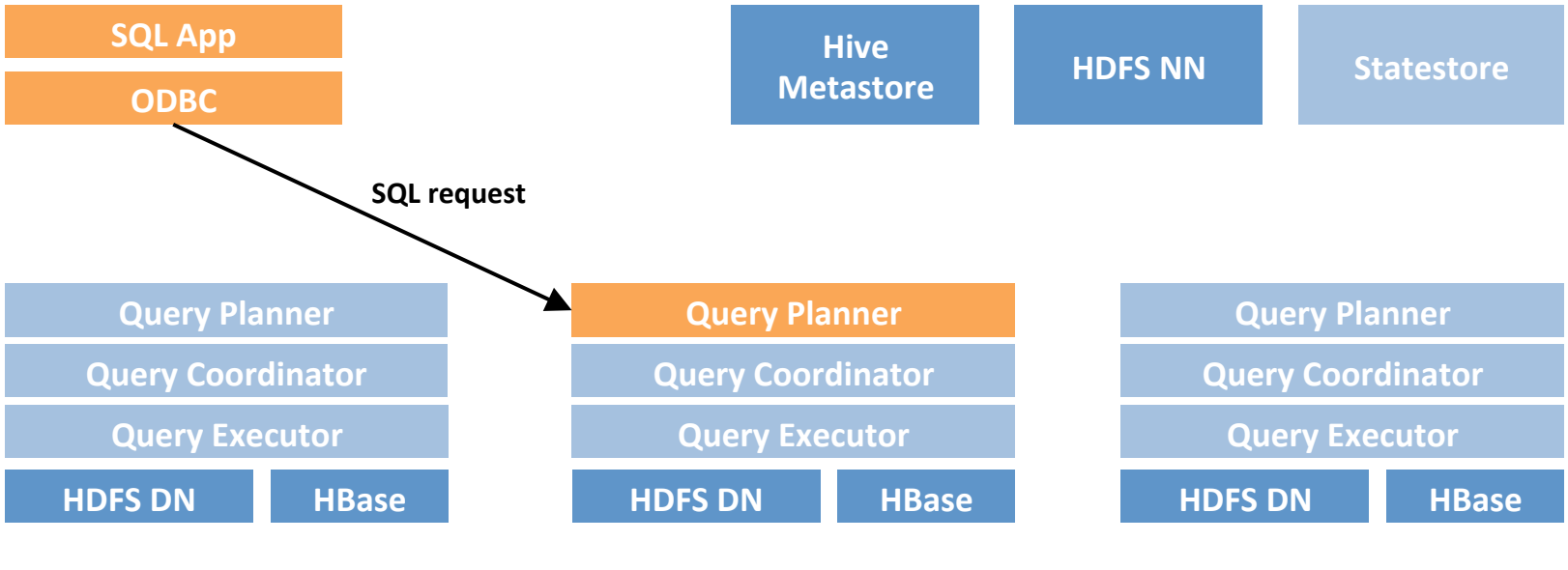
Open Source

- Apache-licensed

cloudera® IMPALA

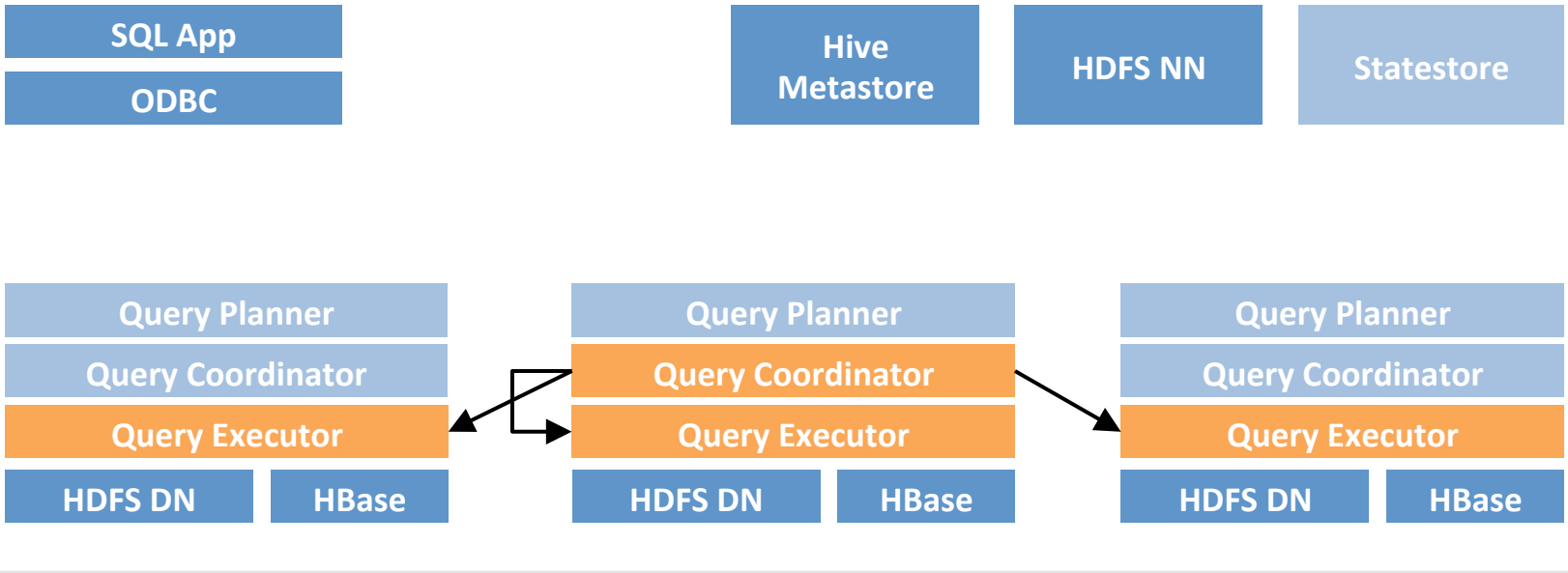
Impala Query Execution

1) Request arrives via ODBC/JDBC/Beeswax/Shell



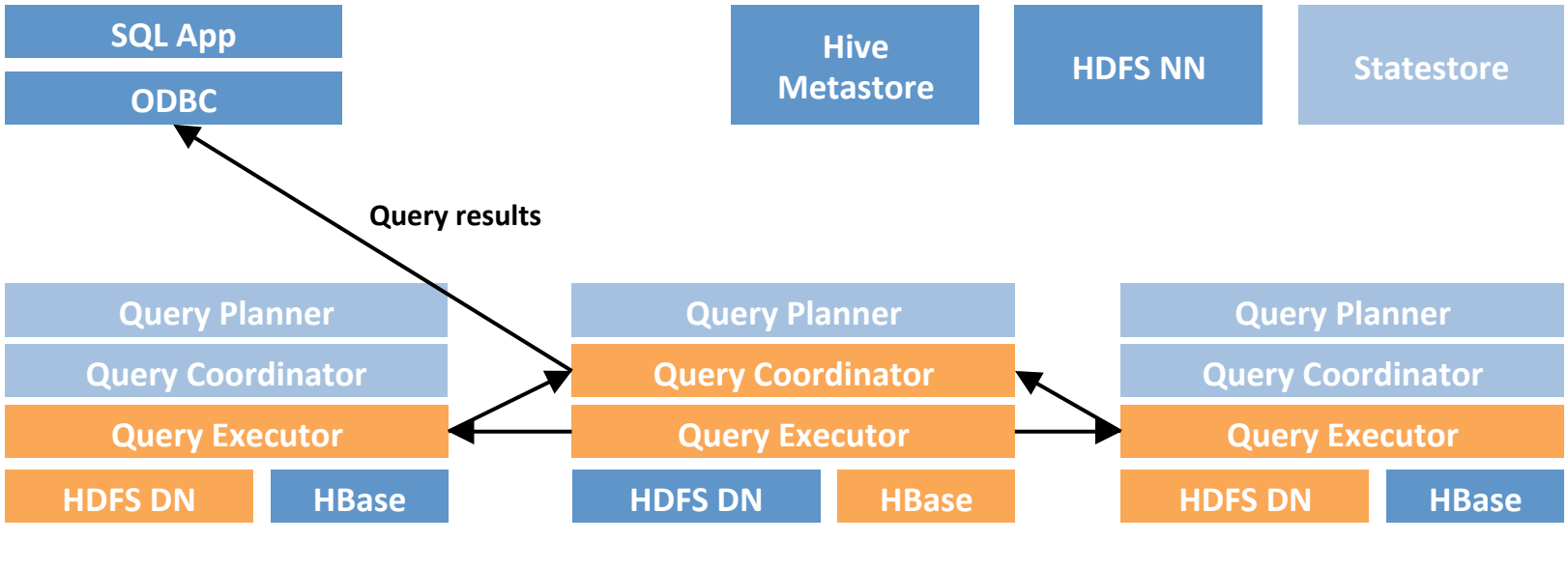
Impala Query Execution

2) Planner turns request into collections of plan fragments
3) Coordinator initiates execution on impalad(s) local to data



Impala Query Execution

4) Intermediate results are streamed between impalad(s)
5) Query results are streamed back to client



Parquet File Format: http://parquet.io

Open source, columnar file format for Hadoop developed by Cloudera & Twitter

- Rowgroup format: file contains multiple horiz. slices
- Supports storing each column in a separate file
- Supports fully shredded nested data
- Column values stored in native types
- Supports index pages for fast lookup
- Extensible value encodings

Table setup (using the Hive shell)

```
CREATE EXTERNAL TABLE webdataIn
  PARTITIONED BY (year int, month int, day int)
  ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.avro.AvroSerDe'
  STORED AS INPUTFORMAT
    'org.apache.hadoop.hive.ql.io.avro.AvroContainerInputFormat'
  OUTPUTFORMAT
    'org.apache.hadoop.hive.ql.io.avro.AvroContainerOutputFormat'
  LOCATION '/flume/webdata/avro/'
  TBLPROPERTIES
    ('avro.schema.url' = 'hdfs:///user/mpercy/UserAction.avsc');


/* this must be done for every new day: */
ALTER TABLE webdataIn
  ADD PARTITION (year = 2013, month = 06, day = 20);
```

Conversion to Parquet

Create the archive table from Impala:

```
impala> CREATE TABLE webdataArchive
  (txn_id bigint, ts bigint, action string, product_id bigint, user_ip string,
user_name string, path string, referer string)
  PARTITIONED BY (year int, month int, day int)
  STORED AS PARQUETFILE;
```

Conversion to Parquet

Load the Parquet table from an Impala SELECT:

```
ALTER TABLE webdataArchive
      ADD PARTITION (year = 2013, month = 6, day = 20);

INSERT OVERWRITE TABLE webdataArchive
 PARTITION (year = 2013, month = 6, day = 20)
 SELECT txn_id, ts, action, product_id, user_ip, user_name, path, referer
  FROM webdataIn
  WHERE year = 2013 AND month = 6 AND day = 20;
```

Partition update & metadata refresh

Impala can't see the Avro data until you add the partition and refresh the metadata for the table

- Pre-create partitions on a daily basis

```
hive> ALTER TABLE webdataIn
        ADD PARTITION (year = 2013, month = 6, day = 20);
```

- Refresh metadata for incoming table **every minute**

```
impala> REFRESH webdataIn
```

Querying against both data sets

Query against the correct table based on job schedule

```
SELECT txn_id, product_id, user_name, action
  FROM webdataIn
  WHERE year = 2013 AND month = 6
    AND (day = 25 OR day = 26)
    AND product_id = 30111
UNION ALL
SELECT txn_id, product_id, user_name, action
  FROM webdataArchive
  WHERE NOT (year = 2013 AND month = 6
    AND (day = 25 OR day = 26))
    AND product_id = 30111;
```

Query results

txn_id	product_id	user_name	action
1372244973556	30111	ylucas	click
1372244169096	30111	fdiaz	click
1372244563287	30111	vbernard	view
1372244268481	30111	ntrevino	view
1372244381025	30111	oramsey	click
1372244242387	30111	wwallace	click
1372244227407	30111	thughes	click
1372245609209	30111	gyoder	view
1372245717845	30111	yblevins	view
1372245733985	30111	moliver	click
1372245756852	30111	pjensen	view
1372245800509	30111	ucrane	view
1372244848845	30111	tryan	click
1372245073616	30111	pgilbert	view
1372245141668	30111	walexander	click
1372245362977	30111	agillespie	click
1372244892540	30111	ypham	view
1372245466121	30111	yowens	buy
1372245515189	30111	emurray	view

Views

- It would be much better to be able to define a view to simplify your queries instead of using UNION ALL
- Hive currently supports views
- Impala 1.1 will add VIEW support
  - Available next month (July 2013)

Other considerations

- Don't forget to delete the small files once they have been compacted
- Keep your incoming data table to a minimum
  - REFRESH <table> is expensive
- Flume may write duplicates
  - Your webdataIn table may return duplicate entries
  - Consider doing a de-duplication job via M/R before the INSERT OVERWRITE
  - Your schema should have keys to de-dup on
  - DISTINCT might help with both of these issues

Even lower latency via Flume & HBase

- HBase
  - Flume has 2 HBase sinks:
    - ASYNC_HBASE is faster, doesn't support Kerberos or 0.96 yet
    - HBASE is slower but uses the stock libs and supports everything
  - Take care of de-dup by using idempotent operations
    - Write to unique keys
  - Stream + Batch pattern is useful here if you need counters
    - Approximate counts w/ real-time increment ops
    - Go back and reconcile the streaming numbers with batch-generated accurate numbers excluding any duplicates created by retries

Flume ingestion strategies

- In this example, we used “netcat” because it’s easy
- For a system that needs reliable delivery we should use something else:
  - Flume RPC client API
  - Thrift RPC
  - HTTP source (REST)
  - Spooling Directory source
- Memory channel vs. File channel

For reference

Flume docs

- User Guide: flume.apache.org/FlumeUserGuide.html
- Dev Guide: flume.apache.org/FlumeDeveloperGuide.html

Impala tutorial:

- www.cloudera.com/content/cloudera-content/cloudera-docs/Impala/latest/Installing-and-Using-Impala/ciiu_tutorial.html

Flume 1.4.0

- Release candidate is out for version 1.4.0
- Adds:
  - SSL connection security for Avro source
  - Thrift source
  - SolrCloud sink
  - Flume embedding support (embedded agent)
  - File channel performance improvements
  - Easier plugin integration
  - Many other improvements & bug fixes
- Try it: http://people.apache.org/~mpercy/flume/apache-flume-1.4.0-RC1/

Flume: How to get involved!

- Join the mailing lists:
  - user-subscribe@flume.apache.org
  - dev-subscribe@flume.apache.org
- Look at the code
  - github.com/apache/flume – Mirror of the flume.apache.org git repo
- File a bug (or fix one!)
  - issues.apache.org/jira/browse/FLUME
- More on how to contribute:
  - cwiki.apache.org/confluence/display/FLUME/How+to+Contribute

cloudera
Ask Bigger Questions

Thank You!

Mike Percy
Work: Software Engineer, Cloudera
Apache: PMC Member & Committer, Apache Flume
Twitter: @mike_percy
Code: github.com/mpercy/flume-rtq-hadoop-summit-2013