Apache Spark for RDBMS Practitioners: How I Learned to Stop Worrying and Love to Scale

Luca Canali, CERN

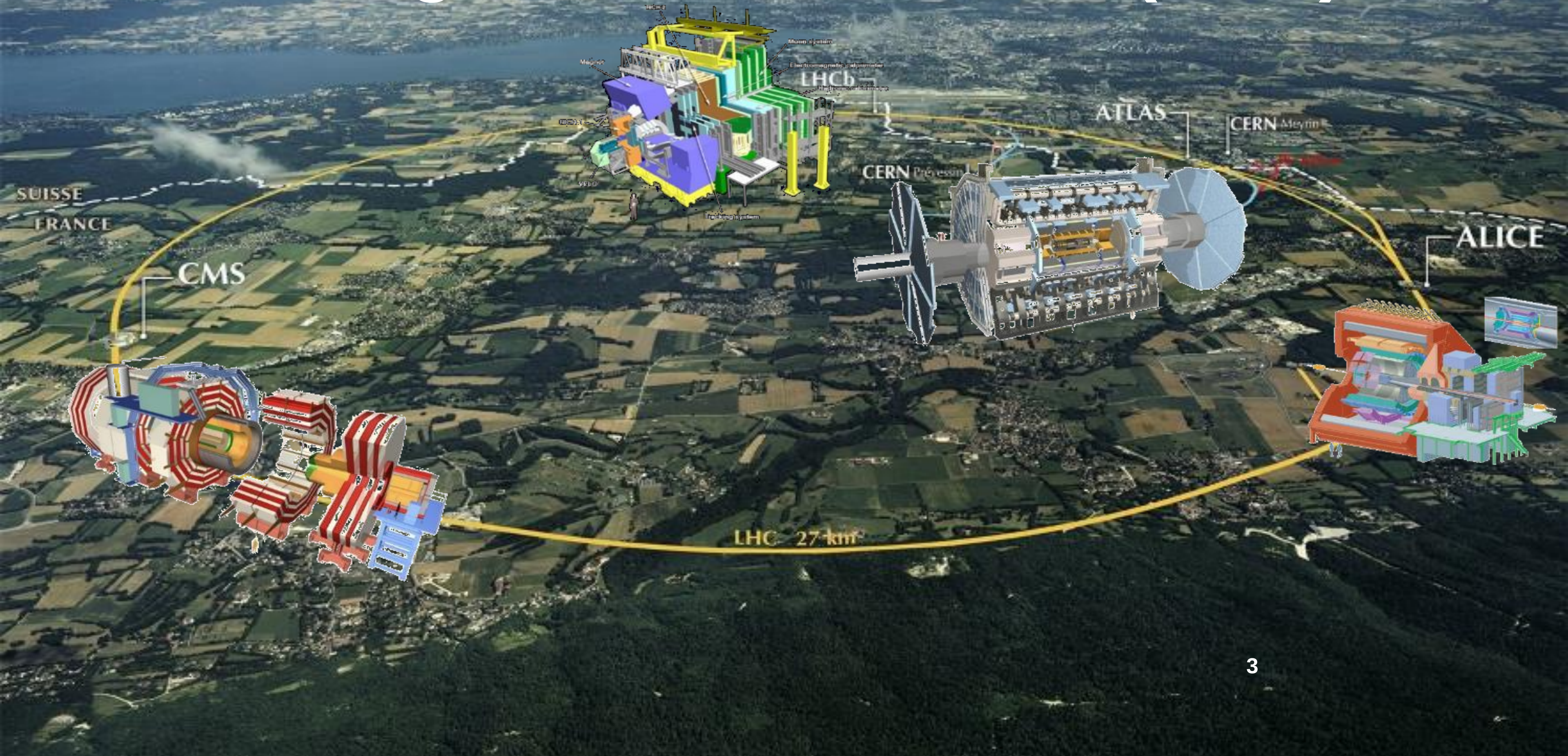
#SAISDev11

About Luca

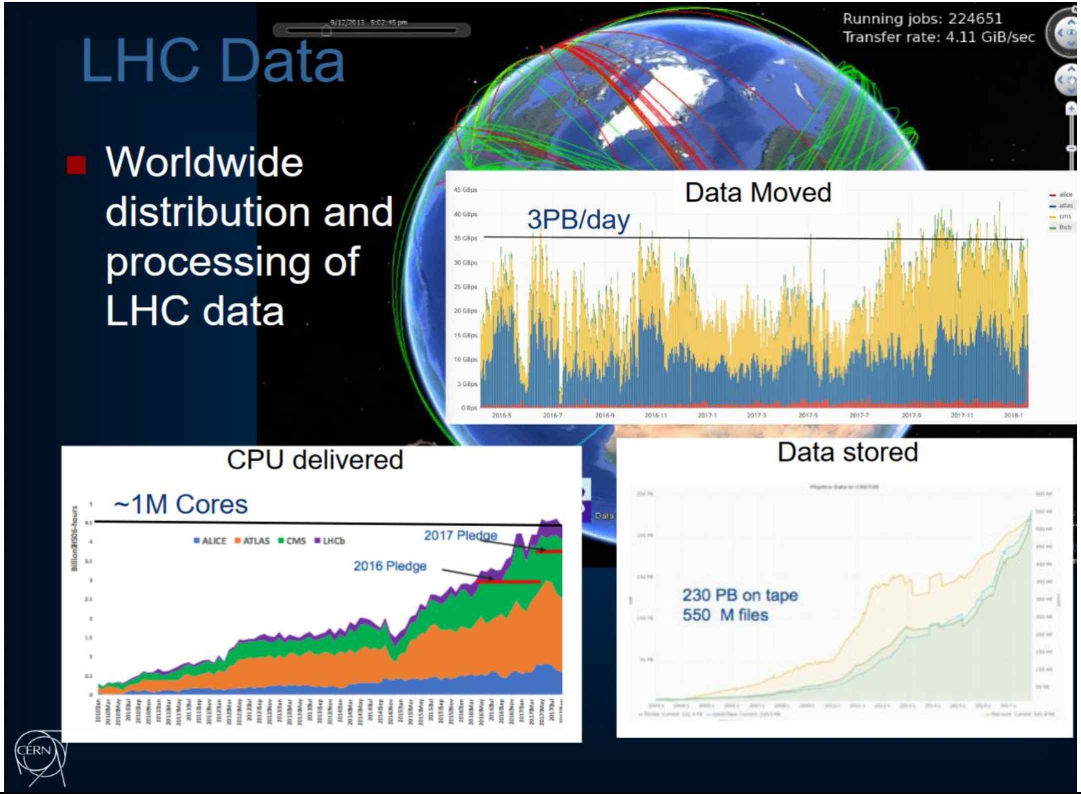
- Data Engineer and team lead at CERN
  - Hadoop and Spark service, database services
  - 18+ years of experience with data(base) services
  - Performance, architecture, tools, internals
- Sharing and community
  - Blog, notes, tools, contributions to Apache Spark

@LucaCanaliDB – http://cern.ch/canali

The Large Hadron Collider (LHC)



LHC and Data



LHC data processing teams (WLCG) have built custom solutions able to operate at very large scale (data scale and world-wide operations)

Overview of Data at CERN

- Physics Data ~ 300 PB -> EB
- "Big Data" on Spark and Hadoop ~10 PB
  - Analytics for accelerator controls and logging
  - Monitoring use cases, this includes use of Spark streaming
  - Analytics on aggregated logs
  - Explorations on the use of Spark for high energy physics
- Relational databases ~ 2 PB
  - Control systems, metadata for physics data processing, administration and general-purpose

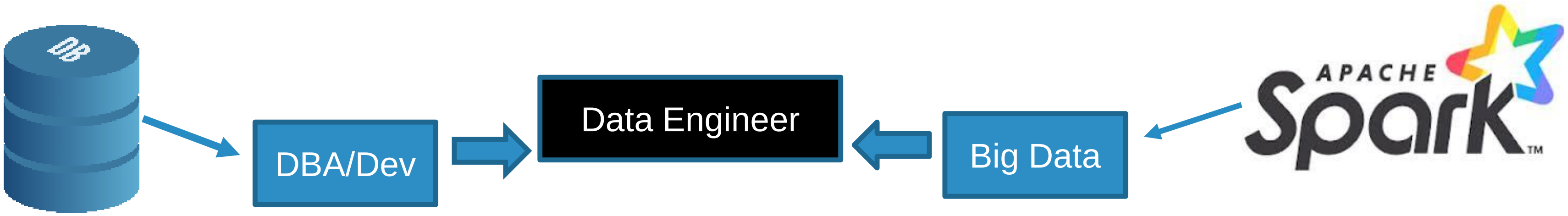
Apache Spark @ CERN

- Spark on YARN/HDFS
  - In total ~1850 physical cores and 15 PB capacity
- Spark on Kubernetes
  - Deployed on CERN cloud using OpenStack
  - See also, at this conference: “Experience of Running Spark on Kubernetes on OpenStack for High Energy Physics Workloads”

Cluster Name	Configuration	Software Version
Accelerator logging	20 nodes (Cores 480, Mem - 8 TB, Storage – 5 PB, 96GB in SSD)	Spark 2.2.2 – 2.3.1
General Purpose	48 nodes (Cores – 892,Mem – 7.5TB,Storage – 6 PB)	Spark 2.2.2 – 2.3.1
Development cluster	14 nodes (Cores – 196,Mem – 768GB,Storage – 2.15 PB)	Spark 2.2.2 – 2.3.1
ATLAS Event Index	18 nodes (Cores – 288,Mem – 912GB,Storage – 1.29 PB)	Spark 2.2.2 – 2.3.1

Motivations/Outline of the Presentation

- Report experience of using Apache Spark at CERN DB group
  - Value of Spark for data coming from DBs
- Highlights of learning experience



Goals

- Entice more DBAs and RDBMS Devs into entering Spark community
- Focus on added value of scalable platforms and experience on RDBMS/data practices
  - SQL/DataFrame API very valuable for many data workloads
  - Performance instrumentation of the code is key: use metric and tools beyond simple time measurement

Spark, Swiss Army Knife of Big Data

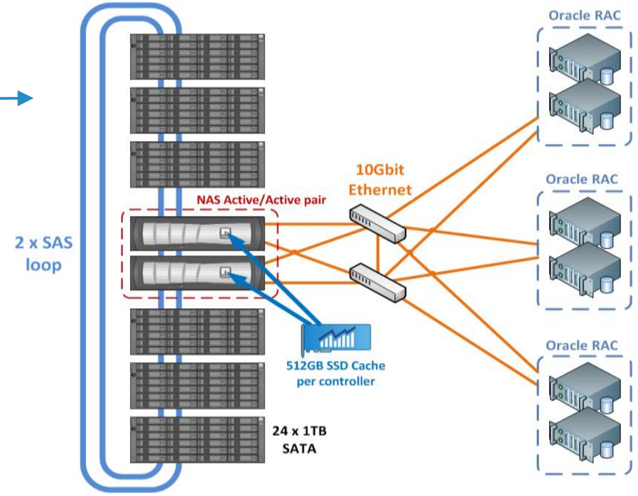
One tool, many uses

- Data extraction and manipulation
- Large ecosystem of data sources
- Engine for distributed computing
- Runs SQL
- Streaming
- Machine Learning
- GraphX
- ..

Image Credit: Vector Pocket Knife from Clipart.me

Problem We Want to Solve

- Running reports on relational DATA
  - DB optimized for transactions and online
  - Reports can overload storage system
- RDBMS specialized
  - for random I/O, CPU, storage and memory
  - HW and SW are optimized at a premium for performance and availability



Offload from RDBMS

- Building hybrid transactional and reporting systems is expensive
  - Solution: offload to a system optimized for capacity
  - In our case: move data to Hadoop + SQL-based engine

Spark Read from RDBMS via JDBC

```
val df = spark.read.format("jdbc")
.option("url","jdbc:oracle:thin:@server:port/service")
.option("driver", "oracle.jdbc.driver.OracleDriver")
.option("dbtable", "DBTABLE")
.option("user", "XX").option("password", "YY")
.option("fetchsize",10000)
.option("sessionInitStatement",preambleSQL)
.load()
```

Optional DB-specific Optimizations
SPARK-21519

option("sessionInitStatement", """BEGIN execute immediate 'alter session set "_serial_direct_read"=true'; END;""")

Challenges Reading from RDMBs

- Important amount of CPU consumed by serialization-deserialization transferring to/from RDMS data format
  - Particularly important for tables with short rows
- Need to take care of data partitioning, parallelism of the data extraction queries, etc
  - A few operational challenges
  - Configure parallelism to run at speed
  - But also be careful not to overload the source when reading

Apache Sqoop to Copy Data

- Scoop is optimized to read from RDBMS
- Sqoop connector for Oracle DBs has several key optimizations
- We use it for incremental refresh of data from production RDBMS to HDFS “data lake”
  - Typically we copy latest partition of the data
  - Users run reporting jobs on offloaded data from YARN/HDFS

Sqoop to Copy Data (from Oracle)

Sqoop has a rich choice of options to customize the data transfer

```
sqoop import \
--connect jdbc:oracle:thin:@server.cern.ch:port/service
--username ..
-P \
-Doraoop.chunk.method=ROWID
--direct \
--fetch-size 10000 \
--num-mappers XX \
--target-dir XXXX/mySqoopDestDir \
--table TABLESNAME \
--map-column-java FILEID=Integer,JOBID=Integer,CREATIONDATE=String \
--compress --compression-codec snappy \
--as-parquetfile
```

Oracle-specific optimizations

Streaming and ETL

- Besides traditional ETL also streaming important and used by production use cases

Streaming data into Apache Kafka

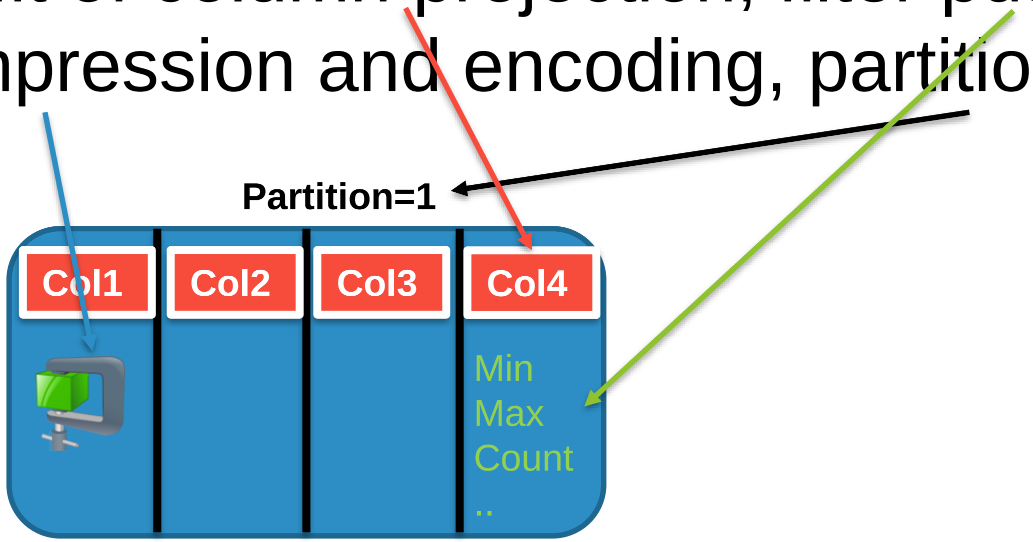
Spark and Data Formats

- Data formats are key for performance
- Columnar formats are a big boost for many reporting use cases
  - Parquet and ORC
  - Column pruning
  - Easier compression/encoding
- Partitioning and bucketing also important

Data Access Optimizations

Use Apache Parquet or ORC

- Profit of column projection, filter push down, compression and encoding, partition pruning



Parquet Metadata Exploration

Tools for Parquet metadata exploration, useful for learning and troubleshooting

Row count   Size in bytes   Column chunk size: compressed / uncompressed / ratio

```
row group 1:          RC:2840100 TS:154947976 OFFSET:4
--------------------------------------------------------------------------------
ss_sold_time_sk:        INT32 SNAPPY DO:0 FPO:4 SZ:2419027/588623[illegible]/2.43 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_item_sk:             INT32 SNAPPY DO:0 FPO:2419031 SZ:504085[illegible]/5040503/1.00 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_customer_sk:         INT32 SNAPPY DO:0 FPO:7459884 SZ:4200678/7168827/1.71 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_cdemo_sk:            INT32 SNAPPY DO:0 FPO:11660562 SZ:4179788/7133328/1.71 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_hdemo_sk:            INT32 SNAPPY DO:0 FPO:15840350 SZ:3948359/4755600/1.20 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_addr_sk:             INT32 SNAPPY DO:0 FPO:19788709 SZ:4199290/7144061/1.70 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_store_sk:            INT32 SNAPPY DO:0 FPO:23987999 SZ:1405046/2279438/1.62 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_promo_sk:            INT32 SNAPPY DO:0 FPO:25393045 SZ:3264270/3341421/1.02 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_ticket_number:       INT32 SNAPPY DO:0 FPO:28657315 SZ:3748010/7125883/1.90 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_quantity:            INT32 SNAPPY DO:0 FPO:32405325 SZ:2569908/2646790/1.03 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_wholesale_cost:      INT32 SNAPPY DO:0 FPO:34975233 SZ:5036313/5113188/1.02 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_list_price:          INT32 SNAPPY DO:0 FPO:40011546 SZ:5422519/5500119/1.01 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_sales_price:         INT32 SNAPPY DO:0 FPO:45434065 SZ:5386171/5463066/1.01 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_ext_discount_amt:    INT32 SNAPPY DO:0 FPO:50820236 SZ:3721043/6098549/1.64 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_ext_sales_price:     INT32 SNAPPY DO:0 FPO:54541279 SZ:11202990/11309483/1.01 VC:2840100 ENC:PLAIN,BIT_PACKED,RLE
ss_ext_wholesale_cost:  INT32 SNAPPY DO:0 FPO:65744269 SZ:11235173/11309896/1.01 VC:2840100 ENC:PLAIN,BIT_PACKED,RLE
ss_ext_list_price:      INT32 SNAPPY DO:0 FPO:76979442 SZ:11232056/11307593/1.01 VC:2840100 ENC:PLAIN,BIT_PACKED,RLE
ss_ext_tax:             INT32 SNAPPY DO:0 FPO:88211498 SZ:6226701/6299755/1.01 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_coupon_amt:          INT32 SNAPPY DO:0 FPO:94438199 SZ:3721043/6098549/1.64 VC:2840100 ENC:PLAIN_DICTIONARY,BIT_PACKED,RLE
ss_net_paid:            INT32 SNAPPY DO:0 FPO:98159242 SZ:11192504/11308687/1.01 VC:2840100 ENC:PLAIN,BIT_PACKED,RLE
ss_net_paid_inc_tax:    INT32 SNAPPY DO:0 FPO:109351746 SZ:11219364/11308178/1.01 VC:2840100 ENC:PLAIN,BIT_PACKED,RLE
ss_net_profit:          INT32 SNAPPY DO:0 FPO:120571110 SZ:11235592/11308829/1.01 VC:2840100 ENC:PLAIN,BIT_PACKED,RLE
```

Useful tools:
parquet-tools
parquet-reader

Info at:
https://github.com/LucaCanali/Miscellaneous/blob/master/Spark_Notes/Tools_Parquet_Diagnostics.md

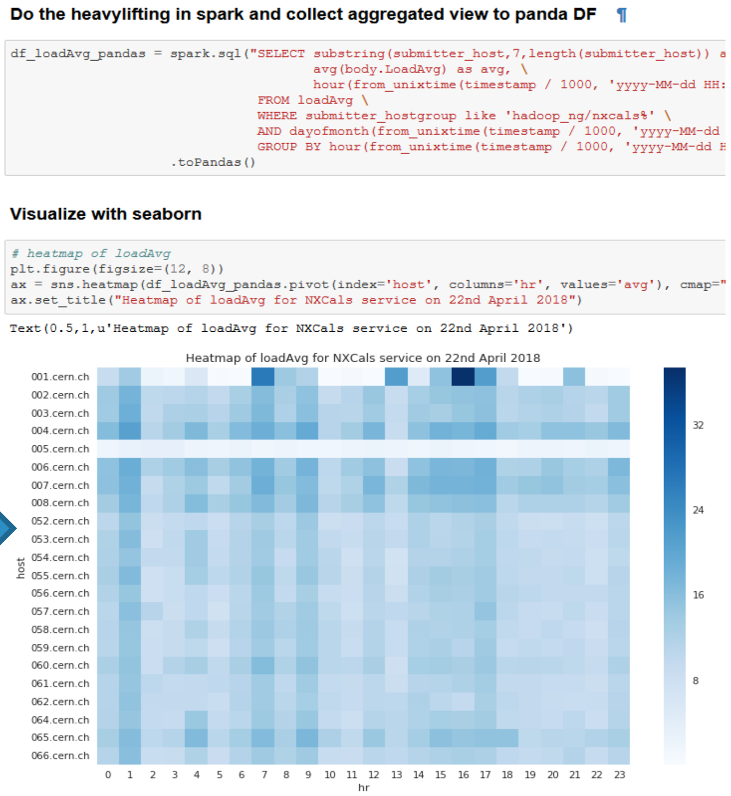
SQL Now

After moving the data (ETL),
add the queries to the new system
Spark SQL/DataFrame API is a powerful engine

- Optimized for Parquet (recently improved ORC)
- Partitioning, Filter push down
- Feature rich, code generation, also has CBO (cost based optimizer)
- Note: it's a rich ecosystem, options are available (we also used Impala and evaluated Presto)

Spark SQL Interfaces: Many Choices

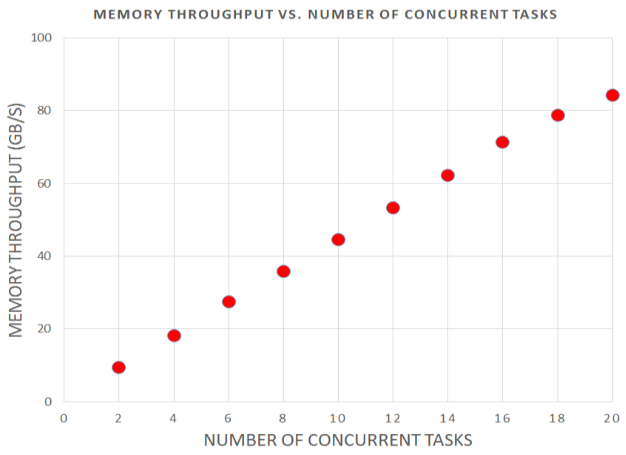
- PySpark, spark-shell, Spark jobs
  - in Python/Scala/Java
- Notebooks
  - Jupyter
  - Web service for data analysis at CERN swan.cern.ch
- Thrift server



Lessons Learned

Reading Parquet is CPU intensive

- Parquet format uses encoding and compression
- Processing Parquet -> hungry of CPU cycles and memory bandwidth
- Test on dual socket CPU
  - Up to 3.4 Gbytes/s reads



Benchmarking Spark SQL

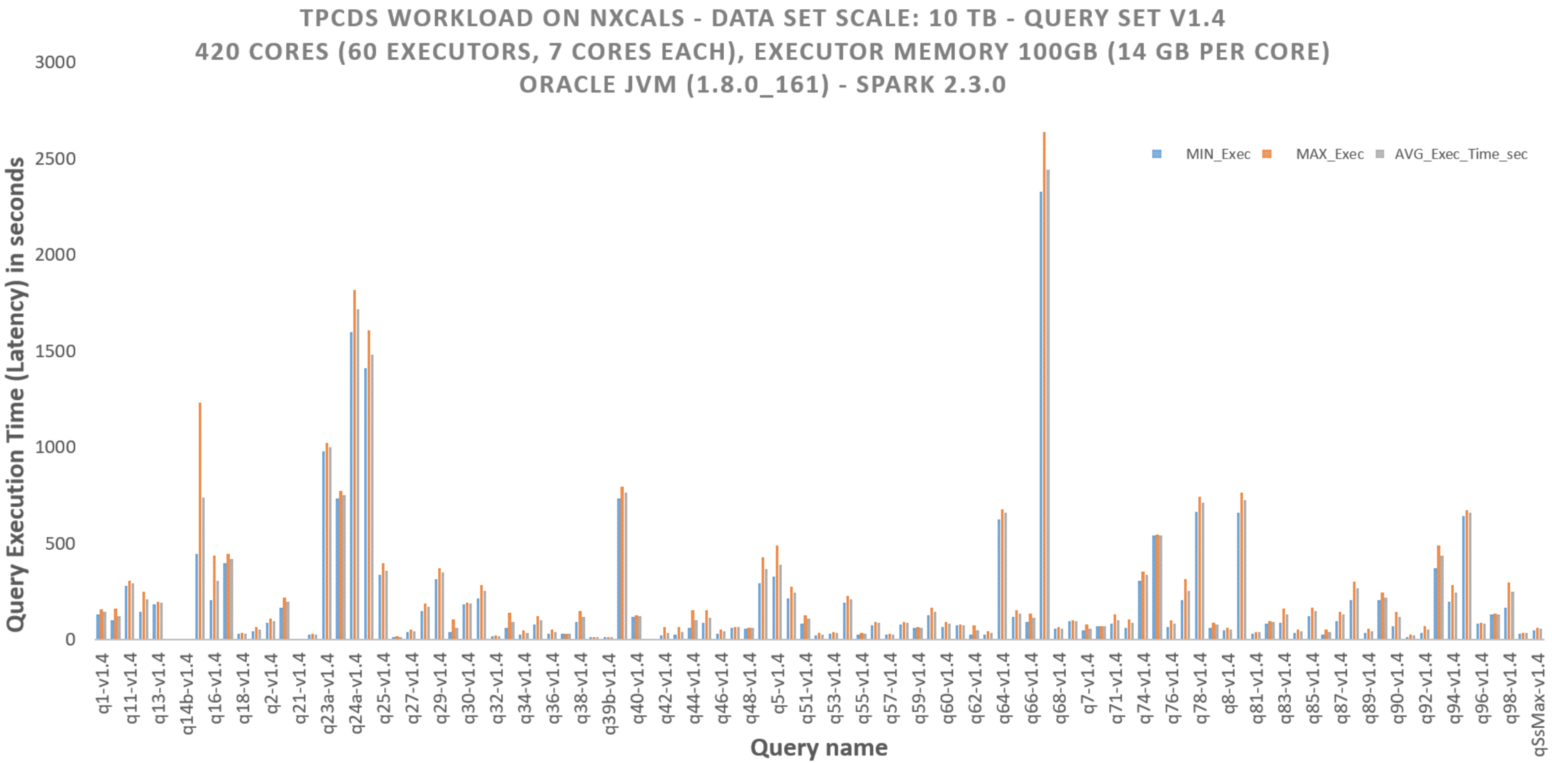
Running benchmarks can be useful (and fun):

- See how the system behaves at scale
- Stress-test new infrastructure
- Experiment with monitoring and troubleshooting tools

TPCDS benchmark

- Popular and easy to set up, can run at scale
- See https://github.com/databricks/spark-sql-perf

Example from TPCDS Benchmark



Lessons Learned from Running TPCDS

Useful for commissioning of new systems

- It has helped us capture some system parameter misconfiguration before prod
- Comparing with other similar systems
- Many of the long-running TPCDS queries
  - Use shuffle intensively
  - Can profit of using SSDs for local dirs (shuffle)
- There are improvements (and regressions) of
  - TPCDS queries across Spark versions

Learning Something New About SQL

Similarities but also differences with RDBMS:

- Dataframe abstraction, vs. database table/view
- SQL or DataFrame operation syntax

```
// Dataframe operations
val df = spark.read.parquet("..path..")
df.select("id").groupBy("id").count().show()
// SQL
df.createOrReplaceTempView("t1")
spark.sql("select id, count(*) from t1 group by
id").show()
```

..and Something OLD About SQL

Using SQL or DataFrame operations API is equivalent

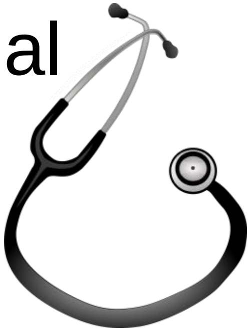
- Just different ways to express the computation
- Look at execution plans to see what Spark executes
  - Spark Catalyst optimizer will generate the same plan for the 2 examples

```
== Physical Plan ==
*(2) HashAggregate(keys=[id#98L], functions=[count(1)])
+- Exchange hashpartitioning(id#98L, 200)
   +- *(1) HashAggregate(keys=[id#98L], functions=[partial_count(1)])
      +- *(1) FileScan parquet [id#98L] Batched: true, Format: Parquet,
Location: InMemoryFileIndex[file:/tmp/t1.prq], PartitionFilters: [],
PushedFilters: [], ReadSchema: struct<id:bigint>
```

Monitoring and Performance Troubleshooting

Performance is key for data processing at scale

- Lessons learned from DB systems:
  - Use metrics and time-based profiling
  - It’s a boon to rapidly pin-point bottleneck resources
- Instrumentation and tools are essential



Spark and Monitoring Tools

- Spark instrumentation
  - Web UI
  - REST API
  - Eventlog
  - Executor/Task Metrics
  - Dropwizard metrics library
- Complement with
  - OS tools
  - For large clusters, deploy tools that ease working at cluster-level
- https://spark.apache.org/docs/latest/monitoring.html

WEB UI

- Part of Spark: Info on Jobs, Stages, Executors, Metrics, SQL,..
  - Start with: point web browser driver_host, port 4040

Spark 2.2.0 Jobs Stages Storage Environment Executors SQL Spark shell application UI

Stages for All Jobs

Active Stages: 7
Pending Stages: 7
Completed Stages: 347

Active Stages (7)

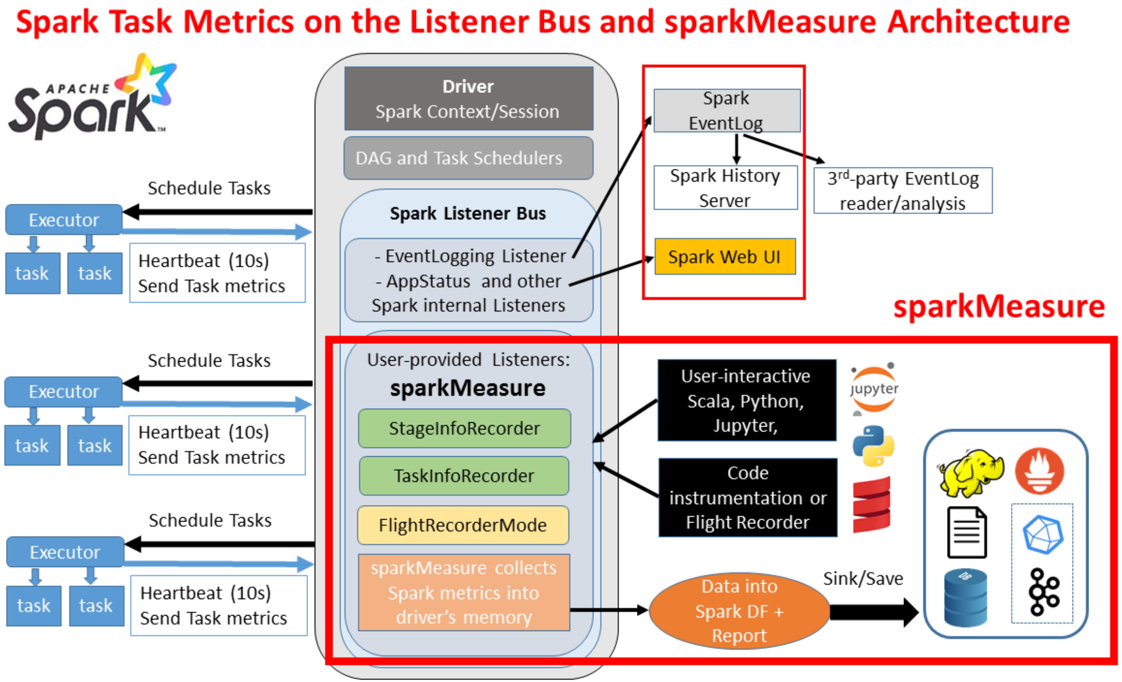
Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
375	benchmark q23b-v1.4 rdd at Query.scala:125 +details (kill)	2017/10/16 14:48:31	Unknown	0/200				
374	benchmark q23b-v1.4 rdd at Query.scala:125 +details (kill)	2017/10/16 14:48:31	Unknown	0/200				
373	benchmark q23b-v1.4 rdd at Query.scala:125 +details (kill)	2017/10/16 14:48:24	2 s	0/787				
372	benchmark q23b-v1.4 rdd at Query.scala:125 +details (kill)	2017/10/16 14:48:31	2 s	0/200			51.2 MB	
371	benchmark q23b-v1.4 rdd at Query.scala:125 +details (kill)	2017/10/16 14:48:31	2 s	0/200			98.4 MB	
367	benchmark q23b-v1.4 rdd at Query.scala:125 +details (kill)	2017/10/16 14:48:24	9 s	1468/1625	7.7 GB			663.4 MB
366	benchmark q23b-v1.4 rdd at Query.scala:125 +details (kill)	2017/10/16 14:48:24	26 s	1377/1379	137.3 GB			264.7 GB

Spark Executor Metrics System

Metrics collected at execution time

Exposed by WEB UI/ History server, EventLog, custom tools

https://github.com/cerndb/sparkMeasure



Spark Executor Metrics System

What is available with Spark Metrics, SPARK-25170

Spark Executor Task Metric name	Short description
executorRunTime	Elapsed time the executor spent running this task. This includes time fetching shuffle data. The value is expressed in milliseconds.
executorCpuTime	CPU time the executor spent running this task. This includes time fetching shuffle data. The value is expressed in nanoseconds.
executorDeserializeTime	Elapsed time spent to deserialize this task. The value is expressed in milliseconds.
executorDeserializeCpuTime	CPU time taken on the executor to deserialize this task. The value is expressed in nanoseconds.
resultSize	The number of bytes this task transmitted back to the driver as the TaskResult.
jvmGCTime	Elapsed time the JVM spent in garbage collection while executing this task. The value is expressed in milliseconds.
+ I/OMetrics, Shuffle metrics , …	26 metrics available, see https://github.com/apache/spark/blob/master/docs/monitoring.md

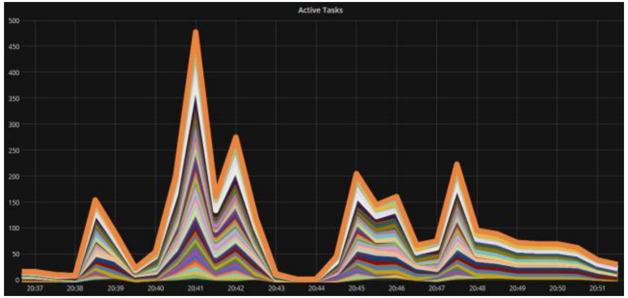
Spark Metrics - Dashboards

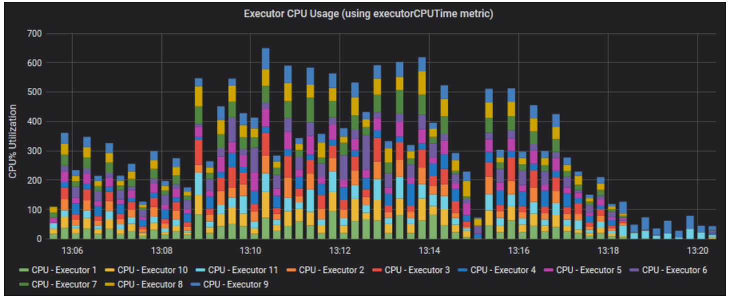
Dropwizard metrics
Graphite sink

Configure: --conf spark.metrics.conf=metrics.properties

Use to produce dashboards to measure online:

- Number of active tasks, JVM metrics, CPU usage SPARK-25228, Spark task metrics SPARK-22190, etc.





More Use Cases: Scale Up!

- Spark proved valuable and flexible
- Ecosystem and tooling in place
- Opens to exploring more use cases and.. scaling up!

Use Case: Spark for Physics

Spark APIs for Physics data analysis

- POC: processed 1 PB of data in 10 hours

Input: Physics data in CERN/HEP storage

EOS

ROOT Data Analysis Framework

Read into Spark (spark-root connector)

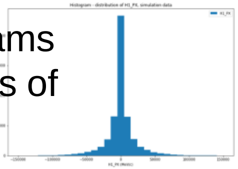
APACHE Spark

Apply computation at scale (UDF)

Data reduction (for further analysis

jupyter

Histograms and plots of interest



See also, at this conference: “HEP Data Processing with Apache Spark” and “CERN’s Next Generation Data Analysis Platform with Apache Spark”

Use Case: Machine Learning at Scale

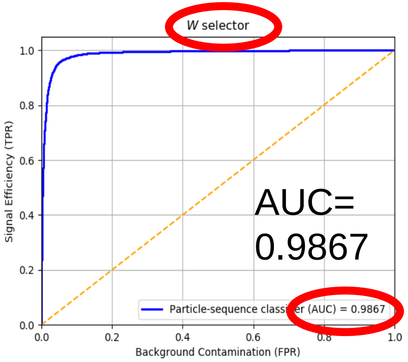
Data and models from Physics

Input: labeled data and DL models

Feature engineering at scale

Hyperparameter optimization (Random/Grid search)

Distributed model training



Output: particle selector model

Summary and High-Level View of the Lessons Learned

Transferrable Skills, Spark - DB

Familiar ground coming from RDBMS world

- SQL
- CBO
- Execution plans, hints, joins, grouping, windows, etc
- Partitioning

Performance troubleshooting, instrumentation, SQL optimizations

- Carry over ideas and practices from DBs

Learning Curve Coming from DB

- Manage heterogeneous environment and usage: Python, Scala, Java, batch jobs, notebooks, etc
- Spark is fundamentally a library not a server component like a RDBMS engine
- Understand differences tables vs DataFrame APIs
- Understand data formats
- Understand clustering and environment (YARN, Kubernets, …)
- Understand tooling and utilities in the ecosystem

New Dimensions and Value

- With Spark you can easily scale up your SQL workloads
  - Integrate with many sources (DBs and more) and storage (Cloud storage solutions or Hadoop)
- Run DB and ML workloads and at scale
  - CERN and HEP are developing solutions with Spark for use cases for physics data analysis and for ML/DL at scale
- Quickly growing and open community
  - You will be able to comment, propose, implement features, issues and patches

Conclusions

- Apache Spark very useful and versatile
  - for many data workloads, including DB-like workloads
- DBAs and RDBMS Devs
  - Can profit of Spark and ecosystem to scale up some of their workloads from DBs and do more!
  - Can bring experience and good practices over from DB world

Acknowledgements

- Members of Hadoop and Spark service at CERN and HEP users community, CMS Big Data project, CERN openlab
- Many lessons learned over the years from the RDBMS community, notably www.oaktable.net
- Apache Spark community: helpful discussions and support with PRs
- Links
  - More info: @LucaCanaliDB – http://cern.ch/canali