Hadoop Tutorials

Daniel Lanza Zbigniew Baranowski

4 sessions

- Hadoop Foundations (today)
- Data Ingestion (20-July)
- Spark (3-Aug)
- Data Analytic tools and techniques (31-Aug)

Hadoop Foundations

Goals for today

- Introduction to Hadoop
- Explore and run reports on example data with Apache Impala (SQL)
- Visualize the result with HUE
- Evaluate different data formats and techniques to improve performance

Hands-on setup

- 12 node virtualized cluster
 - 8GB of RAM, 4 cores per node
 - 20GB of SSD storage per node
- Access (haperf10[1-12].cern.ch)
 - Everybody who subscribed should have the access
 - Try: ssh haperf105 'hdfs dfs -ls'
- List of commands and queries to be used
 \$> sh /afs/cern.ch/project/db/htutorials/tutorial_follow_up

What is Hadoop?

- A framework for large scale data processing
 - Data Volume (Terabytes, Zettabytes)
 - Data Variety (Structured, Unstructured)
 - Data Velocity (Stream processing)

What is Hadoop? Architecture

• Data locality (shared nothing) – scales out



What is Hadoop? Set of components



Hadoop Distributed File System

Hadoop cluster architecture

Master and slaves approach



Node 2

Node 3

Node 4

Node 5

Node X

HDFS in nutshell

- Distributed files system for Hadoop
 - Fault tolerant -> multiple replicas of data spread across a cluster
 - Scalable -> design to deliver high throughputs, sacrificing an access latency
 - Files cannot be modified in place
- Architecture
 - NameNode -> maintains and manages file system metadata (in RAM)
 - DataNodes -> store and manipulate the data (blocks)

How HDFS stores the data

1) File to be stored on HDFS



2) Splitting into 256MB blocks

4) Blocks with their replicas (by default 3) are distributed across Data Nodes

Ζ

3) Ask NameNode where to put them



Interacting with HDFS

Command line (examples)

hdfs	dfs	-ls	<pre>#listing home dir</pre>
hdfs	dfs	-ls /user	#listing user dir…
hdfs	dfs	- du -h /user	<pre>#space used</pre>
hdfs	dfs	- mkdir newdir	<pre>#creating dir</pre>
hdfs	dfs	<pre>-put myfile.csv .</pre>	<pre>#storing a file on HDFS</pre>
hdfs	dfs	-get myfile.csv .	<pre>#getting a file fr HDFS</pre>

• Programing bindings

– Java, Python, C++

7

More about HDFS: https://indico.cern.ch/event/404527/

Using Hadoop for data processing

- Get/produce the data
- Load data to Hadoop
- (optional) restructure it into optimized form
- Process the data (SQL, Scala, Java)
- Present/visualise the results

Using Hadoop for data processing

- Get/produce the data
- Load data to Hadoop
- (optional) restructure it into optimized form
- Process the data (SQL, Scala, Java)
- Present/visualise the results

Example data

• Source



- Meetups are: neighbours getting together to learn something, do something, share something...
- Streaming API
 - curl -s <u>http://stream.meetup.com/2/rsvps</u>

Using Hadoop for data processing

- Get/produce the data
- Load data to Hadoop
- (optional) restructure it into optimized form
- Process the data (SQL, Scala, Java)
- Present/visualise the results

Loading the data with HDFS command

- Store it locally and then move it to HDFS
 - curl -s http://stream.meetup.com/2/rsvps -o meetup_data.json
 - Ctrl + C
 - hdfs dfs -moveFromLocal meetup_data.json meetup.json
- Directly
 - curl -s http://stream.meetup.com/2/rsvps | head -10 | hdfs dfs -put - meetup.json
- Showing
 - hdfs dfs -cat meetup.json

Pre-proccesing required

Convert JSON to Parquet

- SparkSQL

```
> spark-shell
scala> val meetup_data = sqlContext.read.json("meetup.json")
scala> val sel = meetup_data.select("*").withColumnRenamed("group","group_info")
scala> sel.saveAsParquetFile("meetup_parquet")
```

• Convert to CSV with Impala

```
    Create external table
```

```
CREATE EXTERNAL TABLE meetup_parquet
```

LIKE PARQUETFILE '/user/<user_name>/meetup_parquet/<any_parquet_file>.gz.parquet'
STORED AS parquet
LOCATION '/user/<user_name>/meetup_parquet/';

Create table as select

CREATE TABLE meetup_csv

row format delimited fields terminated by '\t' ESCAPED BY '"' LINES TERMINATED BY '\n' AS SELECT

```
... all interesting columns ...
FROM meetup_parquet;
```

Using Hadoop for data processing

- Produce the data
- Load data to Hadoop
- (optional) restructure it into optimized form
- Process the data (SQL, Scala, Java)
- Visualise the results

Why SQL?

- It is simple and powerful
 - interactive, ad-hoc
 - declarative data processing
 - no need to compile
- Good for data exploration and reporting
- Structured data
 - organization of the data in table abstractions
 - optimized processing

Apache Impala

- MPP SQL query engine running on Apache Hadoop
- Low latency SQL queries on
 - Files stored on HDFS , Apache HBase and Apache Kudu
- Faster than Map-Reduce (Hive)
- C++, no Java GC



D More about Impala and Hive: https://indico.cern.ch/event/434650/



Creating our own table

• Create table

```
CREATE TABLE meetup_csv
```

```
(event_id string, event_name string, ...);
```

CREATE TABLE meetup_csv LIKE meetup csv;

Populate table

INSERT INTO meetup_csv
SELECT * FROM meetup_csv;

Create table as select

CREATE TABLE meetup_csv AS SELECT * from meetup_csv;

Querying the data

Counting records (SQL Hello world!)

SELECT count(*) FROM meetup_csv;

Most interesting meetups

SELECT DISTINCT event_name, group_name, venue_name
FROM meetup_csv
WHERE event_id IN
 (SELECT event_id FROM meetup_csv
 GROUP BY event_id ORDER BY count(*) desc
 LIMIT 10);

Not interesting meetings (people did not accept)

SELECT event_name, response, count(*)
 FROM meetup_csv
 WHERE response='no'
 GROUP BY event_name, response
 ORDER BY 3 desc;

Using Hadoop for data processing

- Produce the data
- Load data to Hadoop
- (optional) restructure it into optimized form
- Process the data (SQL, Scala, Java)
- Visualise the results

HUE – Hadoop User Experience

- Web interface to main Hadoop components – HDFS, Hive, Impala, Sqoop, Oozie, Solr etc.
- HDFS: FS browser, permission and ACLs configuration, file uploading
- SQL: query execution, results visualisation
- http://haperf100.cern.ch:8888/

How to check a profile of the execution

- Impala has build in query profile feature
 - \$ impala-shell
 - > SELECT event_name, event_url, member_name, venue_name, venue_lat, venue_lon FROM meetup_csv WHERE time BETWEEN unix_timestamp("2016-07-06 10:30:00")*1000 AND unix_timestamp("2016-07-06 12:00:00")*1000;

```
> profile;
```

- See execution plan
- Per machine or cluster average
 - How much data was read from HDFS
 - How much CPU time was spent on certain operations
 - etc.

profile

• Execution plan profile

Operator	#Hosts	Avg Time	Max Time	#Rows	Est. #Rows	Peak Mem	Est. Peak Mem	Detail
01:EXCHANGE 00:SCAN HDFS	1	0.000ns 1s076ms	0.000ns 1s375ms	971 971	-1 -1	0 24.77 MB	-1.00 B 176.00 MB	UNPARTITIONED zbaranow db.meetup csv
Planner Tim	meline:	3.236ms		•				

• Details for HDFS SCAN fragment (averaged)

H	DFS_SCAN_NODE (id=0):(Total: 1s076ms, non-child: 1s076ms, % non-child: 100.00
	 AverageHdfsReadThreadConcurrency: 0.75
	- AverageScannerThreadConcurrency: 1.00
	- BytesRead: 79.79 MB (83662047)
	- BytesReadDataNodeCache: 0
	- BytesReadLocal: 79.79 MB (83662047)
	- BytesReadRemoteUnexpected: 0
	- BytesReadShortCircuit: 79.79 MB (83662047)
	- DecompressionTime: 0.000ns
	- MaxCompressedTextFileLength: 0
	- NumDisksAccessed: 1 (1)
	- NumScannerThreadsStarted: 1 (1)
	- PeakMemoryUsage: 24.59 MB (25780224)
	- PerReadThreadRawHdfsThroughput: 89.60 MB/sec
	- RemoteScanRanges: 0 (0)
	- RowsRead: 212.61K (212613)
	- Kowsketurnea: 485 (485)
	- RowsReturnedRate: 404.00 /sec
	- ScanRangesComplete: 1 (1)
	- ScannerThreadsInvoluntaryContextSwitches: 7 (7)
	- ScannerThreadsTotalWallClockTime: 846.508ms
	- DelimiterParseTime: 155.501ms
	 MaterializeTupleTime(*): 172.001ms
	 ScannerThreadsSysTime: 0.000ns
	 ScannerThreadsUserTime: 324.950ms
	ScannerThreadeVoluntaryContextSwitches: 12 (12)
	- TotalRawHdfsReadTime(*): 801.008ms
	- TotalReadThroughput: 50.45 MB/sec

Can we optimize the execution?

- Reading all the data: 159.57MB
- Data are stored as text -> not optimally!

- Binary format?
- Apache Avro

Apache Avro data file



- Fast, binary serialization format
- Internal schema with multiple data types including nested ones
 - scalars, arrays, maps, structs, etc
- Schema in JSON

{

Ζ

```
"type": "record",
"name": "test",
"fields" : [
{"name": "a", "type":
"long"},
{"name": "b", "type":
"string"}
]
```

Record {a=27, b='foo'}



Creating Avro table in Impala

- Creating table
 CREATE TABLE meetup_avro LIKE meetup_csv
 STORED AS avro;
- Populating the table

```
INSERT INTO meetup_avro
    SELECT * FROM meetup_csv;
```

- Data size in Avro: 76MB (in CSV was 159MB)
- Run the queries

-~1.4s (in CSV was ~2s)

Can we do it better? (2)

• Still reading more (all) data than needed!

- What if data is stored in such a way that only a subset needs to be read
- Use **partitioning**!

Data partitioning (horizontal)

Group data by certain attribute(s) in separate directories

•	Wi	ll redu	ice ai	mount of da	ta to be read
	Day	Month	Year	No of customers	
	10	Aug	2013	17	Aug 2013
	11	Aug	2013	15	
	12	Aug	2013	21	/user/zaza/mydata/ Aug2U13 /data
	2	Dec	2014	30	
	3	Dec	2014	34	Dec 2014
	4	Dec	2014	31	
	17	Feb	2015	12	/user/zaza/mydata/ Dec2014 /data
	18	Feb	2015	16	
					Feb 2015

Partitioning the data with Impala

• Create a new partitioning table

CREATE TABLE meetup avro part (event_id string, event_name string, time bigint, event_url string, group_id bigint, group_name string, group city string, group country string, group lat double, group lon double, group_state string, group_urlname string, guests bigint, member id bigint, member name string, photo string, mtime bigint, response string, rsvp id bigint, venue id bigint, venue name string, venue lat double, venue lon double) PARTITIONED BY (year INT, month INT, day INT) STORED AS avro;

Partitioning the data with Impala

- Populating partitioning table
 - the data needs to be reload

```
INSERT INTO meetup_avro_part
PARTITION (year, month, day)
SELECT *,
    year(from_unixtime(cast(time/1000 as bigint))),
    month(from_unixtime(cast(time/1000 as bigint))),
    day(from_unixtime(cast(time/1000 as bigint)))
FROM meetup_avro;
```

- Impala will create automatically directories like:

/user/zaza/mydata/year=2016/month=7/day=6/data

• Filter predicates has to be specified on partitioning columns

where year=2016 and month=7 and day=6

Operator	#Hosts	Avg Time	Max Time	#Rows	Est. #Rows	Peak Mem	Est. Peak Mem	Detail
01:EXCHANGE	1	0.000ns	0.000ns	971	-1	0	-1.00 B	UNPARTITIONED
00:SCAN HDFS	1	358.004ms	358.004ms	971	-1	16.22 MB	32.00 MB	zbaranow_db.meetup_avro
partitions=1/	406 file	s=1 size=4	.32MB					

Can we do even better? (3)

- We are interested in reading certain columns
- But we are always reading entire row data
- Solution?
- Columnar store

Day	Month	Year	No of customers
10	Aug	2013	17
11	Aug	2013	15
12	Aug	2013	21
2	Dec	2014	30
3	Dec	2014	34
4	Dec	2014	31
17	Feb	2015	12
18	Feb	2015	16
Col	1 Col2	Col3	Col4

Parquet data format



Based on Google "Dremel"



Pushdowns

Vertical partitioning projection push down) + Horizontal partitioning (predicate push down) = Read only the data you need!										lata
а	b	С		а	b	с		а	b	С
a1	b1	c1		a1	b1	c1		a1	b1	c1
a2	b2	c2		a2	b2	c2		a2	b2	c2
a3	b3	сЗ	+	a3	b3	c3	=	a3	b3	c3
a4	b4	c4		a4	b4	c4		a4	b4	c4
a5	b5	c5		a5	b5	c5		a5	b5	c5

Slicing and dicing

 Horizontal and vertical partitioning – for efficient data processing

Day	Month	Year	No of customers	_				
10	Aug	2013	17		Col1	Col2	Col3	Col4
11	Aug	2013	15					
12	Aug	2013	21					
2	Dec	2014	30	•				
3	Dec	2014	34		Col1	Col2	Col3	Col4
4	Dec	2014	31					
17	Feb	2015	12					
18	Feb	2015	16		Col1	Col2	Col3	Col4

Horizontal and vertical partitioning

• Create a new table

CREATE TABLE meetup_parquet_part (event id string, event name string, time bigint, event_url string, group id bigint, group name string, group_city string, group_country string, group_lat double, group_lon double, group state string, group urlname string, guests bigint, member_id bigint, member_name string, photo string, mtime bigint, response string, rsvp_id bigint, venue_id bigint, venue name string, venue lat double, venue_lon double) PARTITIONED BY (year INT, month INT, day INT) STORED AS parquet;

Horizontal and vertical partitioning

- Populating partitioning table
 - the data needs to be reload

INSERT INTO meetup_parquet_part
PARTITION (year, month, day)
SELECT *,
 year(from_unixtime(cast(time/1000 as bigint))),
 month(from_unixtime(cast(time/1000 as bigint))),
 day(from_unixtime(cast(time/1000 as bigint)))
FROM meetup_avro;

– Size 42MB

• Run queries

Operator	#Hosts	Avg Time	Max Time	#Rows	Est. #Rows	Peak Mem	Est. Peak Mem	Detail
01:EXCHANGE	1	1.000ms	1.000ms	971	-1	0	-1.00 B	UNPARTITIONED
00:SCAN HDFS	1	299.002ms	299.002ms	971	-1	3.53 MB	112.00 MB	<pre>zbaranow_db.meetup_parquet</pre>
BytesRead: 1.	06 MB (1	115834)						

Can we query faster? (4)

- Use compression?
 - Snappy lightweight with decent compression rate
 - Gzip to save more space but affect performance

- Using an index?
- In Hadoop there is a 'format' that has an index
 -> HBase

HBase in a nutshell



- HBase is a **key-value** store on top of HDFS
 - horizontal (regions) + vertical (col. families) partitioning
 - row key values are indexed within regions
 - data typefree data stored in bytes arrays
- Fast random data access by key
- Stored data can be **modified** (updated, deleted)
- Has multiple bindings
 - SQL (Impala/Hive, Phoenix), Java, Python
- Very good for massive concurrent random data access
- ...but not good for big data **sequential** processing!

HBase: master-slaves architecture



HBase table data organisation



More about HBase: <u>https://indico.cern.ch/event/439742/</u>

Creating and loading data to an HBase table with SQL

• Creating HBase table (with 4 column families)

```
$ hbase shell
> create 'meetup_<username>', 'event', 'group', 'member', 'venue'
> quit
```

• Mapping Hive/Impala table to HBase table

```
$ hive
> CREATE EXTERNAL TABLE meetup_hbase
    (key string, event_id string, event_name string, time bigint, ...)
    STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
    WITH SERDEPROPERTIES ("hbase.columns.mapping" =
    ":key,event:event_name,event:time,event:event_url,group:group_id,group:groupup_name,group:group_city,group:group_country,group:group_lat,... ")
    TBLPROPERTIES("hbase.table.name" = "meetup_<username>");
```

Populating the table (key=event_time,even_if,modification time)

```
$ hive
> INSERT INTO meetup_hbase SELECT concat(
        cast(nvl(time, 0) as string), event_id, cast(mtime as string)), *
        FROM meetup_csv;
```

Query data by key on HBase through Impala/Hive

• Run queries

```
$ impala-shell
> SELECT *
    FROM meetup_hbase
    WHERE key BETWEEN "1462060800" AND "1467331200";
> SELECT *
    FROM meetup_hbase
    WHERE key BETWEEN
        cast(unix_timestamp("2016-07-06 10:30:00") as string)
    AND cast(unix_timestamp("2016-07-06 12:00:00") as string);
```

> SELECT * FROM meetup_hbase

```
WHERE key = '14679936000002319268721467404430663'
```

Operator	#Hosts	Avg Time	Max Time	#Rows	Est. #Rows	Peak Mem	Est. Peak Mem	Detail
01:EXCHANGE	1	0.000ns	0.000ns	644	729.44K	0	-1.00 B	UNPARTITIONED
00:SCAN HBASE	1	63.000ms	63.000ms	644	729.44K	4.00 KB	1.00 GB	

```
00:SCAN HBASE [zbaranow_db.meetup_hbase]
start key: 1467801000
stop key: 1467806400\0
```

Formats summary

Hands-on results





Production data





When to use what?

- **Partitioning** -> always when possible
- Fast full data (all columns) processing -> Avro
- Fast analytics on subset of columns -> **Parquet**
- Only when predicates on the same key columns -> HBase (data deduplication, low latency, parallel access)

- **Compression** in order to further reduce the data volume
 - without sacrificing performance -> Snappy
 - when data access is sporadic -> Gzip/Bzip or derived

Summary

- Hadoop is a framework for distributed data processing
 - designed to scale out
 - optimized for sequential data processing
 - HDFS is the core of the system
 - many components with multiple functionalities
- You do not have to be a Java guru to start using it
- Choosing data format, partitioning scheme is a key to achieve good performance and optimal resource utilisation

Questions & feedback

it-dep-db-sas@cern.ch