Apache Drill – Evolution, Use cases & Roadmap

Neeraja Rentachintala, Senior Director of Product Management

12/6/2016

Topics

- Apache Drill overview
- Use cases
- Product evolution & roadmap

Industry's First Schema-free SQL engine for Big Data

APACHE DRILL

Access to *any* data type, *any* data source

- Relational
- Nested data
- Schema-less

Rapid time to insights

- Query data in-situ
- *No Schemas* required
- Easy to get started

Integration with existing tools

- *ANSI* SQL
- BI/Analytics tool integration
- User Defined Functions

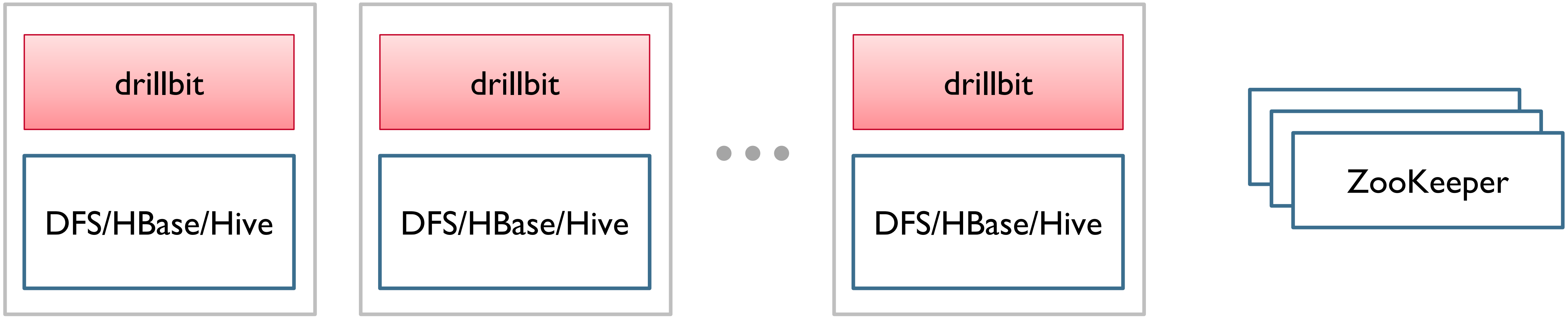
Scale in all dimensions

- TB-PB of scale
- 1000’s of users
- 1000’s of nodes

Granular security

- Authentication
- Row/column level controls
- De-centralized

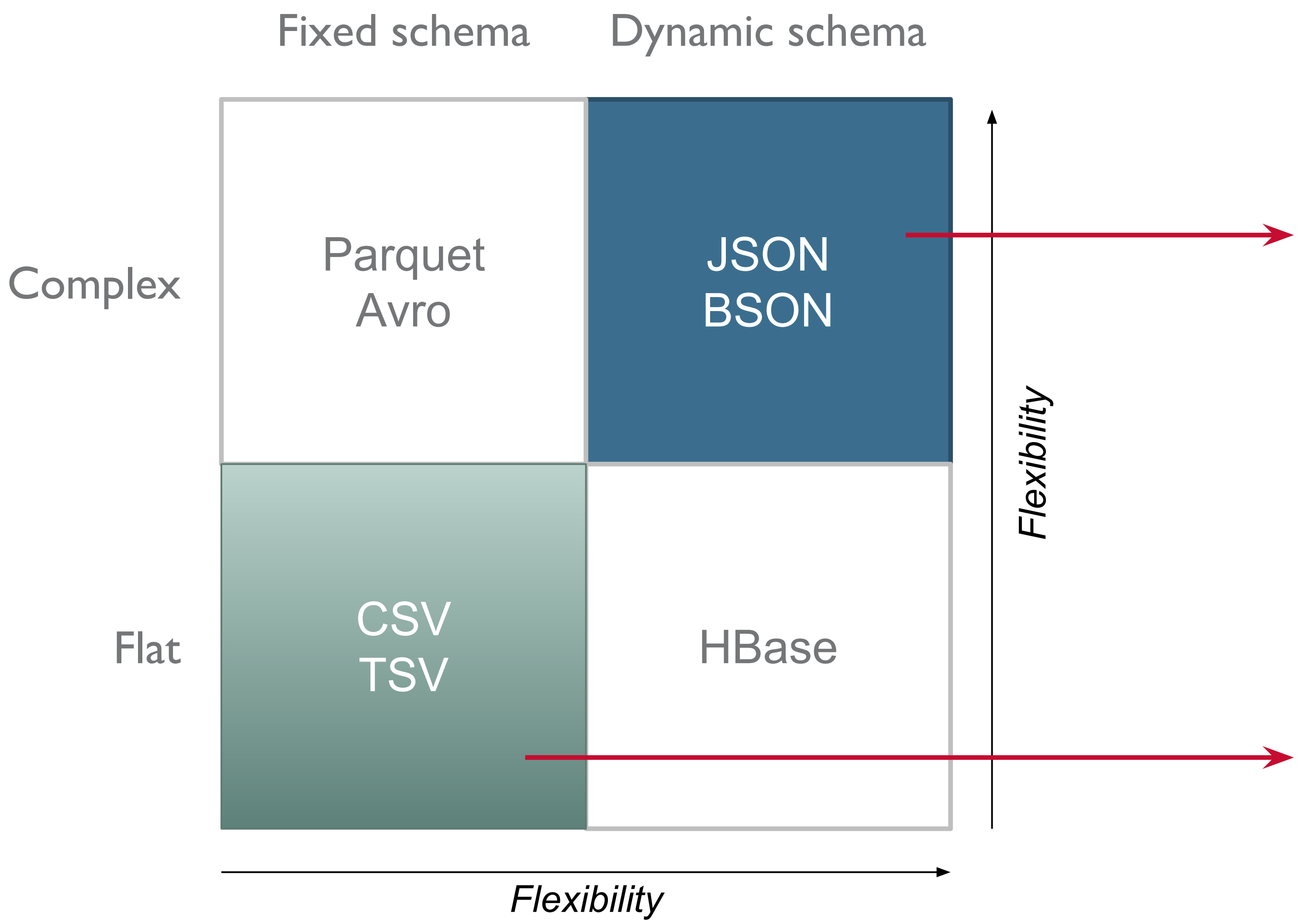
Drill is a distributed query engine



- Execution in Drill
  - Scale-out MPP
  - Hierarchical "JSON like" data model
  - Columnar and Vectorized processing
  - Optimistic execution (no MR, Spark)
  - Runtime compilation
  - Late binding
  - Extensible

- Optimization in Drill
  - Apache Calcite+ Parallel optimizations
  - Data locality awareness
  - Projection pruning
  - Filter pushdown
  - Partition pruning
  - CBO & pluggable optimization rules
  - Metadata caching

Drill's Data Model is Flexible



Apache Drill table

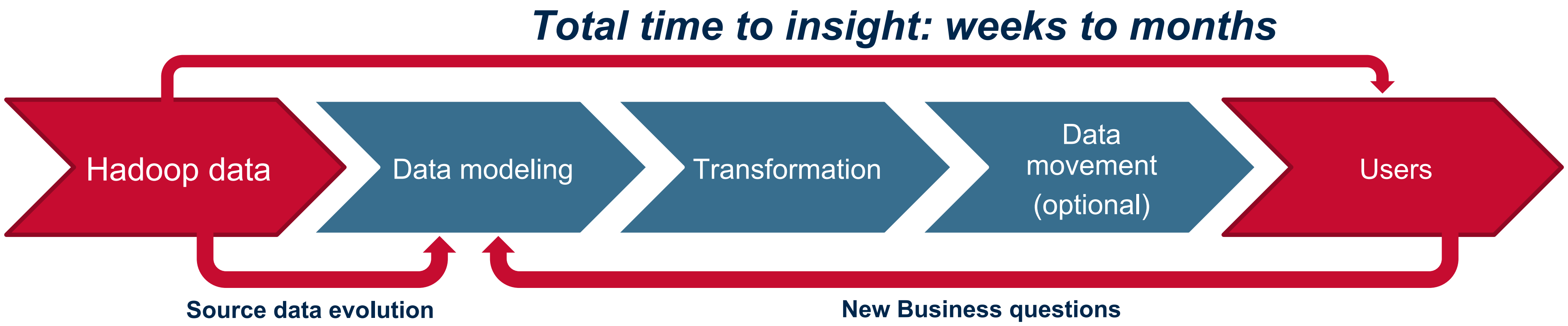
```
{
  name: {
    first: Michael,
    last: Smith
  },
  hobbies: [ski, soccer],
  district: Los Altos
}
{
  name: {
    first: Jennifer,
    last: Gates
  },
  hobbies: [sing],
  preschool: CCLC
}
```

RDBMS/SQL-on-Hadoop table

Name	Gender	Age
Michael	M	6
Jennifer	F	3

Enabling "As-It-Happens" Business with Instant Analytics

Traditional approach



Exploratory approach with Drill

Drill Enables Traditional and New types of Analytics @ Scale

Performance SLAs & Concurrency

Data exploration/Adhoc queries

Multiple BI use cases in multi-tenant organizations

User facing “Analytics as Service” applications

Example - Data exploration/Adhoc queries

Lets take a IOT race cars scenario

Producers

Sensors data

Sensor Data V1


```
{
  "_id" : "car1-1.462271134E9/102.982",
  "car" : "car1",
  "race_id" : "20160503102534",
  "racetime" : 102.982,
  "records" : [ {
    "race_id" : "20160503102534",
    "racetime" : 102.982,
    "sensors" : {
      "Distance" : 1107.563599,
      "RPM" : 1241.948975,
      "Speed" : 29.808878
    },
    "timestamp" : 1.462271134E9
  }, {
    "race_id" : "20160503102534",
    "racetime" : 103.172,
    "sensors" : {
      "Distance" : 1113.882324,
      "RPM" : 1244.506958,
      "Speed" : 29.794189
    },
    "timestamp" : 1.462271134E9
  }
```

Namespace of a Drill query

Workspace

- Sub-directory
- HBase namespace
- Hive database

Table

- Pathnames
- HBase table
- Hive table

```
SELECT * FROM dfs.demo.`sample_car_data_v1.json`
```

Storage plugin instance

- File system (Text, Parquet, JSON)
- HBase/MapR-DB
- Hive Metastore/HCatalog
- Easy API to go beyond Hadoop (MongoDB, JDBC, ES etc)

Drill has ability to discover schemas on the fly

$ tar -xvzf apache-drill-1.9.0.tar.gz

$ bin/sqlline -u jdbc:drill:zk=local

> SELECT * FROM dfs.demo.`sample_car_data_v1.json` LIMIT 1;

```
+-----+-----+---------+----------+---------+-----------+
| _id | car | race_id | racetime | records | timestamp |
+-----+-----+---------+----------+---------+-----------+
| car1-1.462271134E9/102.982 | car1 | 20160503102534 | 102.982 |
```


[{"race_id":"20160503102534","racetime":102.982,"sensors":{"Distance":1107.563599,"RPM":1241.948975,"Speed":29.808878},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":103.172,"sensors":{"Distance":1113.882324,"RPM":1244.506958,"Speed":29.794189},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":103.322,"sensors":{"Distance":1118.141846,"RPM":1249.089233,"Speed":29.776337},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":103.572,"sensors":{"Distance":1126.747314,"RPM":1243.363037,"Speed":29.756483},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":103.786,"sensors":{"Distance":1133.979004,"RPM":1240.901978,"Speed":29.732235},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":103.958,"sensors":{"Distance":1139.061035,"RPM":1243.04248,"Speed":29.699705},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":104.138,"sensors":{"Distance":1145.604126,"RPM":1241.426025,"Speed":29.664589},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":104.3,"sensors":{"Distance":1150.693237,"RPM":1237.156738,"Speed":29.629286},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":104.546,"sensors":{"Distance":1158.682251,"RPM":1235.059326,"Speed":29.567074},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":104.722,"sensors":{"Distance":1164.483276,"RPM":1233.207031,"Speed":29.520657},"timestamp":1.462271134E9},{"race_id":"20160503102534","racetime":104.878,"sensors":{"Distance":1169.551392,"RPM":1230.01062,"Speed":29.481007},"timestamp":1.462271134E9},
{"race_id":"20160503102534","racetime":112.658,"sensors":{"Distance":1408.384399,"RPM":1251.863525,"Speed":29.924427},"timestamp":1.462271134E9}] | 1.462271134E9 |


```
+-----+-----+---------+----------+---------+-----------+
```


10

Query and Manipulate complex data (Contd)

```
> SELECT
     car,
     AVG(t.records.sensors.Speed) AS`m/s`,
    (AVG(t.records.sensors.Speed)*18)/5 AS `km/h`
  FROM
      (SELECT *,FLATTEN(records) AS records
       FROM dfs.demo.`sample_car_data_v1.json`) t
  GROUP BY car;

+-------+---------------------+---------------------+
|  car  |         m/s         |        km/h         |
+-------+---------------------+---------------------+
| car1  | 48.712233453333305  | 175.36404043199988  |
| car2  | 32.86935792448977   | 118.32968852816316  |
| car3  | 46.06944574676411   | 165.8500046883508   |
| car4  | 45.020420319341206  | 162.07351314962835  |
| car5  | 44.31281895333338   | 159.52614823200014  |
| car6  | 44.57982400883707   | 160.48736643181346  |
| car7  | 45.07025832114878   | 162.2529299561356   |
| car8  | 44.91297666539465   | 161.68671599542074  |
| car9  | 43.887101968333305  | 157.9935670859999   |
+-------+---------------------+---------------------+
```

Sensor Data V2

- 5 main data points:
  - Speed (m/s)
  - RPM
  - Distance (m)
  - **Fuel**
  - **Gear**

New fields added
new sensors


```
{
  "_id" : "car3-1.462279145E9/2.342",
  "car" : "car3",
  "race_id" : "20160503123905",
  "racetime" : 2.342,
  "records" : [ {
    "race_id" : "20160503123905",
    "racetime" : 2.342,
    "sensors" : {
      "Distance" : 2011.013672,
      "Fuel" : 4.843411,
      "Gear" : 2.0,
      "RPM" : 611.679199,
      "Speed" : 22.373652
    },
    "timestamp" : 1.462279145E9
  }, {
    "race_id" : "20160503123905",
    "racetime" : 3.176,
    "sensors" : {
      "Distance" : 2031.738525,
      "Fuel" : 4.822249,
      "Gear" : 2.0,
      "RPM" : 722.421448,
      "Speed" : 26.686924
    },
    "timestamp" : 1.462279145E9
  }
```

Query on new fields added in V2 instantly

```
> SELECT
      car,
      t.records.sensors.gear AS gear,
      AVG(t.records.sensors.Speed) AS `m/s`,
     (AVG(t.records.sensors.Speed)*18)/5 AS `km/h`
  FROM
      (SELECT *,  FLATTEN(records) AS records
       FROM dfs.demo.`sample_car_data_v2.json`) t
  GROUP BY car , t.records.sensors.gear
  ORDER BY car , t.records.sensors.gear LIMIT 10;

+-------+---------+--------------------+--------------------+
|  car  | gear|         m/s          |         km/h         |
+-------+---------+--------------------+--------------------+
| car3  | 1.0     | 11.897254207792207 | 42.830115148051945 |
| car3  | 2.0     | 27.522497517799366 | 99.08099106407772  |
| car3  | 3.0     | 37.204457530487815 | 133.93604710975615 |
| car3  | 4.0     | 49.01672581017499  | 176.46021291662996 |
| car3  | 5.0     | 61.91693234567113  | 222.90095644441607 |
| car3  | 6.0     | 70.04746468410457  | 252.17087286277646 |
| car4  | 1.0     | 10.57082569811321  | 38.05497251320755  |
| car4  | 2.0     | 24.364864507246377 | 87.71351222608696  |
| car4  | 3.0     | 36.6934715200831   | 132.09649747229918 |
| car4  | 4.0     | 50.961303641552576 | 183.46069310958927 |
+-------+---------+--------------------+--------------------+
```

Drill provides ANSI SQL capabilities

```
> WITH X AS (
                  SELECT car,
                  t.race_id AS race,
                  t.records.sensors.speed AS speed ,
                  RANK() OVER (PARTITION BY t.car, t.race_id
                            ORDER BY  t.records.sensors.speed DESC) AS carracerank
                  FROM (SELECT *, FLATTEN(records) AS records FROM
                       dfs.demo.`sample_car_data_v1.json`) t)
         SELECT X.Car, X.race, X.speed FROM X
         WHERE X.carracerank =1 LIMIT 15;

+-------+---------------+------------+
|  Car  |      race      |   speed    |
+-------+---------------+------------+
| car1  | 20160503102534 | 83.075401  |
| car2  | 20160503102534 | 48.842308  |
| car2  | 20160503102837 | 45.507034  |
| car3  | 20160503103501 | 67.85479   |
| car3  | 20160503103937 | 67.804237  |
| car3  | 20160503104300 | 67.866081  |
| car4  | 20160503102534 | 68.376411  |
| car4  | 20160503102837 | 63.95813   |
| car4  | 20160503103501 | 68.387604  |
| car4  | 20160503103937 | 68.398613  |
| car4  | 20160503104300 | 68.417183  |
| car5  | 20160503102534 | 68.214432  |
| car6  | 20160503102534 | 68.877663  |
| car6  | 20160503102837 | 64.2967    |
| car6  | 20160503103501 | 70.95742   |
+-------+---------------+------------+
```

Drill product evolved significantly since GA

- Just released – Apache Drill 1.9
- Key Product highlights
  - Enhanced Parquet Performance (Parquet filter pushdown, Improved Scans with Async Parquet Reader)
  - Flexible & Dynamic UDFs
  - Null equality joins support
  - Efficient metadata queries

Drill 1.0 GA
- Drill GA

Drill 1.1
- Automatic Partitioning for Parquet Files

SQL Window fu

SUM
- -Ranki
CUME_
DENSE_
PERCEN
RANK a
ROW_N
- Hive im
- SQL Ur
- Complex data enhancements· and more

Drill 1.2
- Native parquet reader for Hive tables

Enhanced Hive

- Drop table syntax
- Metadata caching
- Security support for web UI
- INT 96 data type

Drill 1.3/1.4
- Improved Tableau experience with faster Limit 0

Query Performance/Scal

usage and performance
- New & improved JDBC driver

Drill 1.5/1.6
- Enhanced Stability & scale
  - New memory

Drill on MapR-DB
es

of
g in
y
d
t N
parquet metadata caching
- Limit 0 performance
- Enhanced SQL Window function frame syntax

Drill 1.71/.8
- Drill on YARN integration

Enterprise manageability

- Monitoring via JMX
- Hive CHAR data type support
- Partition pruning enhancements
- Ability to return file names as part of queries

Simplified deployment with YARN

- Drill as a long running application in YARN
- Key features
  - Client tool to launch Drill as YARN application
  - New Drill application master (AM)
  - CPU & memory controls
  - Add/remove nodes to cluster
  - Multiple Drill clusters

Apache Drill | Configuration | Drillbits | Manage | History

YARN Application Master – Drill-on-YARN

Drill Cluster Status

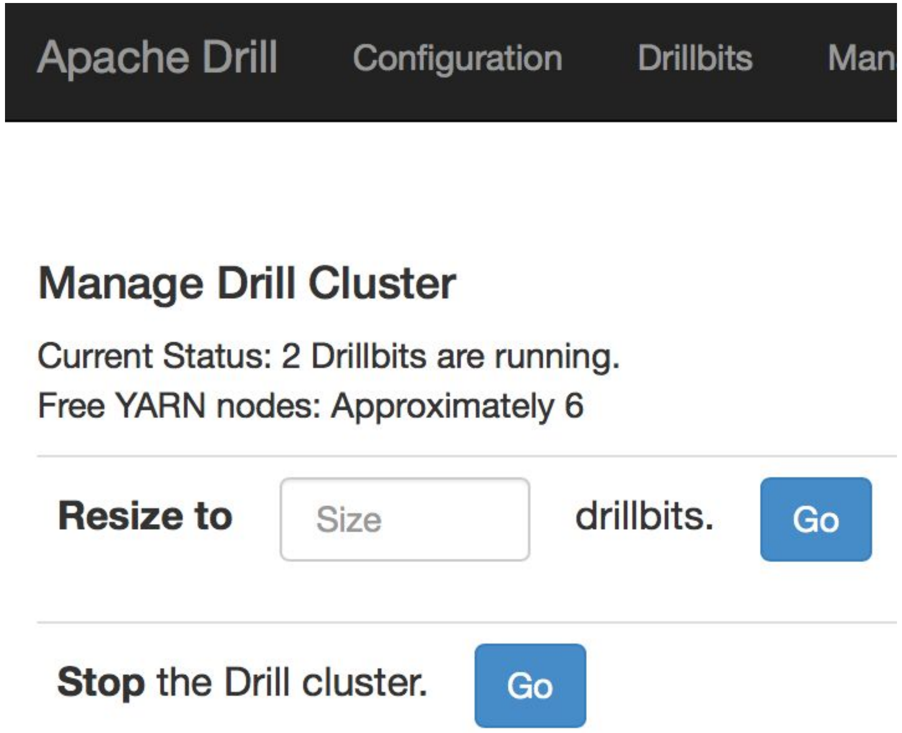
YARN Application ID:	application_1465323848842_0024
YARN Resource Manager:	doy1.mapr.lab
YARN Node Manager for AM:	doy8.mapr.lab | App info
ZooKeeper:	10.10.102.25:5181/drill/drillbits1
State:	LIVE
Target Drillbit Count:	8
Live Drillbit Count:	6
Total Drill Virtual Cores:	12
Total Drill Memory (MB):	36,864
Yarn Node Count:	7
Available Yarn Virtual Cores:	14
Available Yarn Memory (MB):	8,192

Pool	Type	Target Drillbit Count	Total Drillbits	Live Drillbits	Memory per Drillbit (MB)	VCores per Drillbit
pool-1	basic	8	8	6	6,144	2

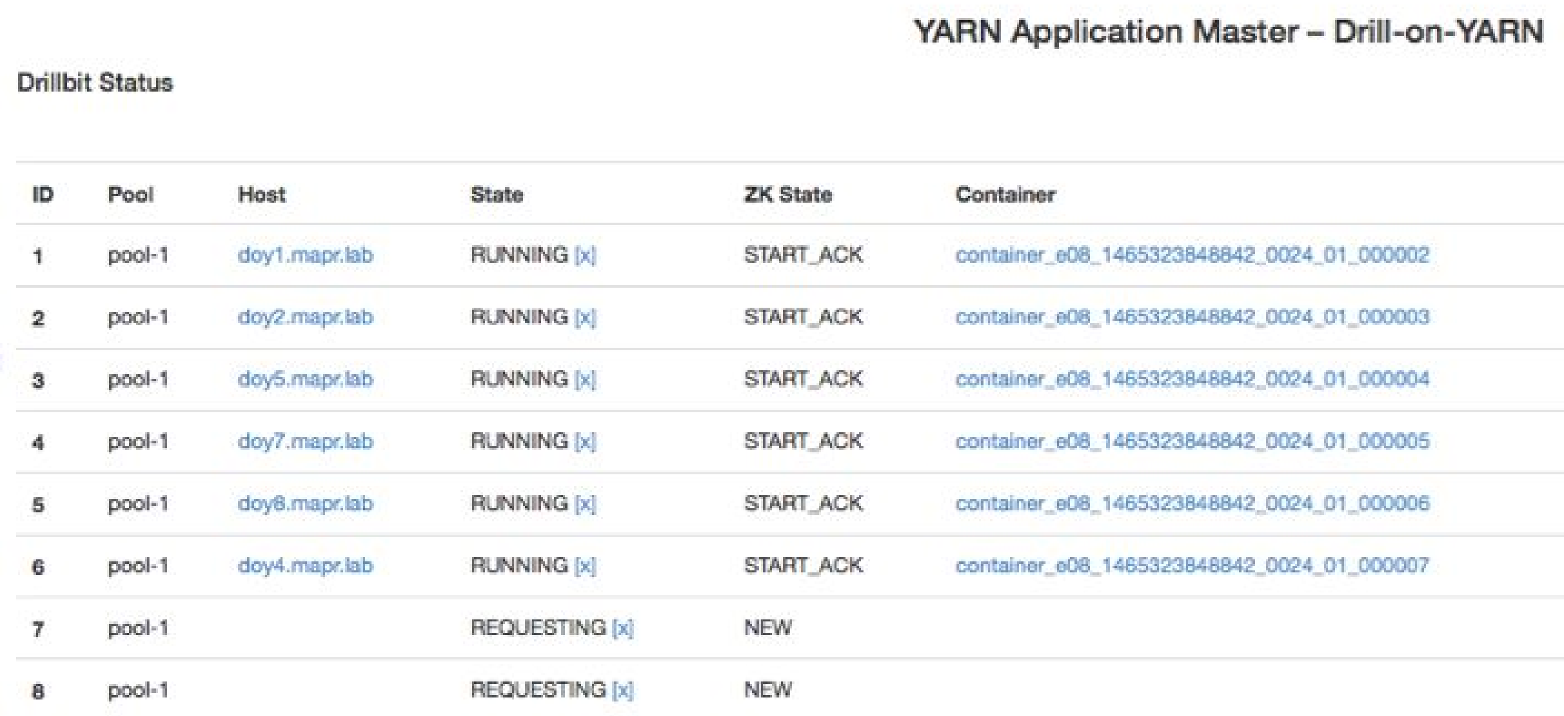
Drill configuration w/YARN

Drill on YARN (contd)

Drill cluster management under YARN



Drill cluster status under YARN



YARN Application Master – Drill-on-YARN

Drillbit Status

ID	Pool	Host	State	ZK State	Container
1	pool-1	doy1.mapr.lab	RUNNING [x]	START_ACK	container_e08_1465323848842_0024_01_000002
2	pool-1	doy2.mapr.lab	RUNNING [x]	START_ACK	container_e08_1465323848842_0024_01_000003
3	pool-1	doy5.mapr.lab	RUNNING [x]	START_ACK	container_e08_1465323848842_0024_01_000004
4	pool-1	doy7.mapr.lab	RUNNING [x]	START_ACK	container_e08_1465323848842_0024_01_000005
5	pool-1	doy6.mapr.lab	RUNNING [x]	START_ACK	container_e08_1465323848842_0024_01_000006
6	pool-1	doy4.mapr.lab	RUNNING [x]	START_ACK	container_e08_1465323848842_0024_01_000007
7	pool-1		REQUESTING [x]	NEW	
8	pool-1		REQUESTING [x]	NEW	

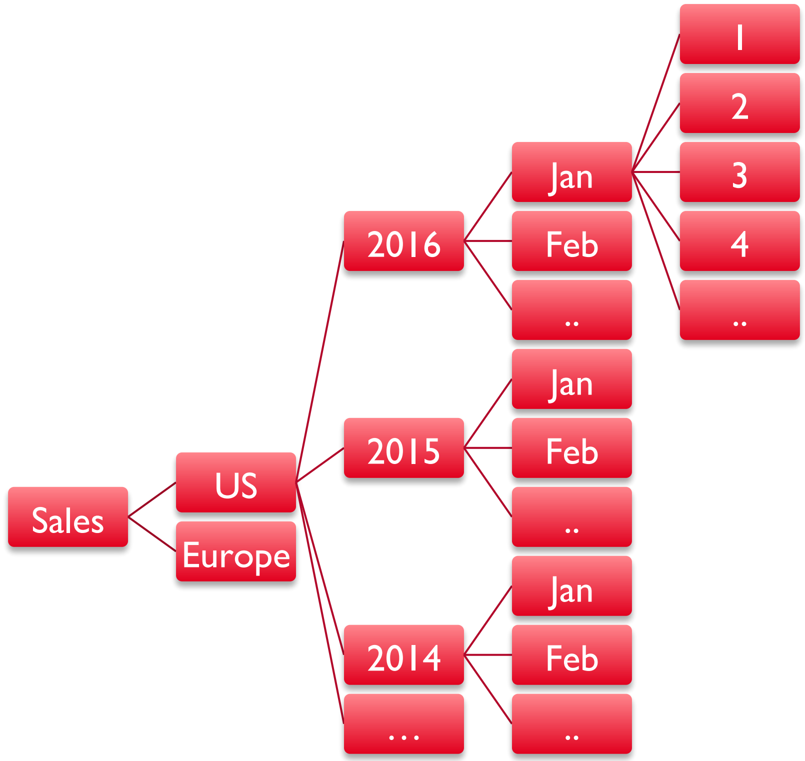
New JMX based monitoring

- New JMX based metrics Drill Web Console or MapR Spyglass (Beta) or a remote JMX monitoring tool, such as Jconsole
- Various system and query metrics
  - drill.queries.running
  - drill.queries.completed
  - heap.used
  - direct.used
  - waiting.count …

Drill metrics dashboard with Spyglass

Improved query performance with Partition pruning

- Partition pruning allows a query engine to determine and retrieve the smallest needed dataset to answer a given query
- Data can be partitioned
  - At the time of ingestion into the cluster
  - As part of ETL via Hive or Spark or other batch processing tools
  - Drill support CTAS with PARTITION BY clause
- Drill does partition pruning for queries on partitioned Hive tables as well as file system queries



Select * from Sales
Where dir0='US' and dir1 ='2015'

Metadata Caches to Speed up Query Planning

- Helps reduce query planning time significantly when working with large # of Parquet files (thousands to millions)

- Highly optimized cache with the key metadata from parquet files
  - Column names, data types, nullability, row group size…

- Recursive cache creation at root level or selectively for specific directories or files
  - Ex: REFRESH TABLE METADATA dfs.tmp.BusinessParquet;

- Metadata caching is better suited for large amounts of data with moderate rate of change

Improved Query performance with Parquet Filter pushdown

- Applies during planning time
- Evaluates filter condition before the scan
- Planner evaluates filter conditions and checks if a Parquet row group can be eliminated
- Requires Parquet files to have min/max statistics

Example

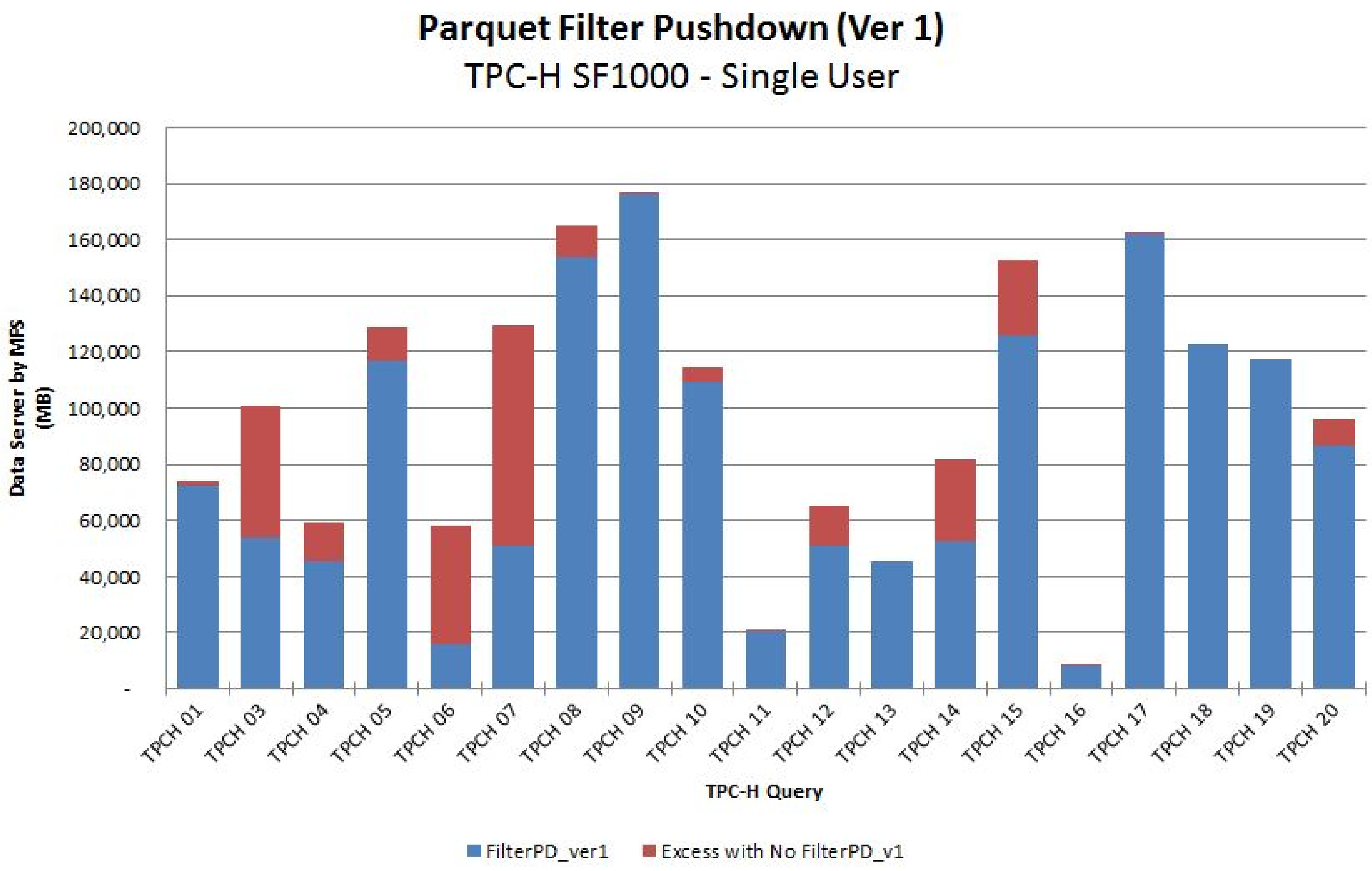
*SELECT * from table_t1 WHERE date_column between date '2016-01-01' and date '2016-01-31'*

Row group 1 : date_column : min = 2015-01-01 max = 2015-12-31

Row group 2 : date_column : min = 2016-01-01 max = 2016-12-31

Only row group 2 will be scanned

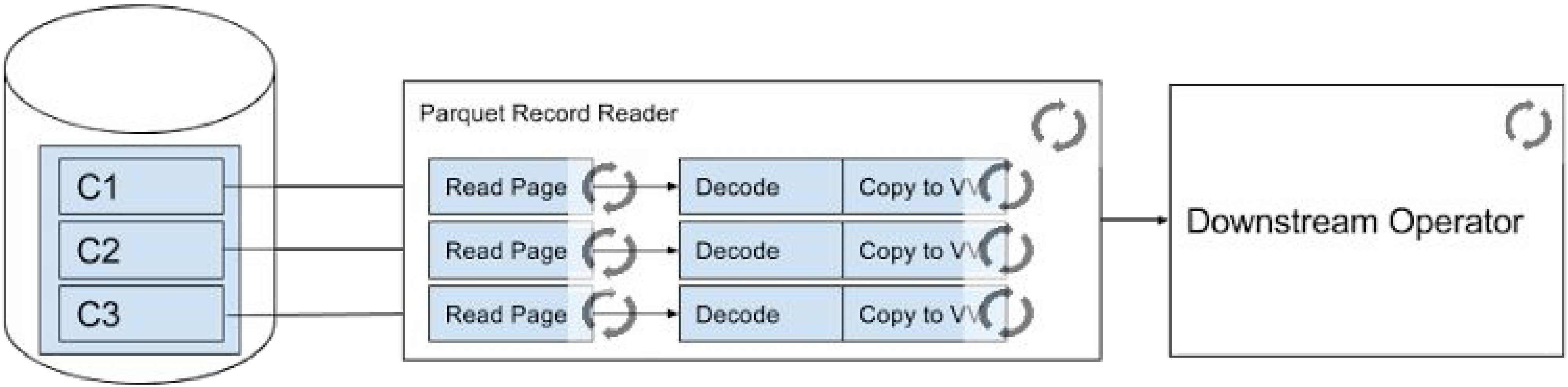
Parquet Filter pushdown



TPCH Queries	Selectivity	**Without FltrPD (MB)**	**With FltrPD (MB)**	I/O Reduction
TPCH 06	15%	5,779	1,707	70%
TPCH 07	30%	12,395	5,188	58%
TPCH 14	1%	7,915	5,254	34%
TPCH 20	15%	9,174	8,333	9%

Enhanced query performance w/Asynchronous Parquet reader

- High performance queries for scan intensive analytics (~33% I/O reduction)
- Parquet reader improvements include
  - Buffered reads
  - Parallel reads from file system
  - Parallel decompression and decoding
  - Reading and decoding is pipelined



How all these fit together?

Hive/Spark on File System data (Batch/ETL workloads)

→

Drill + FS w/raw file formats (ex: Text, JSON..) (Data exploration/ Adhoc queries)

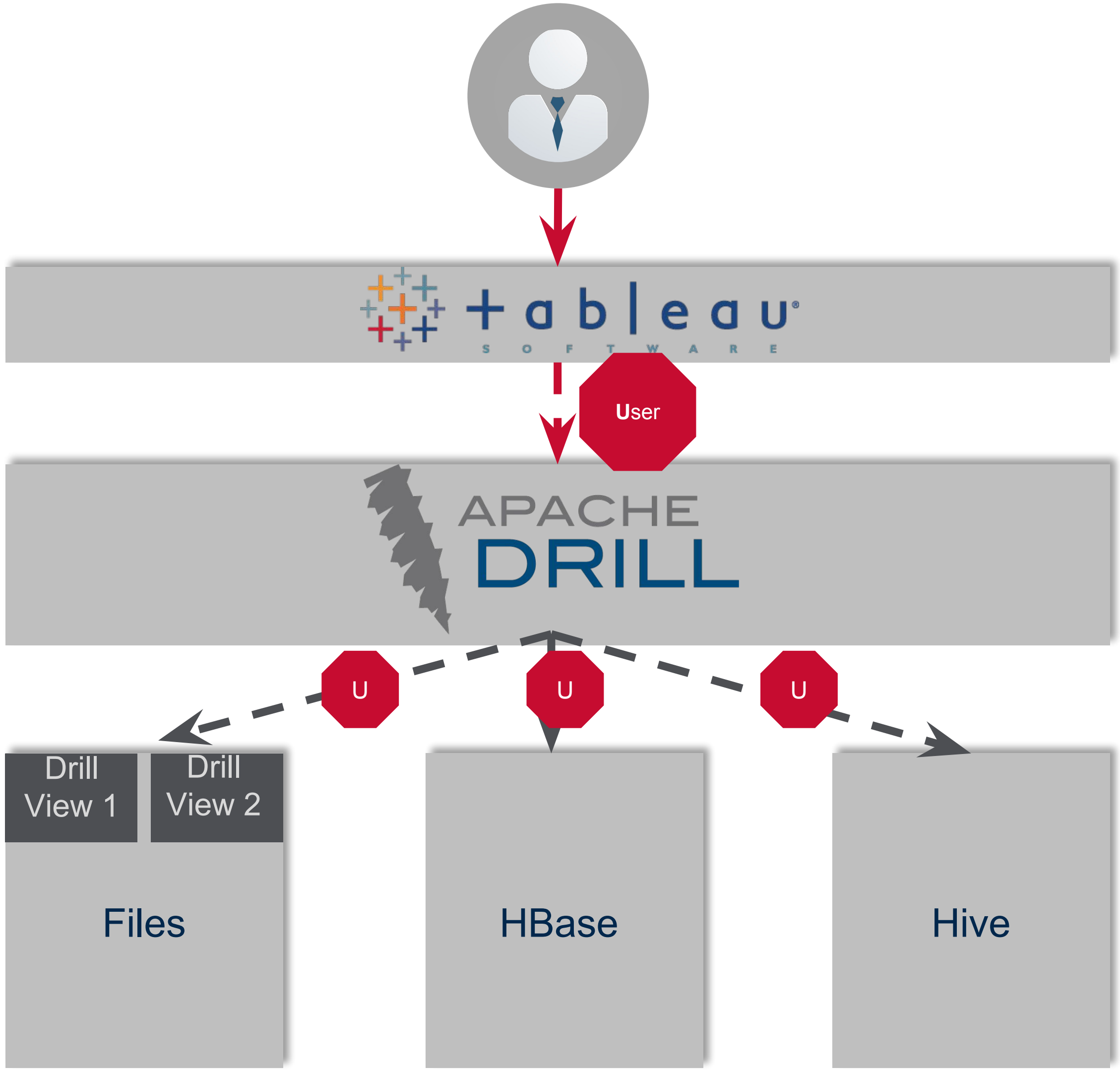
→

Drill + FS w/Parquet (BI/Deep analytics historical data)

→

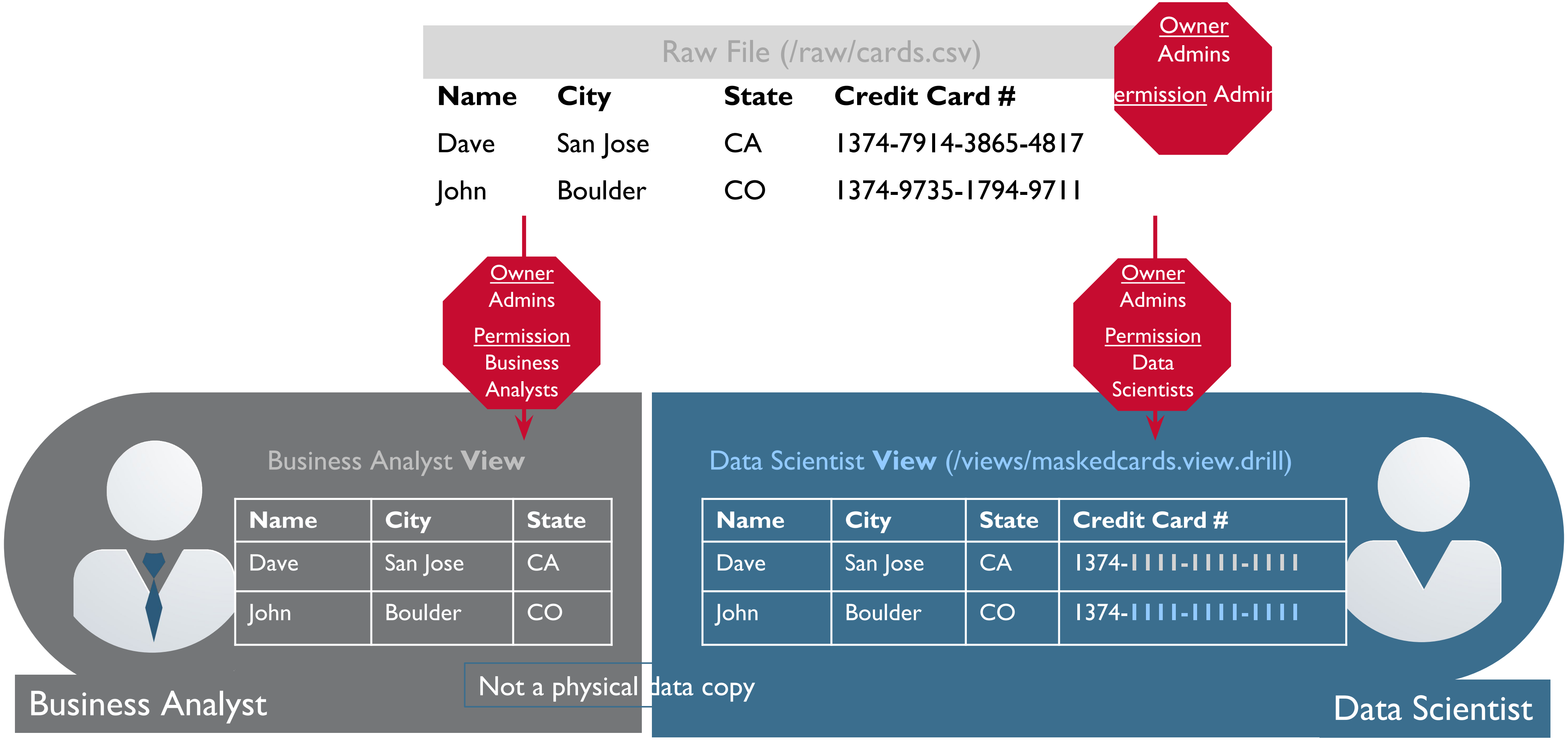
Drill + MapR-DB/HBase (Operational analytics)

End to End security from BI tools to Hadoop

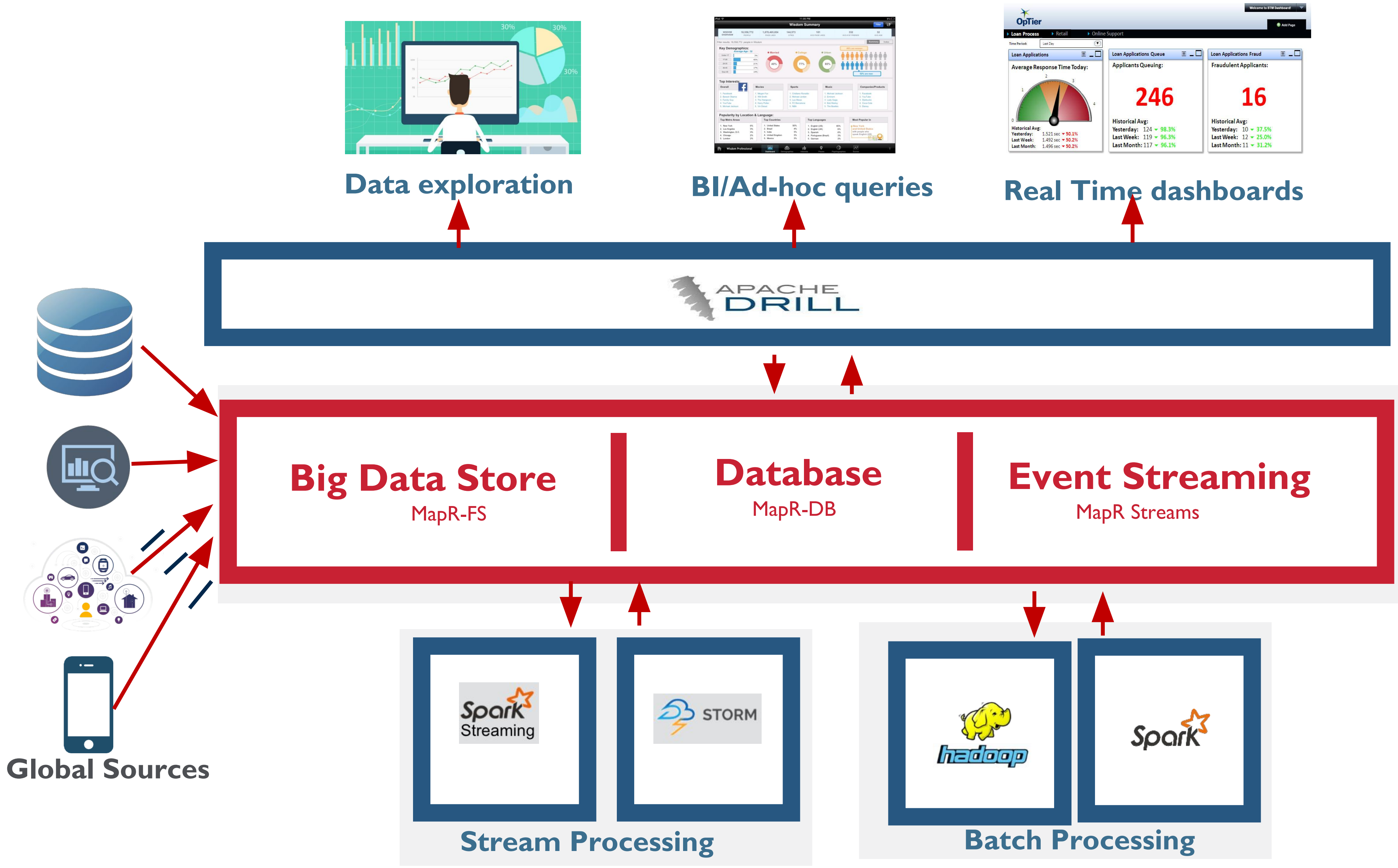


- Security support for JDBC/ODBC drivers & Drill Web UI
- Standard PAM based Authentication
- User impersonation & inbound impersonation
- Fine-grained row and column level access control with Drill Views – no centralized security repository required
- Kerberos & MapR-SASL support between client to Drillbit (Future)
- Encryption support (Future)
  - Between client-Drillbit
  - Between Drillbits

Granular security permissions through Drill views



Evolving towards Unified SQL access layer for MapR platform



- Queries across Files, Tables and Streams
- Real time/Operational analytics
- Schema-less JSON flexibility
- Distributed in-memory SQL engine for high performance at Scale
- Analytics from familiar BI/SQL tools

Drill Best Practices on the MapR Converge Community

Apache Drill Best Practices from the MapR Drill Team

Document created by **Zelaine Fong** on Mar 23, 2016 • Last modified by **Andries Engelbrecht** on Jul 20, 2016

Data Layout and Deployment

Schema Considerations

- Does Drill have a limit on # of columns?
- Are there datatypes in Drill that should be favored vs avoided?

Storage Formats

- What is the preferred storage format for Drill?

Parquet Best Practices

- What is the preferred compression type when using Parquet with Drill?
- What is the recommended parquet block size (when running on MapR-FS) for Drill?
- Can Parquet files created by other tools (e.g., Hive, Spark) be read by Drill?

Data Partitioning

- How can I partition data in Drill?
- How do I determine the right partitioning strategy?
- Are there situations where partition pruning cannot be done in Drill?
- How do I know if partition pruning has been applied to my Drill query?

Query Tuning and Performance

LIMIT 0 Queries

- How can I speed up my LIMIT 0 queries on Drill?
- How do I verify that my Limit 0 Drill queries are benefiting from the optimized code paths?

Hao Zhu
Apr 14, 2016 11:03 AM

★ Correct Answer

For Drill query like the following:

*select * from table where col in (1,2,3,...);*

The in-list evaluation is done in a sequential manner. Therefore, if you increase the number of elements in the in-list, the performance of the query could degrade linearly.
If the number of in-list elements contains at least 20 elements, Drill can optimize this query to use an in-memory hash table to store the in-list elements, and perform a table join instead. This optimization can greatly improve the performance.

For example, this query will not benefit from the optimization because the IN list has only 19 elements.

select count(1) from t where cast(col as int) in (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19);

Whereas, this one will.

select count(1) from t where cast(col as int) in (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20);

To improve the performance of a query with many, but less than 20 in-list elements, you can add duplicate in-list elements so the total number of elements reaches 20.

For example, change the where condition from:
"where col in (1,2,3,4,5,6,7,8,9)"
To:
"where col in (1,2,3,4,5,6,7,8,9,1,1,1,1,1,1,1,1,1,1,1)"

More details are in the following link:
http://www.openkb.info/2015/08/how-to-improve-performance-of-drill.html

See the reply in context

https://community.mapr.com/docs/DOC-1497

Recommendations On Trying and Using Drill

New to Drill?

- Get started with Free MapR On Demand training
- Test Drive Drill on cloud with AWS
- Learn how to use Drill with Hadoop using MapR sandbox

Ready to play with your data?

- Try out Apache Drill in 10 mins guide on your desktop
- Download Drill for your cluster and start exploration
- Comprehensive tutorials and documentation available