cloudera
Ask Bigger Questions

Practical Performance Analysis and Tuning for Cloudera Impala

Greg Rahn | @GregRahn

Strata + Hadoop World 2013 | #strataconf + #hw2013

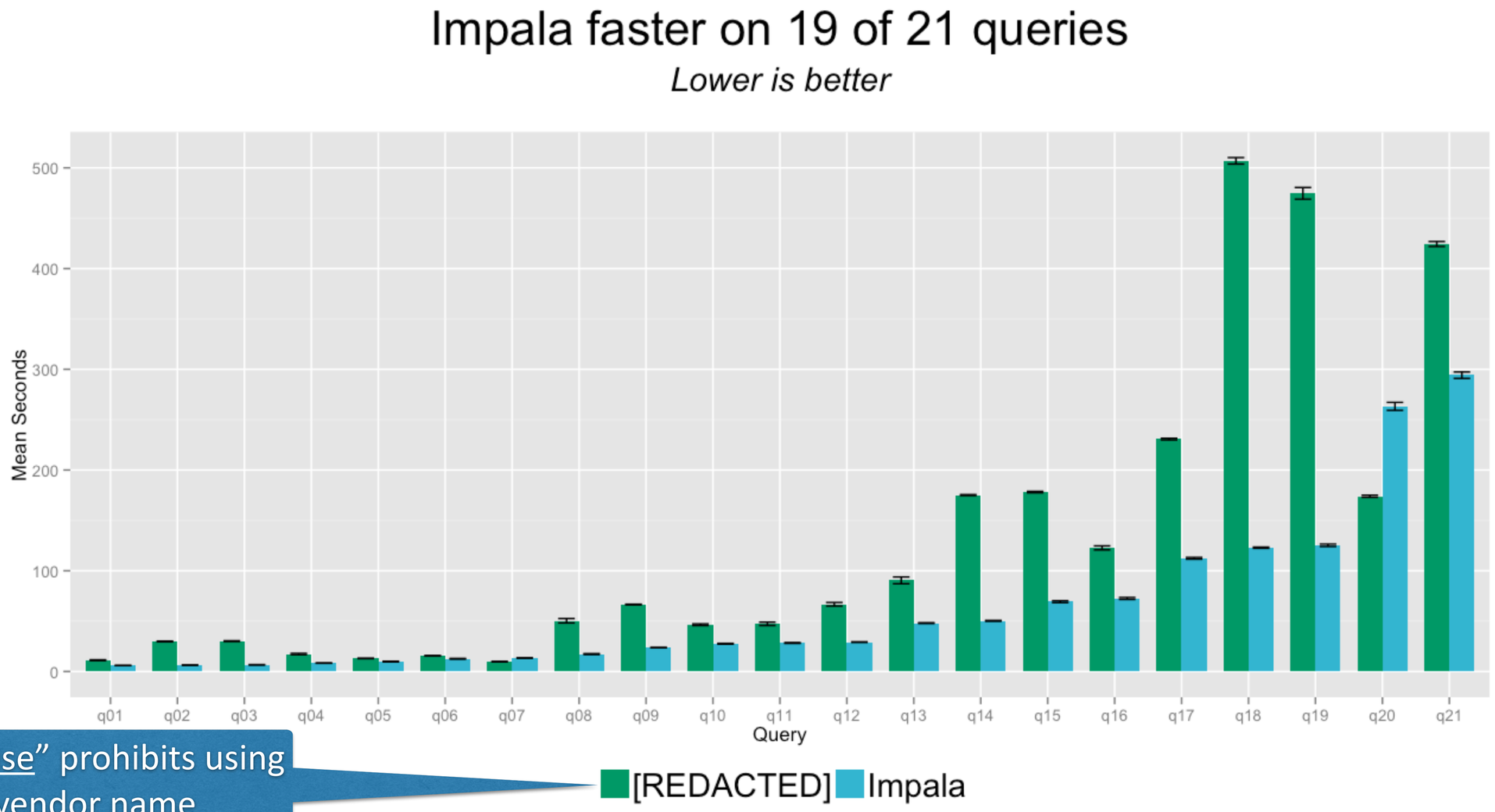
2013-10-30


Google
Greg Rahn
Web
Images
Maps
Shopping
Videos
More
Search tools
About 429,000 results (0.20 seconds)
Greg Rahn | LinkedIn
www.linkedin.com/in/gregrahn
San Francisco Bay Area - Big Data Aficionado
View Greg Rahn's professional profile on LinkedIn. LinkedIn is the world's largest business network, helping professionals like Greg Rahn discover inside ...
Greg Rahn (GregRahn) on Twitter
https://twitter.com/GregRahn
The latest from Greg Rahn (@GregRahn). big data nerd, database junkie, SQL guru, machine learning enthusiast, data scientist in training, vldb performance ...
Structured Data | Thoughts on: Big Data, Hadoop, Databases ...
structureddata.org/
10 hours ago - Office Hour with Greg Rahn @ the Cloudera Booth 10/29/2013 11:00am – 11:30am EDT (30 minutes) Room: 3rd Floor, Mercury Ballroom, ...

Just how fast is Impala?

You might just be surprised

How fast is Impala?



"DeWitt Clause" prohibits using DBMS vendor name

cloudera
Ask Bigger Questions

Practical Performance

Running with Impala

Pre-execution Checklist

- Data types
- Partitioning
- File Format

Data Type Choices

- Define integer columns as INT/BIGINT
  - Operations on INT/BIGINT more efficient than STRING
- Convert “external” data to good “internal” types on load
  - e.g. CAST date strings to TIMESTAMPS
  - This avoids expensive CASTs in queries later

Partitioning

- **"The fastest I/O is the one that never takes place."**
- Understand your query filter predicates
  - For time-series data, this is usually the date/timestamp column
  - Use this/these column(s) for a partition key(s)
- Validate queries leverage partition pruning using EXPLAIN
- You *can* have too much of a good thing
  - A few thousand partitions per table is probably OK
  - Tens of thousands partitions is probably too much
  - Partitions/Files should be no less than a few hundred MBs

Partition Pruning in EXPLAIN

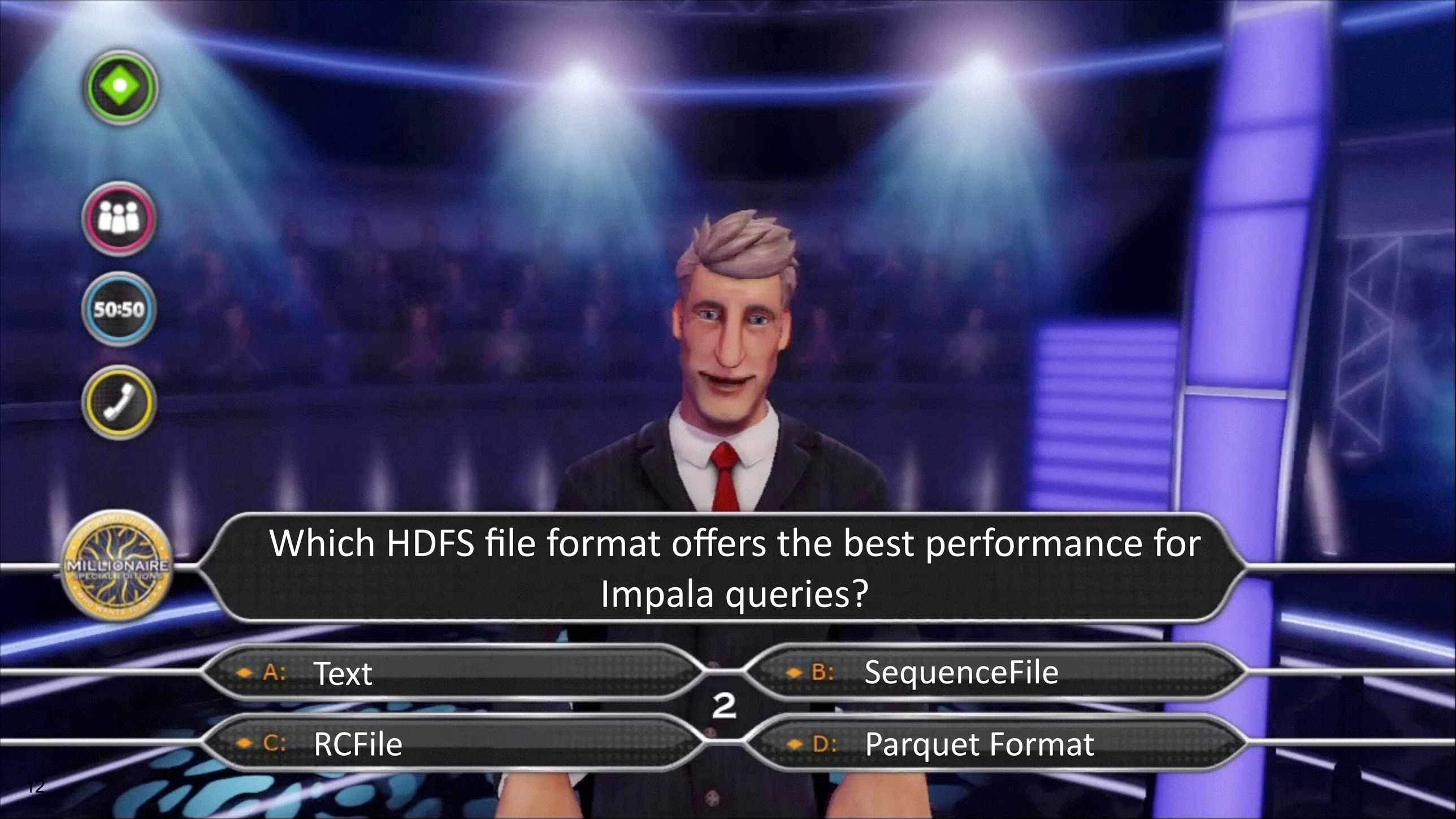
```
explain
select count(*)
from sales_fact
where sold_date in('2000-01-01','2000-01-02');

- Partition Pruning (partitioned table)
0:SCAN HDFS
   table=grahn.sales_fact #partitions=2 size=803.15MB
   tuple ids: 0

- Filter Predicate (non-partitioned table)
0:SCAN HDFS
   table=grahn.sales_fact #partitions=1 size=799.42MB
   predicates: sold_date IN ('2000-01-01', '2000-01-02')
   tuple ids: 0
```

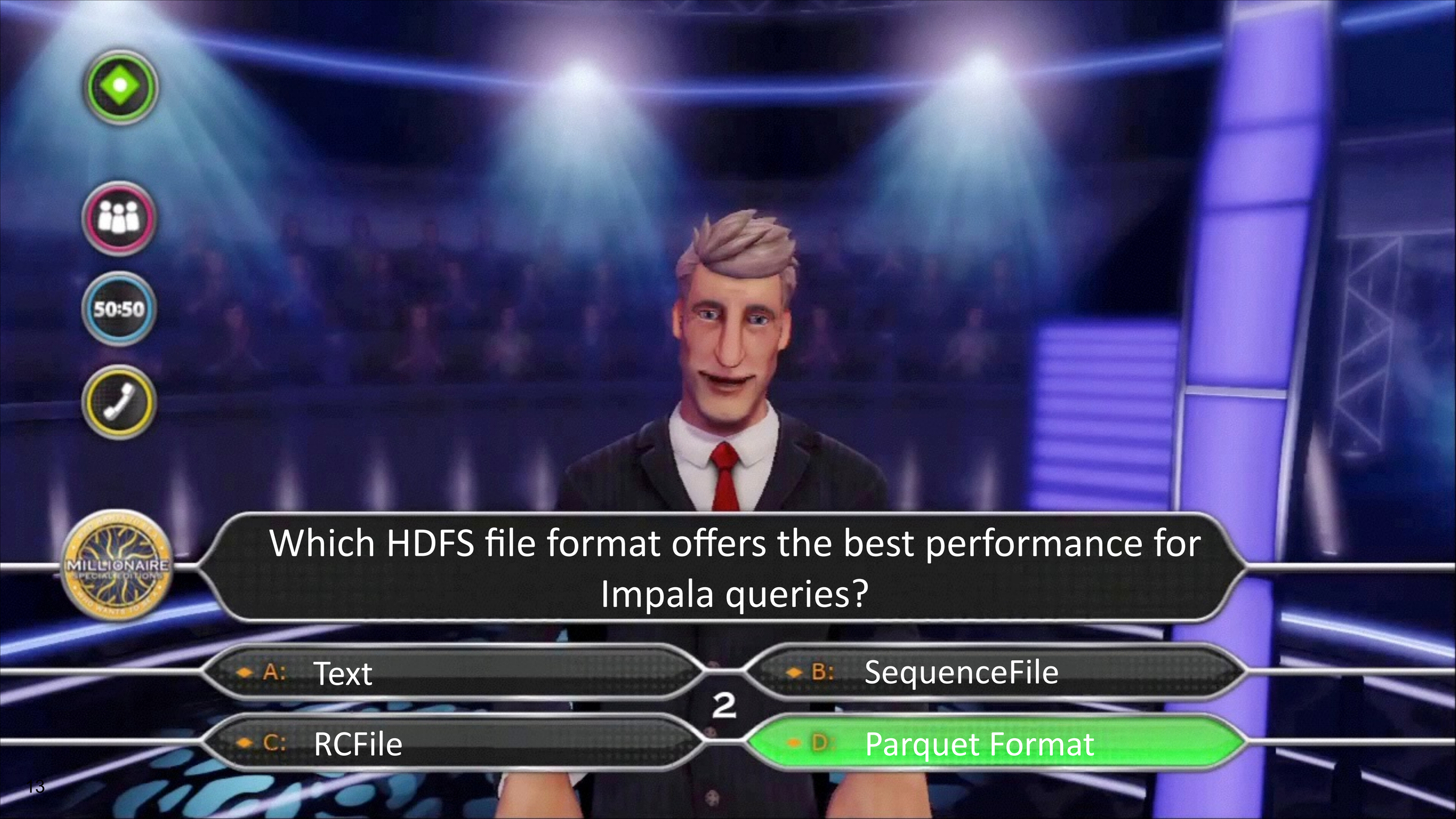
Note the partition count and missing “predicates” filter due to parse time partition pruning

File Format Choices

What format is optimal for Impala?

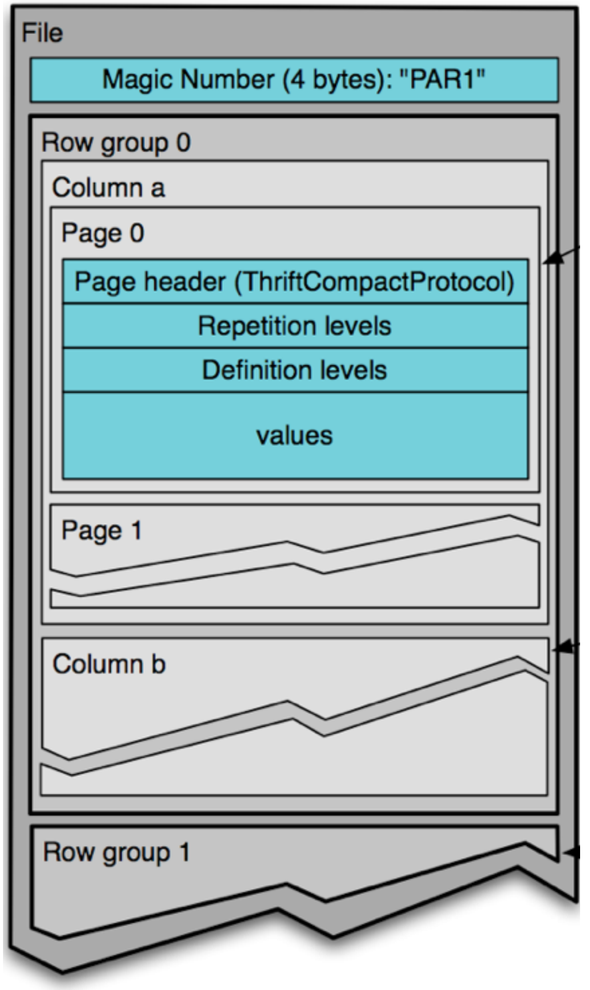

WHO WANTS TO BE A
MILLIONAIRE
SPECIAL EDITIONS
WHO WANTS TO BE A
Press START


50:50
MILLIONAIRE
Which HDFS file format offers the best performance for Impala queries?
A: Text
B: SequenceFile
2
C: RCFile
D: Parquet Format


50:50
MILLIONAIRE
Which HDFS file format offers the best performance for Impala queries?
A: Text
B: SequenceFile
2
C: RCFile
D: Parquet Format

Why use Parquet Columnar Format for HDFS?

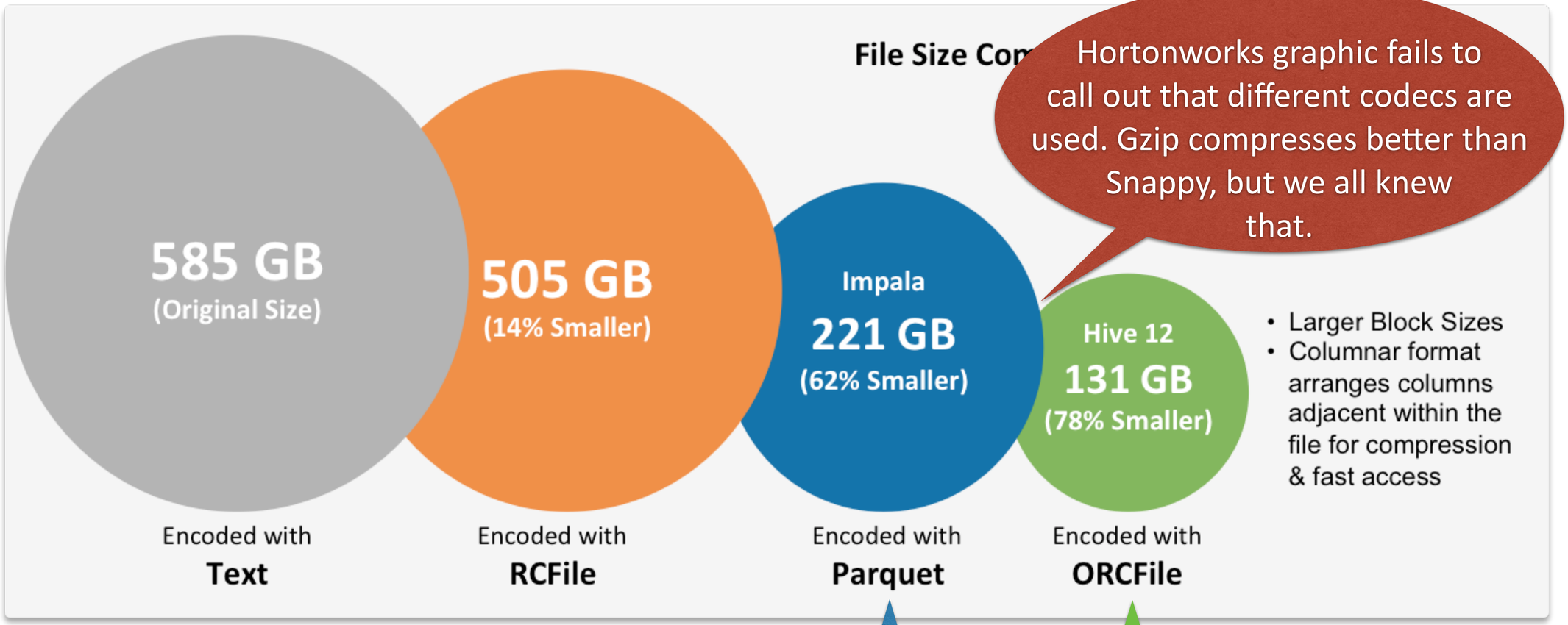
- Well defined open format - http://parquet.io/
  - Works in Impala, Pig, Hive & Map/Reduce
- I/O reduction by only reading necessary columns
- Columnar layout compresses/encodes better
- Supports nested data by shredding columns
  - Uses techniques used by Google's ColumnIO
- Impala loads use Snappy compression by default
  - Gzip available: set PARQUET_COMPRESSION_CODEC=gzip;
- Quick word on Snappy vs. Gzip



Quick Note on Compression

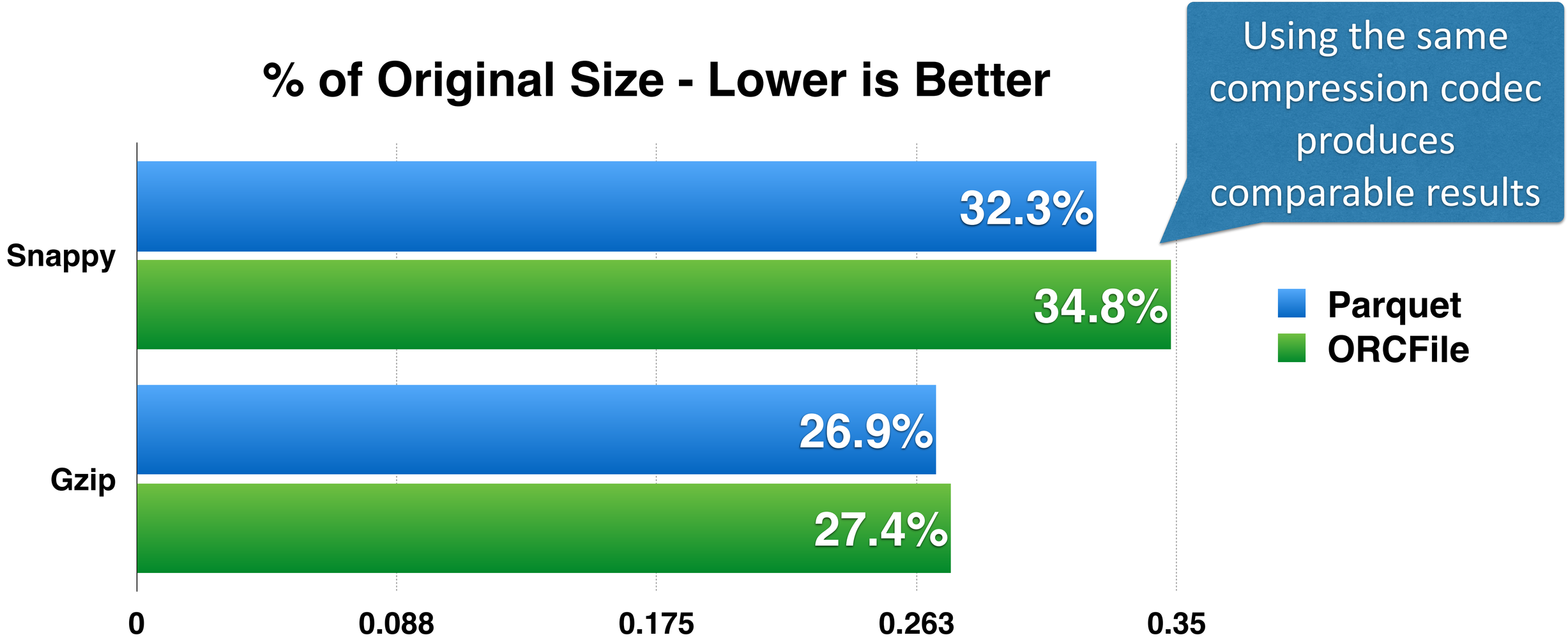
- Snappy
  - Faster compression/decompression speeds
  - Less CPU cycles
  - Lower compression ratio
- Gzip/Zlib
  - Slower compression/decompression speeds
  - More CPU cycles
  - Higher compression ratio
- It's all about trade-offs

It’s all about the compression codec



Source: http://hortonworks.com/blog/orcfile-in-hdp-2-better-compression-better-performance/

TPC-DS 500GB Scale Factor



Dremel made simple with Parquet

Wednesday, September 11, 2013 | By Julien Le Dem (@J_) [16:04 UTC]

Tweet

Columnar storage is a popular technique to optimize analytical workloads in parallel RDBMs. The performance and compression benefits for storing and processing large amounts of data are well documented in academic literature as well as several commercial analytical databases.

The goal is to keep I/O to a minimum by reading from a disk only the data required for the query. Using Parquet at Twitter, we experienced a reduction in size by one third on our large datasets. Scan times were also reduced to a fraction of the original in the common case of needing only a subset of the columns. The principle is quite simple: instead of a traditional row layout, the data is written one column at a time. While turning rows into columns is straightforward given a flat schema, it is more challenging when dealing with nested data structures.

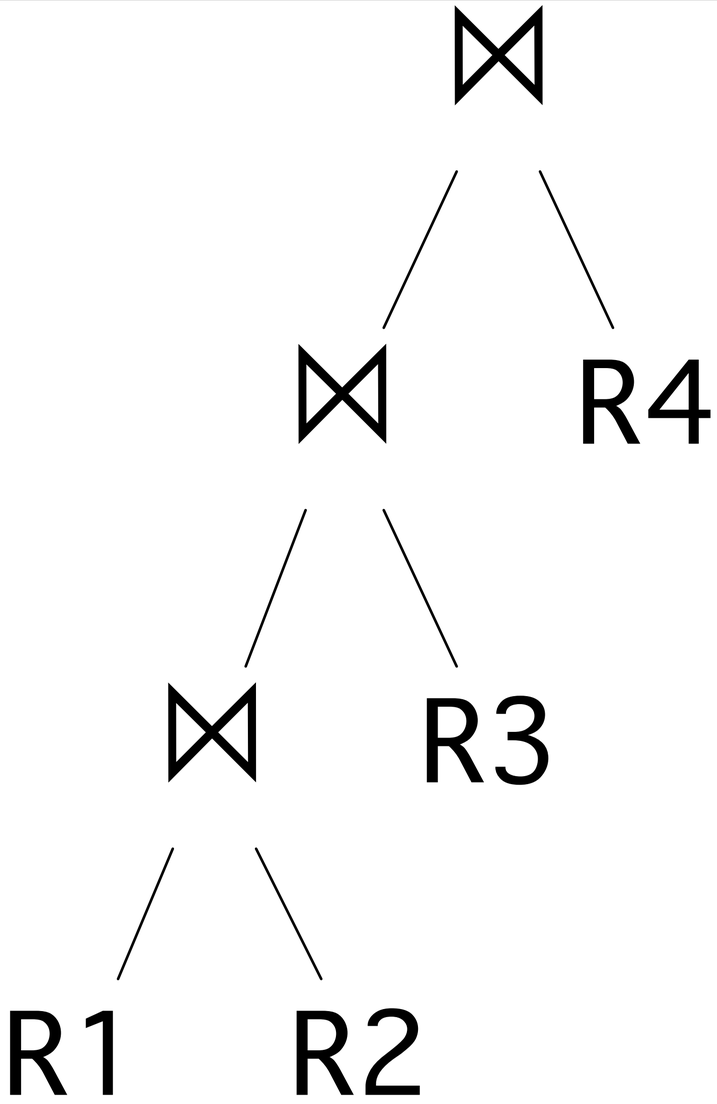
We recently introduced Parquet, an open source file format for Hadoop that provides columnar storage. Initially a joint effort between Twitter and Cloudera, it now has many other contributors including companies like Criteo. Parquet stores nested data structures in a flat columnar format using a technique outlined in the Dremel paper from Google. Having implemented this model based on the paper, we decided to provide a more accessible explanation. We will first describe the general model used to

Query Execution

What happens behind the scenes

Left-Deep Join Tree

- The **largest* table** should be **listed first** in the FROM clause
- Joins are done in **the order tables are listed** in FROM clause
- **Filter early** - most selective joins/tables first
- v1.2.1 will do JOIN ordering



Two Types of Hash Joins

- Default hash join type is **BROADCAST** (aka replicated)
  - Each node ends up with a copy of the right table(s)*
  - Left side, read locally and streamed through local hash join(s)
  - Best choice for “star join”, single large fact table, multiple small dims
- Alternate hash join type is **SHUFFLE** (aka partitioned)
  - Right side hashed and shuffled; each node gets ~1/Nth the data
  - Left side hashed and shuffled, then streamed through join
  - Best choice for “large_table JOIN large_table”
  - Only available if ANALYZE was used to gather table/column stats*

Hinting Joins

```
select ...
from large_fact
join [broadcast] small_dim
```

```
select ...
from large_fact
join [shuffle] large_dim
```

*square brackets required

Determining Join Type From EXPLAIN

```
explain
select
  s_state,
  count(*)
from store_sales
join store on (ss_store_sk = s_store_sk)
group by
  s_state;

2:HASH JOIN
|  join op: INNER JOIN (BROADCAST)
|  hash predicates:
|    ss_store_sk = s_store_sk
|  tuple ids: 0 1
|
|----4:EXCHANGE
|       tuple ids: 1
|
0:SCAN HDFS
  table=tpcds.store_sales
  tuple ids: 0
```

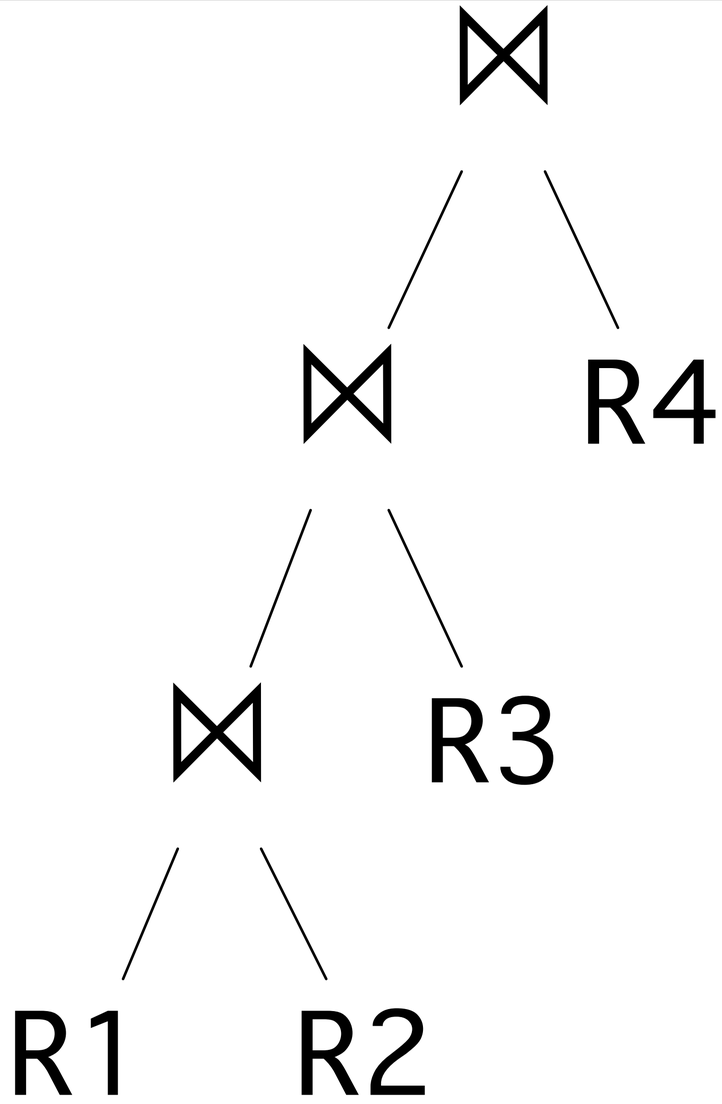
```
explain
select
  c_preferred_cust_flag,
  count(*)
from store_sales
join customer on (ss_customer_sk = c_customer_sk)
group by
  c_preferred_cust_flag;

2:HASH JOIN
|  join op: INNER JOIN (PARTITIONED)
|  hash predicates:
|    ss_customer_sk = c_customer_sk
|  tuple ids: 0 1
|
|----5:EXCHANGE
|       tuple ids: 1
|
4:EXCHANGE
tuple ids: 0
```

Memory Requirements for Joins & Aggregates

- Impala does not “spill” to disk -- pipelines are in-memory
- Operators’ mem usage need to fit within the memory limit
- This **is not** the same as “*all data needs to fit in memory*”
  - Buffered data generally significantly smaller than total accessed data
  - Aggregations’ mem usage proportional to number of groups
- Applies for each in-flight query (sum of total)
- Minimum of 128GB of RAM is recommended for impala nodes

Understanding Operators & Data Flow

- Data is streamed directly between operators
- A **producer** can only send data as fast as the **consumer** can ingest it
- A “slow” operator may slow down the entire pipeline



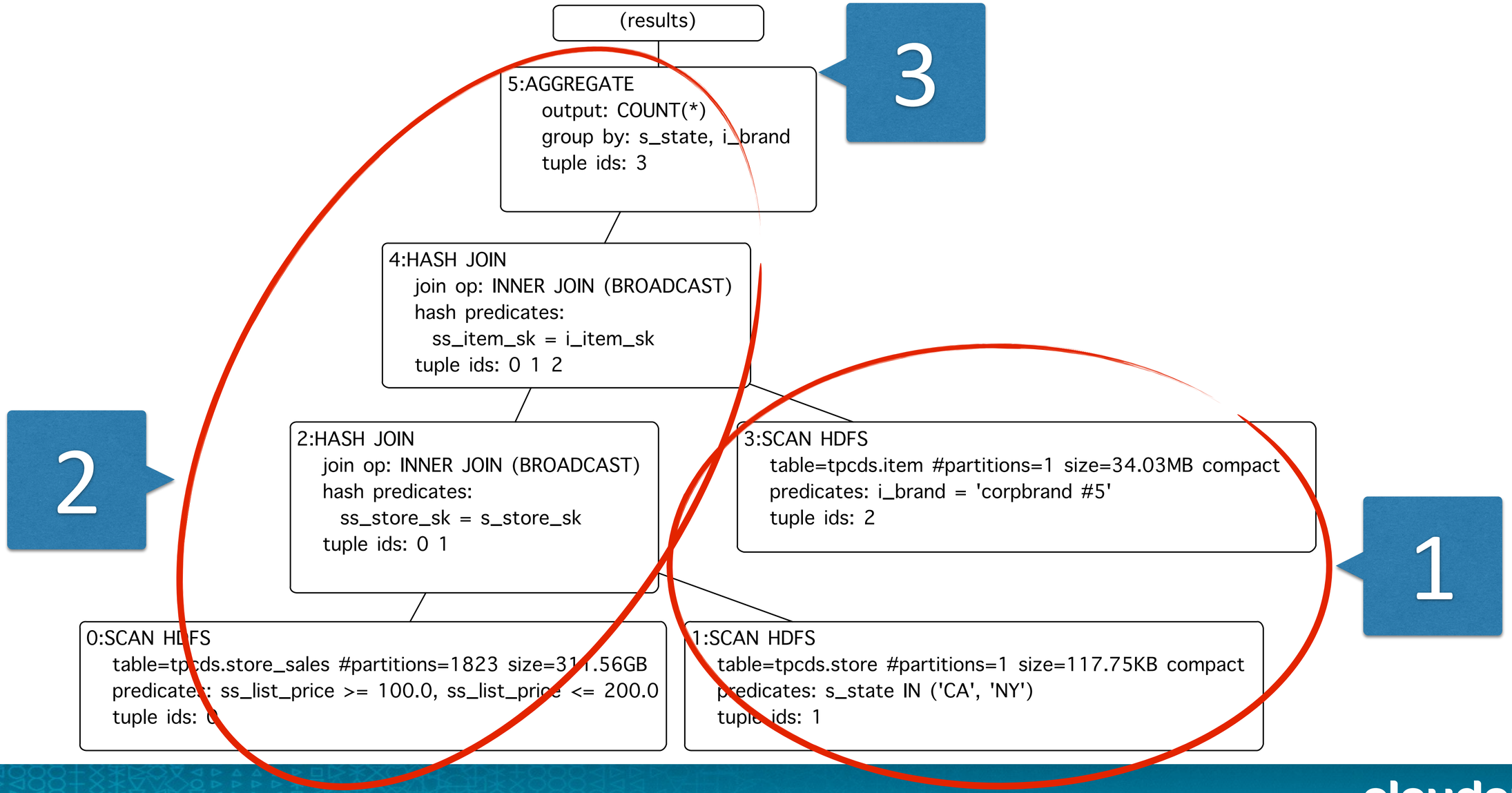
How to use ANALYZE in the Hive shell

- Table Stats
  - analyze table **unpartitioned_tab** compute statistics;
  - analyze table **partitioned_tab** partition(**partition_key**) compute statistics;
- Column Stats
  - analyze table **unpartitioned_tab** compute statistics for columns **c1,c2,...**
  - analyze table **partitioned_tab** partition(**partition_key**) compute statistics for columns **c1,c2,...**

Query Execution Walkthrough

```
select
  s_state,
  i_brand,
  count(*)
from store_sales
join store on (ss_store_sk = s_store_sk)
join item on (ss_item_sk = i_item_sk)
where ss_list_price between 100.0 and 200.0
  and s_state in ('CA', 'NY')
  and i_brand = 'corpbrand #5'
group by
  1,2;
```

Logical Execution Plan



Query Execution Plan

```
5:AGGREGATE
|  output: COUNT(*)
|  group by: s_state, i_brand
|
4:HASH JOIN
|  join op: INNER JOIN (BROADCAST)
|  hash predicates:
|    ss_item_sk = i_item_sk
|
|----7:EXCHANGE
|       tuple ids: 2
|
2:HASH JOIN
|  join op: INNER JOIN (BROADCAST)
|  hash predicates:
|    ss_store_sk = s_store_sk
|
|----6:EXCHANGE
|       tuple ids: 1
|
0:SCAN HDFS
   table=tpcds.store_sales #partitions=1823 size=311.56GB
   predicates: ss_list_price >= 100.0, ss_list_price <= 200.0
```

Impala Debug Web Pages

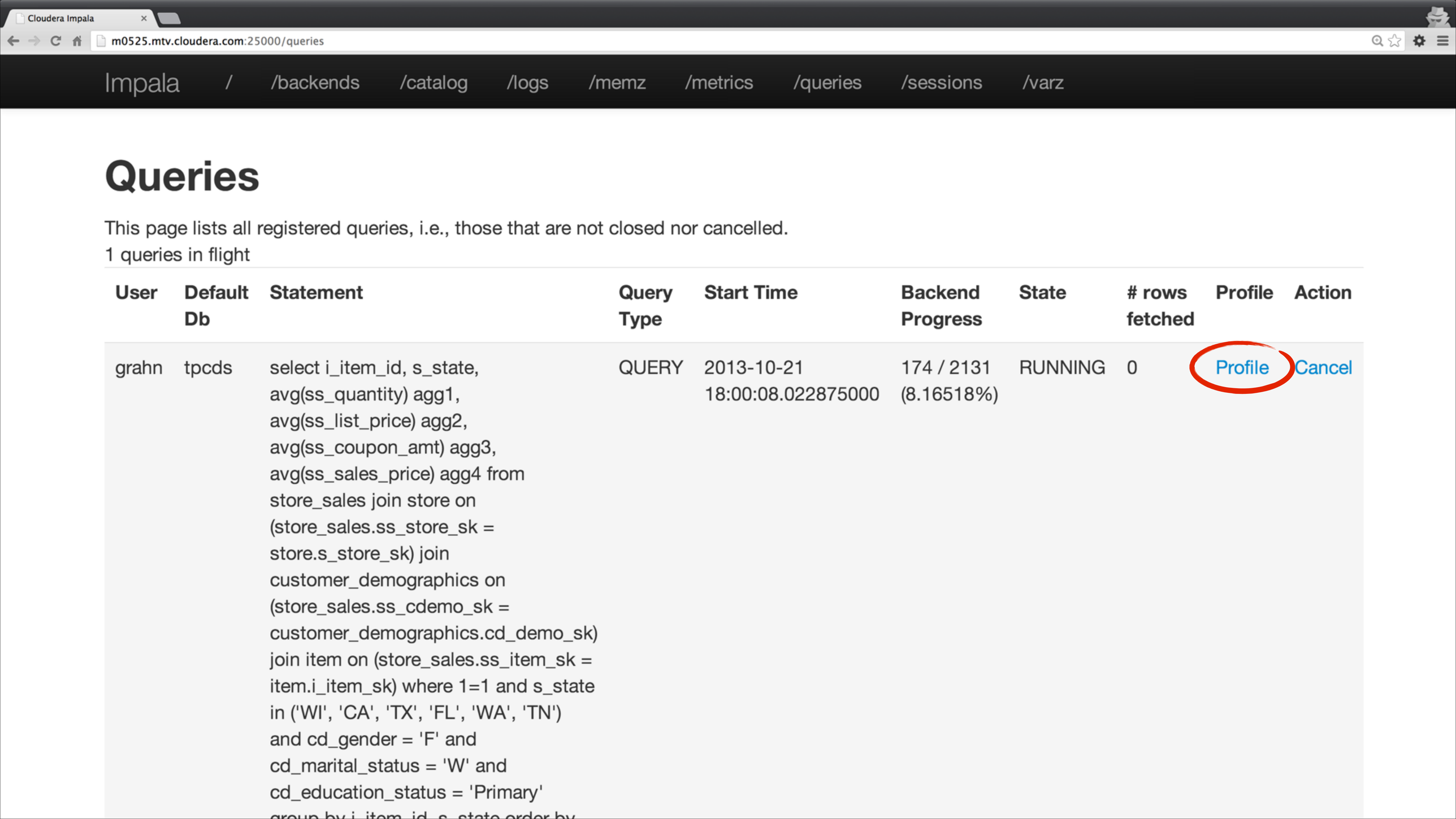
A great resource for debug information

Version

```
impalad version 1.1.1 RELEASE (build 83d5868f005966883a918a819a449f636a5b3d5f)
Built on Fri, 23 Aug 2013 17:52:25 PST
```

Hardware Info

```
Cpu Info:
  Model: Intel(R) Xeon(R) CPU           L5630  @ 2.13GHz
  Cores: 16
  L1 Cache: 32.00 KB
  L2 Cache: 256.00 KB
  L3 Cache: 12.00 MB
  Hardware Supports:
    ssse3
    sse4_1
    sse4_2
    popcnt
Mem Info: 94.47 GB
Disk Info:
  Num disks 15:
    sda (rotational=true)
    sdb (rotational=true)
    sdc (rotational=true)
    sde (rotational=true)
    sdf (rotational=true)
    sdg (rotational=true)
```


Cloudera Impala
m0525.mtv.cloudera.com:25000/queries
Impala / /backends /catalog /logs /memz /metrics /queries /sessions /varz
Queries
This page lists all registered queries, i.e., those that are not closed nor cancelled.
1 queries in flight
User
Default Db
Statement
Query Type
Start Time
Backend Progress
State
rows fetched
Profile
Action
grahn
tpcds
select i_item_id, s_state, avg(ss_quantity) agg1, avg(ss_list_price) agg2, avg(ss_coupon_amt) agg3, avg(ss_sales_price) agg4 from store_sales join store on (store_sales.ss_store_sk = store.s_store_sk) join customer_demographics on (store_sales.ss_cdemo_sk = customer_demographics.cd_demo_sk) join item on (store_sales.ss_item_sk = item.i_item_sk) where 1=1 and s_state in ('WI', 'CA', 'TX', 'FL', 'WA', 'TN') and cd_gender = 'F' and cd_marital_status = 'W' and cd_education_status = 'Primary'
QUERY
2013-10-21 18:00:08.022875000
174 / 2131 (8.16518%)
RUNNING
0
Profile
Cancel

Query Locations

Location	Number of Fragments
m0523.mtv.cloudera.com:22000	1
m0521.mtv.cloudera.com:22000	1
m0519.mtv.cloudera.com:22000	1
m0525.mtv.cloudera.com:22000	1
m0517.mtv.cloudera.com:22000	1

Finished Queries

User	Default Db	Statement	Query Type	Start Time	End Time	Backend Progress	State	# rows fetched	Profile
grahn	default	use tpcds	DDL	2013-10-21 16:42:44.956126000	2013-10-21 16:42:44.961133000	N/A	FINISHED	0	Profile
grahn	tpcds	select i_item_id, s_state, avg(ss_quantity) agg1, avg(ss_list_price) agg2, avg(ss_coupon_amt) agg3, avg(ss_sales_price) agg4 from	QUERY	2013-10-21 16:39:07.710767000	2013-10-21 16:40:41.146307000	2131 / 2131 (100%)	FINISHED	100	Profile

What is the hardware doing?

Monitoring system resources

collectl

collectl.sourceforge.net

Collectl

sourceforge

Latest Version: 3.6.9 October 18 2013

Download Now

warning: I'm declaring sexpr and nvidia deprecated and will remove them from the kit in 2014

There are a number of times in which you find yourself needing performance data. These can include benchmarking, monitoring a system's general heath or trying to determine what your system was doing at some time in the past. Sometimes you just want to know what the system is doing right now. Depending on what you're doing, you often end up using different tools, each designed to for that specific situation.

Unlike most monitoring tools that either focus on a small set of statistics, format their output in only one way, run either interatively or as a daemon but not both, collectl tries to do it all. You can choose to monitor any of a broad set of subsystems which currently include buddyinfo, cpu, disk, inodes, infiniband, lustre, memory, network, nfs, processes, quadrics, slabs, sockets and tcp.

The following is an example taken while writing a large file and running the collectl command with no arguments. By default it shows cpu, network and disk stats in *brief format*. The key point of this format is all output appears on a single line making it much easier to spot spikes or other anomalies in the output:

```
[mjs@poker] collectl

#<--------CPU--------><-----------Disks-----------><-----------Network---------->
#cpu sys inter  ctxsw KBRead  Reads  KBWrit Writes netKBi pkt-in  netKBo pkt-out
  37  37   382    188      0      0   27144    254     45     68       3      21
  25  25   366    180     20      4   31280    296      0      1       0       0
  25  25   368    183      0      0   31720    275      2     20       0       1
```

In this example, taken while writing to an NFS mounted filesystem, collectl displays interrupts, memory usage and nfs activity with timestamps. Keep in mind that you can mix and match any data and in the case of *brief format* you simply need to have a window wide enough to accommodate your output.

```
[mjs@poker] collectl -sjmf -oT

#          <-------Int--------><-----------Memory-----------><------NFS Totals------>
#Time      Cpu0 Cpu1 Cpu2 Cpu3 Free Buff Cach Inac Slab  Map  Reads Writes Meta Comm
08:36:52  1001   66    0    0   2G 201M 609M 363M 219M 106M      0      0    5    0
08:36:53   999 1657    0    0   2G 201M   1G 918M 252M 106M      0  12622    0    2
08:36:54  1001 7488    0    0   1G 201M   1G   1G 286M 106M      0  20147    0    2
```

You can also display the same information in *verbose format*, in which case you get a single line for each type of data at the expense of more screen real estate, as can be seen in this example of network data during NFS writes. Note how you can actually see the network traffic stall while waiting for the server to physically write the data.

```
[mjs@poker] collectl -sn --verbose -oT

# NETWORK SUMMARY (/sec)
#          KBIn  PktIn SizeIn  MultI   CmpI  ErrIn  KBOut PktOut  SizeO   CmpO ErrOut
08:46:35   3255  41000     81      0      0      0 112015  78837   1454      0      0
08:46:36      0      9     70      0      0      0     29     25   1174      0      0
08:46:37      0      2     70      0      0      0      0      2    134      0      0
```

collectl Utilities

collectl-utils.sourceforge.net

Collectl Utilities

sourceforge

Latest Version: 4.7.1, March 20, 2013

Download Now

NEW! colgui will be removed from kit in 2014 - you should use colmux!

NEW! The version numbering has now changed so collectl-utils, colplot and colmux are all the same

The focus of collectl has always been efficient performance data collection and its display on a single machine. This set of utilities have been developed to enhace the use of collectl in 2 dimensions:

- graphics
- multi-system support

Graphics

Colplot is a web-based plotting utility that uses gnuplot to generate plots against collectl-generated files that have been generated in plot format. The sample plot on the collectl home page was generated with colplot.

There are over 70 *standard* plots and a definition language that allows you to define your own if none of the existing ones meet your needs. If there are files for more than one system, colplot will generate separate plots for each system. Colplot also has an option that allows it to periodically redisplay the plots, which means if the files you point it to are being updated in real-time, colplot can show a dynamic plot. It can also save plots as invidual png files, as pdf files if ghostscript is installed or even email them to you. There is also a command line interface that will run on an X-enabled terminal.

Colgui is a utility whose focus is to display reasonably dense real-time graphics for one or more systems by starting collectl and directing it to send its output back to itself. Colgui requires perl-tk to build the graphics, which unfortunately is not the most efficient way to do this and so I'm officially declaring it *end-of-life* and will be removing it from the kit some time after 2014. While it seems to work reasonably well for less than 10 or 20 systems and could actually be a good starting point for someone who might like to build their own implementation, I personally haven't used it in years, especially after writing colmux. In fact, if you want to try building your own implementation colmux is a much better starting point and if you ask, I'll be happy to help get you started.

I've tried real hard to keep the quality of collectl and the utilities up to the highest quality by *eating my own dog food* and use collectl, colmux and colplot literally every day to see what's happening with all the servers in HP's Public Cloud. I just don't find colgui all that useful anymore and fear my lack of use will lead to longer term problems and really don't want to spend any time trying to support it. Since the main bulk of its code is actually shared with *colplot*, there's a good chance it will continue to work just fine, but as I said try colmux and I think you'll find it much more useful.

Multi-system Support

As already described above, both colplot and colgui support multiple systems and for looking at many types of data, particularly of a historical nature, colplot is really the only way to go. However, there are times when you want to look at what's going on (or went on in the past) on your cluster and want to see real numbers.

How many times is *top* the very first utility you run to see what's happening on your system? Colmux can do just that for an entire cluster of systems, supporing the ability to run virtually any collectl command in a *top-like* fashion, complete with sorting by any column. Sometimes you may be only interested in looking at one or two types of data as a single row of numbers, watching for changes in behaviors between lines. Colmux supports this form of output as well.

Like colgui, this utility starts collectl running on a number of systems and directs them to send their output back but rather than display graphics it can either sort the results by a column of your choice, displaying only as many lines as will fit in your terminal window OR display columns of text for that small number of user-specifiied data elements. By displaying this data in these two compact forms it makes it very easy to see a high-level view of what all systems in your cluster are doing and if any misbehave they're very easy to identify.

updated Feb 22, 2011

Colmux

NEW!Colmux Tutorial!

Introduction

Have you ever seen an nfs server getting beaten up but didn't know which of the many hundreds of clients were doing the beating? Or have you wondered if an application was leaking memory when it ran but there was no easy way to observe all the memory on all the nodes at the same time? Or how about whether or not a few disks in a large farm had slow access times and so were slowing down *all* the disks? It has always been easy to observe all of these types of behaviors with collectl one node at a time, or even plot the data after the fact with colplot. But observing cluster-wide activity in real-time has never been that easy, until now.

As its name implies, colmux is a *collectl multiplexor*, which allows one to collect data from multiple systems and treat it as a single data stream, essentially extending collectl's functionality to a set of hosts rather than a single one. Colmux has been tested on clusters of over 1000 nodes but one should also take note that this will put a heavier load on the system on which colmux is running.

Colmux runs in 2 distinct modes: *Real-Time* and *Playback*. In real-time mode, colmux actually communicates with instances of collectl running on remote systems which in turn are collecting real-time performance metrics. In playback mode colmux also communicates with a remote copy of collectl but in this case collectl is playing back a data file collected some time in the past.

Colmux can also provide its output in 2 distinct formats: *single-line* and *multi-line*. In single-line format colmux reports the multiplexed data from all systems on a single line by allowing the user to choose a small number of variables to display, based on both the display width and the number of systems. While it is possibly to handle more than a couple of dozen systems, (see the example at the bottom of this page), one rarely does so because of the screen width or their ability to read 1-point font. However it is also possible to redirect the output to a file for *off-line* viewing, via a text editor or a spreadsheet.

Colmux has been extensively tested on versions of collectl from V3.3.6 forward and there have been some additional enhancements made to V3.5.0, which is the recommended minimal version. You should first make sure all the systems of interest have the latest versions of collectl installed or at least those at V3.3.6 or newer.

Colmux also provides the ability for dynamic interaction with the keyboard arrow keys if the optional perl module *Term::ReadKey* has been installed. To see if this is the case and that colmux can find it run with -v and you should see the following:

```
colmux -v
colmux: 3.0 (Term::ReadKey: V2.30)
```

Restriction

Colmux *requires* passwordless ssh between it and all hosts it is monitoring

Using colmux

Although colmux does not have any required switches, *-command* is one of the two most important as you use it to tell collectl what switches to use when running. Colmux will then take care of multiplexing the command out to multiple instances of collectl either running them in real-time or playback mode. The other key switch, also not required but typically used, is *-address* because it identifies the remote system(s) on which to run colmux. The default addess is that of the host colmux is running on.

The inclusion of a playback filename in the collectl command instructs colmux to run in playback mode and the use of colmux's *-cols* switch tells it to produce output in *single-line* format. By using various combinations of these switches you can get colmux to run in any 4 distinct modes as shown in the following table:

	Real-Time	**Playback**
Single-line	-cols	-command "-p filename" -cols
Multi-line	*default*	-command "-p filename"

Let's discuss these 4 options separately to give a better feel for what they actually mean and when you might use them. Note that the 2 operational modes have nothing to do with the way the data is displayed and the 2 formats have nothing to do with the way the data is collected - in other words a complete separation between form and function.

Real-Time Mode

Using collectl + colmux (text files)

```
$ colmux -address m05[10-14] -command "-scdn --dskopts i -i 2" -column host -reverse
# Mon Oct 21 19:49:09 2013  Connected: 5 of 5
#     <----CPU[HYPER]-----><--------------Disks---------------><----------Network---------->
#Host cpu sys inter  ctxsw KBRead  Reads Size KBWrit Writes Size   KBIn  PktIn  KBOut  PktOut
m0510  40   8 21850  28712  1083K   8701  128     22      2   15   2511   1842    146     563
m0511  30   5 16607  23605  1081K   8596  134      0      0    0    267    541   2003    1507
m0512  42   8 21950  28289  1066K   8546  128     38      2   19   2392   1763    198     566
m0513  40   8 22608  28365  1085K   8693  128      0      0    0   2331   1718    210     580
m0514  43   9 33925  29633  1084K   8609  129     18      1   18    996   1644   5571    4188

Returned 100 row(s) in 283.22s
```

I/O maxed out at 1GB/s per node

Using collectl + colmux (parquet format)

```
$ colmux -address m05[10-14] -command "-scdn --dskopts i -i 2" -column host -reverse
# Mon Oct 21 19:59:43 2013  Connected: 5 of 5
#     <----CPU[HYPER]-----><--------------Disks---------------><----------Network---------->
#Host cpu sys inter  ctxsw KBRead  Reads Size KBWrit Writes Size   KBIn  PktIn  KBOut  PktOut
m0510  59   2 13362  50547 155648   1219  128     28      3   11     32     38     10      47
m0511  60   3 15997  51888 323776   2524  128      0      0    0     74     77     22      85
m0512  59   3 14019  51875 217086   1701  128      0      0    0     44     51     17      64
m0513  59   2 13917  51271 204420   1610  127      0      0    0     38     46     13      55
m0514  56   2 14555  52411 252984   1981  128      0      0    0     78     99     21      89

Returned 100 row(s) in 96.62s
```

Less than 1/3 the I/O thanks to Parquet format

3x query speedup

Quick Review

Impala performance checklist

Impala Performance Checklist

- Choose appropriate data types
- Leverage partition pruning
- Adopt Parquet format
- Gather table & column stats
- ~~Order JOINS optimally~~
- Validate JOIN type
- Monitor hardware resources

Automatic in v1.2.1

Test drive Impala

- Impala Community:
  - Download: http://cloudera.com/impala
  - Github: https://github.com/cloudera/impala
  - User group: impala-user on groups.cloudera.org

Cloudera VM - Hadoop, made easy. - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Cloudera VM - Hadoop, made e...

file:///home/cloudera/Documents/quick-hadoop/index.html

Google

Most Visited Cloudera Cloudera Manager Hue HDFS NameNode Hadoop JobTracker HBase Master Solr

Computer

cloudera's Home

Trash

Eclipse

CentOS

cloudera

Ask Bigger Questions

Hadoop, made easy.

Use Hadoop

Query Apache Hive and Cloudera Impala, search and customize Apache Solr, browse and manipulate files and directories in the Hadoop Distributed File System (HDFS), create and run Apache Pig scripts, visually manage Apache Oozie workflow/coordinator/bundle applications, create, submit and browse MapReduce jobs...

Hue

Administer Hadoop

Setup and monitor the health of the cluster, start and stop services like HDFS, Job Tracker, update and deploy configurations, search logs, perform audits, analyse performance graphs and metrics...

Cloudera Manager

SELECT questions FROM audience;

Thank you.


cloudera®
Ask Bigger Questions