Distributed Systems Architecture

brought to you by Alexey Grishchenko

Apache Spark Future

Everyone around the internet is constantly talking about the bright future of Apache Spark. How cool it is, how innovative it is, how fast it is moving, how big its community is, how big the investments into it are, etc. But what is really hiding behind this enthusiasm of Spark adepts, and what is the real future of Apache Spark?

In this article I show you the real data and real trends, trying to be as agnostic and unbiased as possible. This article is not affiliated with any vendor.

Official Position

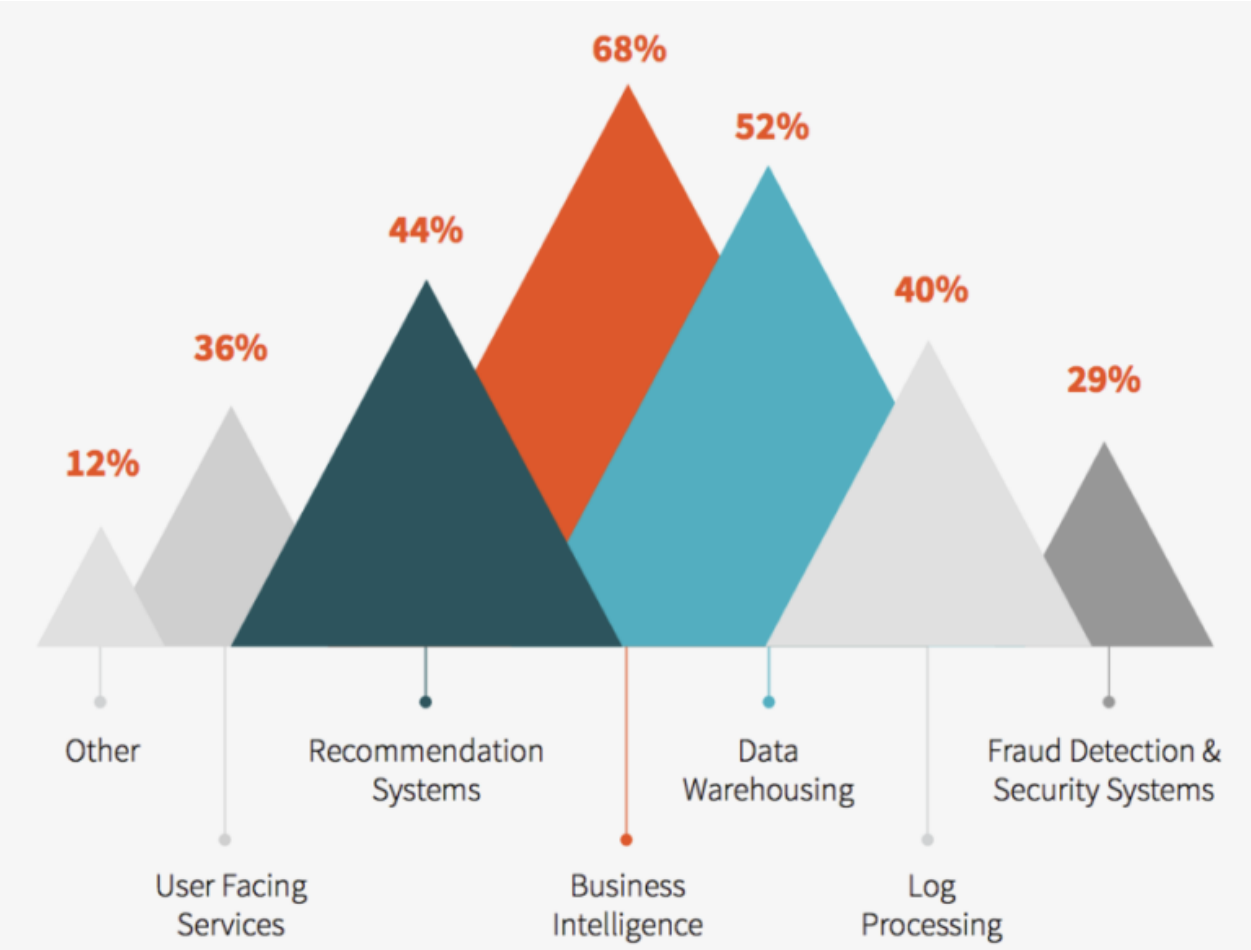
Let's start with official position of Databricks on the shiny future of Apache Spark. Here is the slide from Databricks presentation on Apache Spark 2.0, the major new release of this tool:

You can see that 2 out of 3 new major features are related to SQL: SQL 2003 compliance and Tungsten Phase 2, that was targeted to greatly speed up SparkSQL by delivering a big number of performance optimizations. The last improvement is streaming, but again – structured streaming, which would underneath reuse parts of the code introduced for SparkSQL (same presentation):

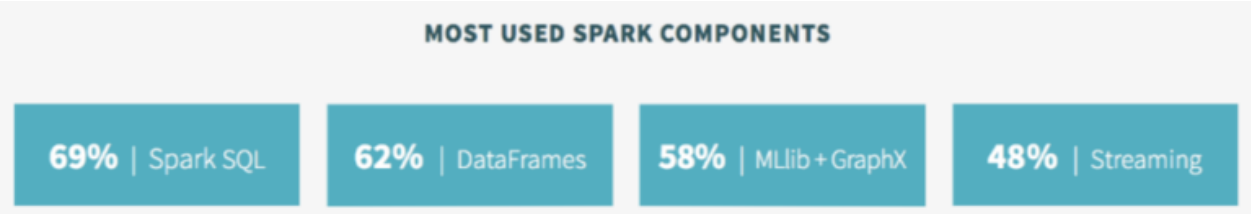
So it is getting interesting – all the 3 major improvements introduced in Spark 2.0 are about SQL!

Spark Survey 2015

So far so good, let's take a look at the Spark Survey handled by Databricks one year ago. Most interesting parts are this:



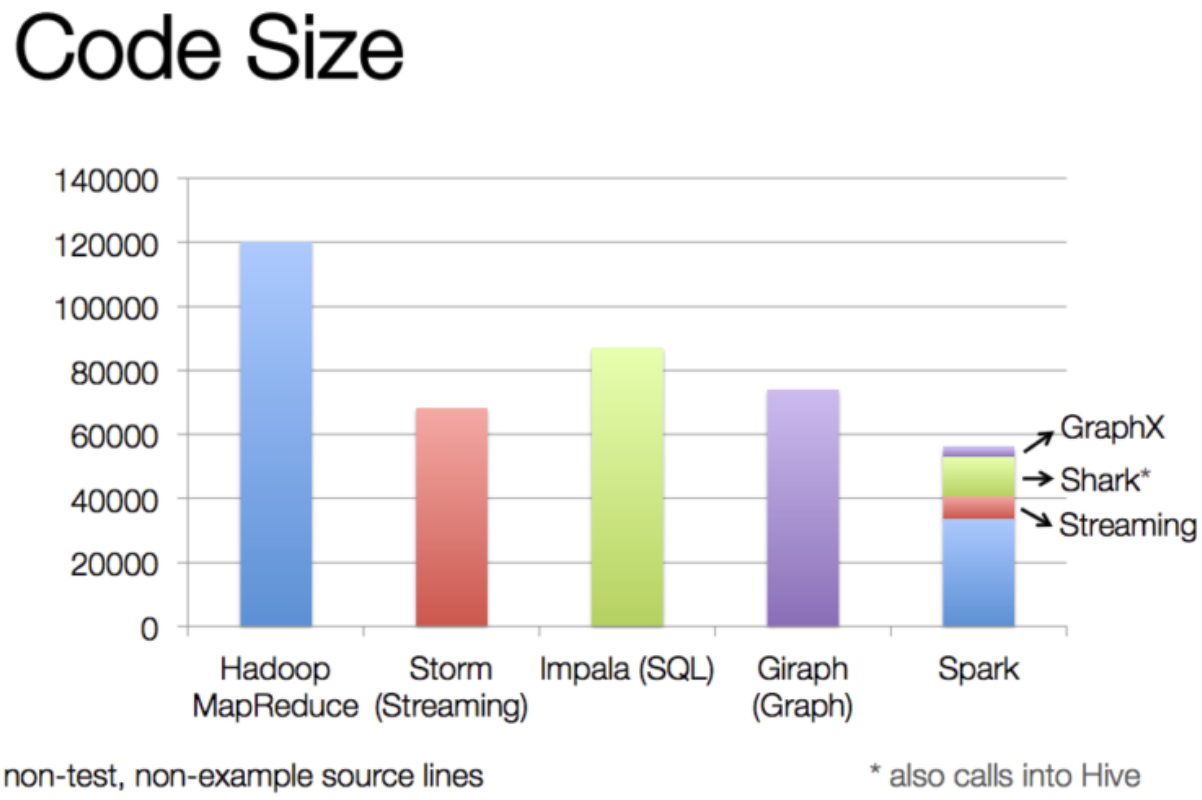
And this:



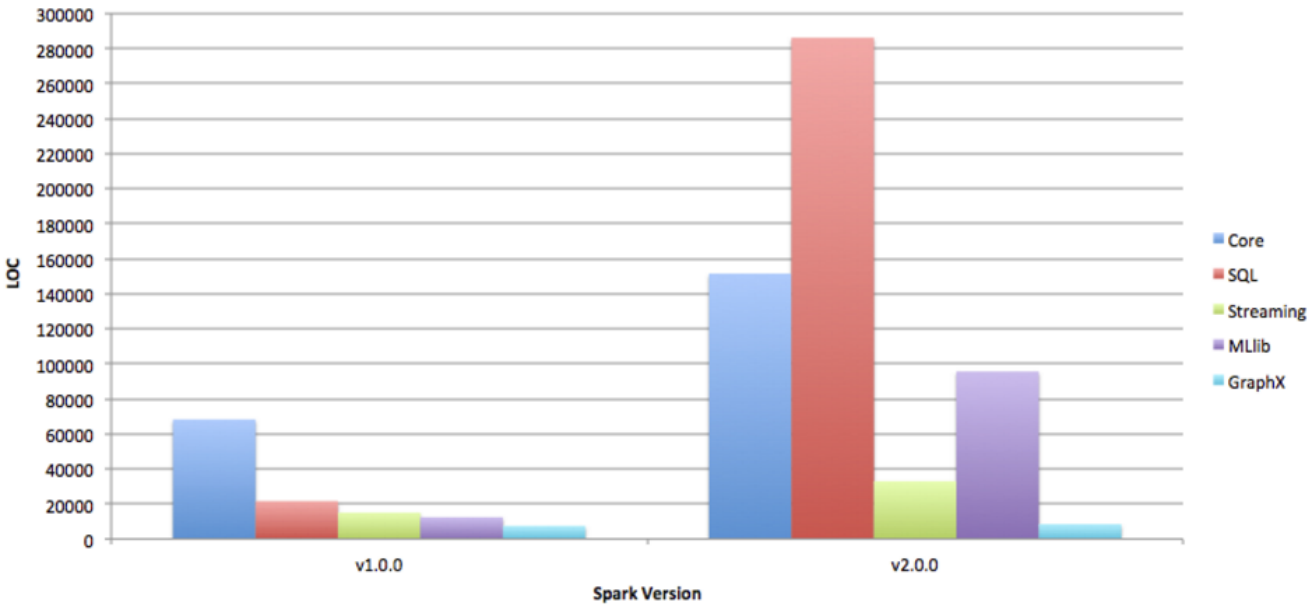
You can see that 69% of the customers are using SparkSQL, and 62% using DataFrames, which essentially use the same processing layer with SparkSQL (Catalyst optimizer and in-memory columnar storage). Also, two biggest use cases for Apache Spark are Business Intelligence (68%) and Data Warehousing (52%), both of them are pure SQL areas.

Apache Spark Code

Again, what was the original idea of Apache Spark, when it was introduced by AMPLab of Berkley? Let's take a look at Matei Zaharia's presentation on Apache Spark from Spark Summit 2013:



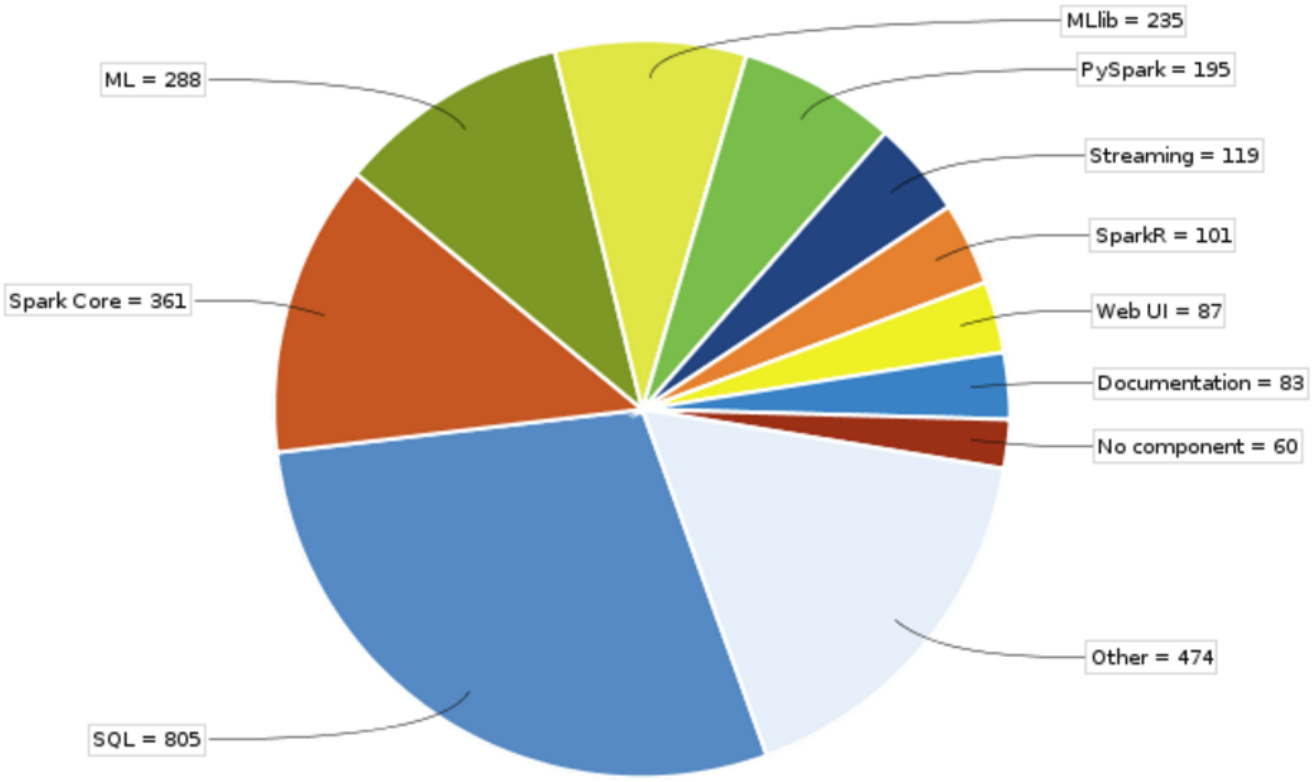
One of the biggest Apache Spark advantages is simplicity. Its code is compact, and everything is based on the Core engine, introducing RDDs and DAGs. But what about now? We can easily check this, as Apache Spark is an open source project. Here you can find some statistics built on its source code:



Left group of columns represents Apache Spark v1.0.0, approximately the same release Matei was speaking about on the Spark Summit 2013. The right columns represent the current master branch which is approximately the same as Spark v2.0.0 (6 commits ahead). Take a look at who is leading now – the biggest traction in community is caused by SparkSQL and MLlib! Streaming is growing times slower, while GraphX has almost nothing new, its code base has grown by roughly 1000 lines of code.

Apache Spark JIRA

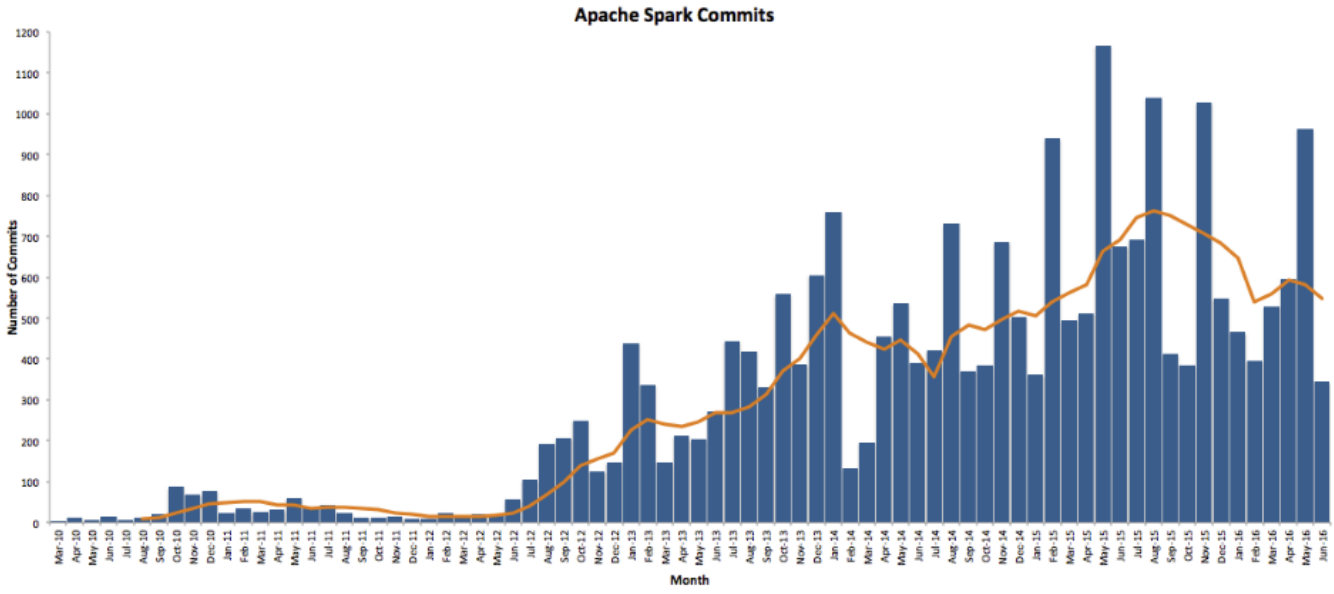
Good, now let's turn from historical perspective to the future perspectives of Apache Spark. Let's take a look at the open issues in Apache Spark JIRA, splitting them by the component:



It is no longer a surprise for you, but SQL component is related to 34% of the issues, while Core is only 15%.

Apache Spark Contributions

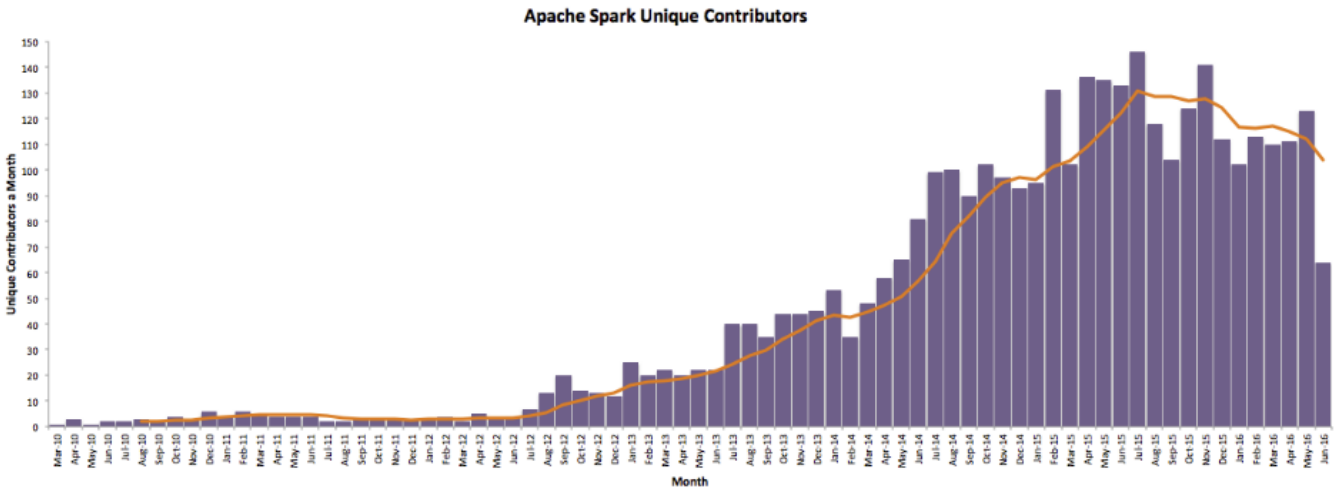
I will drop some more charts before moving to the conclusions. Here is the number of commits to Apache Spark per month since the project was established:



Orange line is a moving average over the past 6 months, it is used to normalize the contribution peaks and show the general contribution trend. We can see from this chart, that Databricks works on Apache Spark

using 3-months development cycles, and the missing peak of 2016'Feb corresponds to the time they were working on Apache Spark 2.0 release.

And now another graph, number of unique contributors to Apache Spark a month:



Again, orange line is a trend line showing moving average across the last 6 months.

Conclusions

Apache Spark was introduced by AMPLab as a general-purpose distributed data processing framework. Databricks was formed from the AMPLab people who worked on Apache Spark, to make this engine a huge commercial success, and this is when the things went wrong. Corporates can vote for the project direction with their money, while everything community can offer is limited individual contributions. Little-by-little Apache Spark is moving from being general purpose execution engine, to the corporate space, where SQL is the main and only standard for data processing. Apache Spark starts to compete with MPP solutions (Teradata, HP Vertica, Pivotal Greenplum, IBM Netezza, etc.) and SQL-on-Hadoop solutions (Cloudera Impala, Apache HAWQ, Apache Hive, etc.). At the moment Apache Spark is not positioned as their competitor because of the obvious fact – in its current state it will lose this battle. But it is getting closer and closer to its real competitors, and here is where the things are getting interesting: enterprises want the functionality they used to, and it is shaping the future of Apache Spark, putting it into the category of solutions where it cannot efficiently compete. Databricks team is putting tremendous efforts in making it a good competitor in SQL space, but it has a very low chance of winning this battle against 30-years veterans like Teradata and 40-years like Oracle.

And here are the promised conclusions:

- SparkSQL is the future of Apache Spark. Apache Spark competes in SQL space against MPP databases and SQL-on-Hadoop solutions, and the battle is tough
- Apache Spark is getting substantially bigger (650'000 LOC already) and more complex, increasing the entry barrier for new contributors
- Enterprise investments to Apache Spark turn out to be the investments in making it capable of integrating with their products (Spark on IBM Mainframe, Spark-Netezza Connector, Spark on Azure, Spark in Power BI, etc.), not really making Apache Spark better

My personal perspective on this is the following:

- In 1 year Spark would start being officially competitive with MPP and SQL-on-Hadoop solutions
- In 2 years Spark would lose the battle against MPP and MPP-on-Hadoop solutions and take a niche of Hive in Hadoop ecosystem
- In 2 years it will lose the market share in stream processing to specialized solutions like Apache Heron and Apache Flink
- In 3 years in might also lose the share in distributed machine learning to new age solutions like TensorFlow

Happy coding to you all! If you have any questions, I'm open for the discussion both here and in my twitter @0x0FFF.

This entry was posted in DBMS, Spark, SQL-on-Hadoop and tagged apache spark, future of apache spark, Spark, sparksql on June 14, 2016 [https://0x0fff.com/apache-spark-future/] .

36 thoughts on "Apache Spark Future"

Sasha Parfenov
June 14, 2016 at 11:08 pm

"In 2 years it will lose the market share in stream processing to specialized solutions like Apache Heron and Apache Flink"

Heron is not an Apache project. But Apache Apex is a new top level apache project for stream processing, which should be mentioned in this context.

0x0FFF Post author
June 15, 2016 at 7:28 am

Agree, Heron is *not yet* Apache project, but it would be soon: https://github.com/heronproject/apache-proposal

Prithiviraj Damodaran
November 8, 2016 at 11:27 am

IMHO, Heron adoption may not catch on really quickly as it uses MESOS as opposed to YARN

Applicative (@applctv)
June 14, 2016 at 11:27 pm

One advantage of Spark as opposed to an MPP is that you can still break out of SparkSQL abstraction (DataFrame/DataSet) into a lower level RDD and perform computation that is not trivial to define in SQL or put your data through an ML pipeline. Sure, it will never be as performant as MPPs, but this feature might sustain it for a lot longer, and give it weight in the ecosystem.

0x0FFF Post author
June 15, 2016 at 7:40 am

I agree with you, and this is why I like Spark. But the problem here is that this is only one part of the big puzzle, and I think the reason you put is the one that will let Spark win over Hive. But not over MPP – it is not only about performance, it is also about ACID properties, backup-restore and distasted recovery and many more. Spark will surely take a place in “Big Data” ecosystem, but it won’t completely replace MPP and MPP-over-Hadoop

Reynold Xin
June 15, 2016 at 8:50 am

Nice link bait title and interesting analysis. I think you are either intentionally or unintentionally misinterpreting what Spark SQL is in your analysis – Spark SQL is not just about SQL. It turns out the primitives required for general data processing (eg ETL) are not that different from the relational operators, and that is what Spark SQL is. Spark SQL is the new Spark core with the Catalyst optimizer and the Tungsten execution engine, which powers the DataFrame, Dataset, and last but not least SQL.

This design is actually one of the major architectural advantage of Spark. A core engine (used to be Spark core, and now increasing so the Spark SQL project) that is shared by multiple user facing APIs, providing

great flexibility as well as collective performance improvements. I have not seen a lot of other projects with the same level of ability and audacity to constantly reinvent itself towards better design.

Using Spark SQL as the component to analyze the amount of work and attribute those to potential competition with MPP is very misleading and possibly meaningless here. Not entirely your fault though. Perhaps we should have given Spark SQL a different name instead.

0x0FFF Post author
June 15, 2016 at 9:09 am

I understand that SparkSQL is not only about SQL, but also DataFrame and Dataset (which you have merged together in Spark 2.0). But here is what I am talking about: both SQL queries and DF/DS transformations are ultimately lead to the same Catalyst optimizer execution planning, i.e. they are both translated to the same execution primitives, which in my opinion means that effectively DataFrame offers you just a different syntax to express the same operations you can do in SQL (and vice versa). But the main advantage of Spark was the fact that it allowed more data transformation flexibility compared to original SQL, like functional programming transformation descriptions with inline lambdas, the coolest stuff I've seen in the last years.

Here I'm telling that moving on and on with columnar and typed approach of SparkSQL, you are moving towards the space where RDBMSs play much better than Spark, and away from the space where Spark really shines

---

Ofir Manor
June 15, 2016 at 10:22 pm

I like this post and the research behind it, but I think your analysis is off… Some random thoughts:
1. Spark Dataset is a cleaner, easier and better general-purpose API than RDD. It is not about SQL, it is about developer productivity and by being less opaque, it unlocks a lot of possible dramatic optimizations.
2. It seems that both ML and graph processing over Spark are moving to "dataset 2.0", so I don't see how it is "just a different syntax to express the same operations you can do in SQL". It is a good foundation for functional programming, SQL and iterative algorithms.
3. Regarding perf vs. MPP – I saw the Tungestan project as "catch up on all major MPP low-level perf optimizations" – vectorization, code generation, columnar data encoding and efficient memory management (2.0 also adds efficient columnar I/O processing with Parquet, and the optimizer is supposed to decent as well).
I think Spark 2.0 is already pretty competitive with other SQL-on-Hadoop solutions, but that is not really fair, as all the Hadoop vendors are rushing to rebase their offering on Spark anyway (at the very least, no one heavily promotes MapReduce and Tez – Cloudera still loves Impala tough). So, Spark SQL does not really compete with Hive, it is more like Hive-on-Spark will become the default soon.

4. MPP databases are a legacy niche (saying that as a former Greenplum employee…). Spark let's you combine together SQL and "ETL" and machine learning and graph processing and custom code, either in batch mode or streaming mode, which is dramatically more ambitious than any DW platform out there. Regarding perf (or price/perf) vs. legacy MPP – it is officially hard to tell (can't officially publish benchmarks), but Cloudera's "Impala vs. a mystery leading commerical MPP (DBMS-Y)" posts of early 2014 can give a good approximation of the gap between those legacy MPP databases and the open source offering (spoiler – there is a reason why they are a legacy niche).
5. Your perspective of "Spark=MPP SQL engine" is hard to reconcile with the massive backing of companies like IBM, which generally decided last year to migrate all its many analytical products to Spark (and even periodically shares update reports on this).

0x0FFF Post author
June 16, 2016 at 7:34 am

1. I used both DF and RDD and don't share your enthusiasm, sorry. I clearly understand the benefits of DS being columnar and typed, which allows you to have a big number of optimizations, but because of this it is less flexible

2. Yes, datasets are becoming "new core" of Apache Spark just like Reynold Xin said. And yes, they are replacing RDDs for MLlib and GraphX. But I repeat again, all the DS operations are translated to the execution steps for Catalyst optimizer, same execution steps used for SQL optimization chains in SparkSQL. So what I'm saying is that in terms of expression power Spark Dataset = SQL. And here you have it – creating DS on top of another DS are views in SQL, caching DS is temp table in SQL, defining UDF to change the field of dataset is the same as UDF in any RDBMS, and so on

3. Yes, and I put it in conclusion – my opinion is that Spark would replace Hive and Tez completely. There won't be "Hive on Spark", there would be just SparkSQL. Also Hortonworks might soon give up on Tez and pay more focus to SparkSQL. MapR has already announced replacement of MapReduce in their platform with Spark

4. You would be surprised, but after 10 years Hadoop still cannot replace MPP databases. I've participated in a number of attempts to do so, and I would say I don't see it replacing MPP in the nearest future. Why do you think there is "data lake", "data hub", and other designs? Because Hadoop cannot replace MPP and vendors propose these solutions to co-exist. Why did Cloudera offered Kudu? To fight the traditional MPP solutions, and Impala+Kudu can potentially replace traditional MPP RDBMSs in DWH use case.
Here I am talking not only about specific MPP RDBMSs, I'm talking about the general MPP design – given the same hardware, MPP would always be faster than any MapReduce-based system (Spark or Hive or Tez), it is just "by design". Also, don't put Impala in the same list with Spark SQL – Impala is MPP engine reading data from Hadoop, while Spark is batch (i.e. "map-reduce compute paradigm"-based), so comparison of Impala vs DBMS-Y is a competition of 2 different implementations of MPP design. On top of this my personal point is "never trust vendor benchmarks" – Impala gives you a great single-query execution timing because each single (big enough) query utilizes 100% of cluster resources, while traditional MPP RDBMSs reserve some resources to better serve concurrent queries

5. I've never said that "Spark = MPP SQL engine". SparkSQL offers you SQL execution engine based on "map-reduce compute paradigm", with query optimization by Catalyst. In this article I'm telling you that the main direction of Apache Spark development is making it better at SQL workloads (supporting all the 99 queries of TPC-DS, supporting ANSI SQL 2003 OLAP extensions, etc.), because it is the thing asked by enterprises. And this alone puts Spark into position when it competes with MPP RDBMSs, and the competition of "map-reduce compute paradigm" vs "MPP compute paradigm" is a lost battle, at least in terms of performance. It is really fun to watch – in fact, Spark marketing avoids direct positioning against traditional MPP RDBMSs, but the enterprises are actively moving Spark in this direction and soon it won't be able to avoid open competition

Regarding IBM and Microsoft investments in Spark – check the links in the last part of my article, I briefly cover their activity

---

Ofir Manor
June 16, 2016 at 10:35 am

I'm not sure a point-by-point discussion will work, since I see reality quite differently. Thinking about, I have two main disagreements with you…
1. Technically, Spark (and Tez) is in no way related to the MapReduce paradigm.
MapReduce design (like Teradata BTW) was "spill to disk between intermediate operators, to allow fast recovery of long running jobs".
Tez, Spark, Impala and the rest of the MPP world do "pipeline the data between operators, to allow faster results" (Spark's RDD is unique, as it enables faster recovery even with pipelining).
So, I can't really parse the technical claims around Spark being "MapReduce-based system". Also technically, it doesn't work on top of map-reduce pairs but can pipeline any DAG (same as Tez, Impala, Greenplum etc).
2. On an higher level, being an MPP database is a technical implementation detail, not really a market. A decade ago, the new MPP vendors (Teradata was MPP since the 80s) wanted to win the EDW market, leveraging MPP on commodity HW as a way to get unprecedented price/performance. The existing players adjusted accordingly (ex: Exadata), and I really don't see anyone getting excited about it anymore.
The "big data" hype is about a platform that is of much wider scope. It adds value over the EDW by combining large scale storage, ETL, machine learning and queries. If some early adopters thought Hadoop would be a magical drop-in replacement for their Teradata / Exadata reporting, only cheaper, I guess it didn't work as well as they expected.
On a related note, the Impala + Kudo also aims at a wider scope – making building a scale-out operational system + analytics simpler, by having a powerful shared storage that balances both needs (conceptually, similar to what SAP Hana proposes – a single shared infra).
If you wanted to evaluate that using current technology (spark 2.0 or latest CDH with Impala), you need to compare to "MPP database" + scalable ETL + a way to add machine learning at scale, and see where you get in terms of complexity, performance, cost etc. Not sure Spark will lose this. I think that is where Spark will continue to go, since that was its track for years now.

0x0FFF Post author
June 16, 2016 at 11:45 am

1. I have used the wrong term, by “map-reduce processing paradigm” I meant more general Batch processing paradigm. And I disagree with you: while Teradata, Greenplum, Impala and HAWQ are MPP solutions, MapReduce, Tez and Spark are Batch solutions.
MPP solutions work by running all the “executors” at the same time, which allows you to pipeline the data between them during “shuffle” and avoid spilling to HDDs
Batch solutions work by splitting the “processing” job into a set of “tasks”, each operating on independent part of data, with dependencies between tasks usually defined by shuffles. This paradigm allows you to schedule tasks independently and safely rerun failed task, but each task output should be saved to HDDs
And both Spark and Tez are Batch processing engines, not MPP. Spark puts intermediate data to HDDs, you can check it in the code, or check the description for spark.local.dir in official documentation
http://spark.apache.org/docs/1.6.1/configuration.html#application-properties
My point is that Batch is always slower than MPP, but it is more reliable for long-running computations (faster recovery that you mentioned). This is why Spark is trying, but would never reach performance level of MPP databases like Greenplum and Teradata

2. Regarding “‘MPP database” + scalable ETL + a way to add machine learning at scale’. Spark does not have a scalable ETL, it has a scalable execution engine just like all the MPP solutions have. You cannot build EDW without ETL tool like Informatica PC, IBM DS, SAS DI and similar – they are the main storage of information about the data consolidation, cleansing and transformations in your EDW. Going with plaintext scripts in Spark (this is what it offers) and using something like Oozie for scheduling them is too risky for enterprises, and no one except startups would go this way.
Regarding machine learning at scale – each MPP database vendor has its own offering for this (Greenplum MADlib, Teradata Aster in-database analytics, Netezza in-database analytics, Vertica, etc.). Of course, their solutions lose to MLlib in terms of functionality, but they exist and used in the field.

IMO, Spark is pulled to play in EDW market by the customers that still buys the message of “cheaper DWH on Hadoop” and requiring strong SQL story from Spark, while Spark has no option of winning in this field. However, it would clearly dominate Hadoop batch processing market, as I state in my predictions, taking the place of Hive (and it has already killed Mahout). However, losing to specialized MPP data processing solutions, to specialized streaming solutions like Heron, and to specialized distributed ML solutions like Tensorflow

---

Reynold Xin
June 17, 2016 at 6:58 pm

You should really look more into the internals. A lot of the fundamental assumptions in your analysis are just wrong.

For example, the Catalyst optimizer has been extended to handle a lot of things that are not SQL specific. Calling it an “optimizer” does not mean it is just a SQL optimizer. You are effectively saying “C++ is just SQL because there is an optimizer”.

0x0FFF Post author
June 17, 2016 at 8:23 pm

I always welcome rational criticism. I agree that I put the SQL question a bit sharp, I just wanted to pay attention to this issue.
I see that SparkSQL is becoming the new core of Apache Spark, and this is the shift I don’t personally support, so I gathered this article to show the fundamental flow of putting SparkSQL in the center of your strategy

I understand that not every “optimizer” is “query optimizer”, and I understand that ultimately any computer program is decomposed to assembler code executed on CPU, which also optimizes much stuff. The important part here is the level of decomposition and set of primitives it is able to operate. I just wanted to show that effectively DataFrame/DataSet API is not much flexible than the same SQL

Reynold Xin
June 17, 2016 at 7:02 pm

BTW a lot of these are still work in progress. Regardless of our discussions and positions taken here, would love to get more feedback to create a better Spark and ultimately make users’ life easier!

0x0FFF Post author
June 17, 2016 at 8:25 pm

Great point, I’d also love to make it better

Reynold Xin
June 17, 2016 at 7:08 pm

One opinion I saw in this blog is that “given the same hardware, MPP would always be faster than any MapReduce-based system, it is just ‘by design’”.

I think this is one of the misconceptions people have about MPP vs “Big Data”, or “MapReduce-based”. MPP really just an implementation detail, and if you go back to the definition of MPP as defined by DeWitt, there is no major gap between what Spark’s engine does and that definition of a shared-nothing parallel database. I’m happy to debate — or even better when I have time I might try write a blog post about it.

0x0FFF Post author
June 17, 2016 at 8:41 pm

Great, I was also thinking about writing a port about it! Would love to read yours and make some comments

The gap between MPP and Batch is not big, but it exists. For certain workloads it is almost invisible, but for certain it is very sensible.

I’m not telling that Batch is a bad design and MPP is good – my personal point is that MPP is good for up to 50 machines in the cluster (because of its pipelining design), and batch is good starting from around the same 50 machines up (because of speculative execution and job tolerance to partial failures)

Another big problem around Spark is hype. Still >90% of people call Spark “in-memory” technology, and same >90% are sure Spark pipelines data just like MPP databases. On top of this put SparkSQL that now supports ANSI SQL 2003 OLAP extensions and all the 99 TPC-DS queries – and you get tens of companies implementing their DWH, unfortunately unsuccessfully. At the beginning it is fun to hear claims like “where is the ACID?”, and “why cannot I just have a transaction here?” after 6+ months of DWH implementation, but later you realize a fundamental flaw of this whole situation

---

Duc Kien
June 30, 2016 at 4:53 pm

Are you familliar with Apache Flink ? I believe it has a pipeline shuffle, does that mean it is closer to MPP than Spark ? It’s currently lacking a SQL interface though.

0x0FFF Post author
July 2, 2016 at 10:16 pm

A bit, but I haven't used it in production. In fact, Flink is neither MPP nor Batch – it is a real-time system, which is completely different class of data processing systems. Also, Flink has recently introduced queryable state, but still its main use case is real-time workload, not analytical one, so I wouldn't put Apache Flink into the same category with Apache Spark

---

Artur
June 23, 2016 at 1:06 pm

```
I'm happy to debate – or even better when I have time I might try write a blog post about it
```

Please, write it! Everyone would be happy to see such blog post from you 🙂

Reynold Xin
June 17, 2016 at 7:15 pm

One other comment: based on your comments, one thing you really enjoyed was the RDD programming model. Catalyst/Tungsten aims to provide the same API while

However, RDD are two separate APIs: one is the user-facing API (the one you liked), and the other one is being used to describe the internal iterator-based plan. The internal API is actually very similar to query plans in databases — we just never advertised it that way.

This tight coupling of the user-facing API with the internal plan API has caused a lot of problems. What Catalyst/Tungsten is really doing is to decouple the external user-facing API from the internal implementation details, while enforcing some structure that sacrifices a little bit of flexibility to achieve much better execution (performance, robustness, integration with external structured data storage systems such as RDBMS, key value stores).

Reynold Xin
June 17, 2016 at 7:18 pm

One more comment: putting "intermediate data on disk" is an implementation detail and something that can be changed. However, it is not performance bottleneck in most cases because exchanging a large amount of data is bottlenecked by the network. Many MPP databases also put intermediate data on disk.

0x0FFF Post author
June 17, 2016 at 8:04 pm

I don't agree that putting intermediate data on disk is implementation detail. The only option to avoid putting intermediate data to HDDs during shuffle is pipelining the shuffle data, which can be achieved if and only if all the "mappers" and "reducers" are running at the same time, which does not comply with the concept of Batch. Batch splits the data processing job into a number of tasks with dependencies between them (i.e. DAG), and the tasks are executed asynchronously. The cluster with 10 execution slots can process a job with 1000 tasks, and there is no way to pipeline the data in this case. However, MPP systems are processing shuffle by always having all "mappers" and "reducers" running at the same time, and thus the data can be pipelined between them.

I agree that SSDs are getting cheaper, RAM is getting cheaper and NVRAM enters the market, so in the nearest future it might turn out that putting intermediate data to the disks does not introduce a sensible overhead (or you would be able to have enough RAM to handle intermediate data in RAM). But for now, all the "big data" clusters (i.e. at least 100TB) are using traditional magnetic spindle HDDs, and the problem of intermediate data storage is important for them

MPP systems really put intermediate data to the HDDs during "sort" steps or hash join / aggregation, but in fact it is the same for Batch systems. Here I'm focusing on what is different between them, while spilling "sort" data is common

Regarding "network" as bottleneck – networks are also progressing, and it is not a surprise that many modern clusters have dual 10GbE per node, which allows you to transfer the data at 2GB/sec – more than 14 HDDs can deliver

---

Reynold Xin
June 17, 2016 at 8:21 pm

"which does not comply with the concept of Batch"

I think this sentence is the indication of the level of discussion and disagreement here. Spark is not a static project that does not evolve and conform to a specific set of implementation details as dictated by "batch" vs "MPP". You seem to have a very rigid definition of "batch" (or MapReduce) vs "MPP" and you are putting Spark into a box. In reality, one major strength of the project is that people working on it are not subject to specific doctrine, and are willing to just pick the best way to do things, rather than confined by "batch" vs x vs y.

As a concrete example, I've personally had done multiple prototypes in the past in which intermediate data does not go to disk (as written in a few research papers already). There are also prototypes done by others that changed Spark to do pipelined shuffle.

0x0FFF Post author
June 17, 2016 at 8:55 pm

I'm glad that Spark is evolving and may offer some new paradigm soon. But I'm not putting Spark into a box, I'm just judging of it by its current implementation. Changing Spark to pipeline shuffles is possible, but would be applicable only to a limited set of cases – mainly the same cases that correspond to processing the amount of data that can fit into RAM, and thus they would provide no real benefit compared to current implementation. I there was a PoC of this, I think this was the main reason why such design was rejected

Ofir Manor
June 19, 2016 at 4:15 pm

I came back to discussion, maybe this is the best place to add:
1. All databases, MPP or others (and all processing engines), have to spill to disk when doing anything interesting – large sorts, large aggregations, large joins etc – as the (per-node) working set is too large (small is usually a solved problem).
The actual threshold is actually not big, as in real-life you need to support concurrent queries, and each query may have multiple "interesting" (RAM-intensive) steps and many regular steps that still need RAM, so that would typically be a few GB of RAM per node (at most), unless there is a dedicated cluster for a specific computation.
2. Spilling to disk is actually decent with spinning disk (large sequential I/O) – which is why it is not really a huge problem (you can squeeze at least 150MB/s per HDD for large sequential I/O, definitely can reach more 1GB/s per node). It is much slower than RAM, but not far from the network throughput. So, disks and network might be highly utilized, but not necessarily become a throughput bottleneck. The real challenge of HDD is that small random I/Os can significantly interference with sequential I/Os.

0x0FFF Post author
June 20, 2016 at 7:50 am

I agree with your arguments. What you forget is that for example, for joining 2 big fact tables Spark would spill data to HDDs 3 times (first table shuffling by join key, second table shuffling by join key, hash table disk overflow), while MPP system would spill only once (hash table disk overflow). Add on top of that MPP databases can usually take advantage of co-located joins, and you get at least 2x-3x performance advantage for MPP

Ofir Manor
June 20, 2016 at 11:17 am

Thanks for the followup!
we might be mixing two arguments…
One is whether Spark can provide "decent" or even "great" performance or price/performance today. Here I was under the impression that SparkSQL was not far behind Impala as of Spark 1.5 (according to Impala's own benchmarks and blog posts *), so I do believe the gap today (Spark 2.0) can't be dramatic, to say the least.
The other is the two year roadmap, and their guaranteed future failure to win over Impala or Greenplum or some other player two years down the line. You do claim that they will focus on this niche, but will fail to deliver two specific (but important) performance features. I would claim that if that would be their focus (as you claim), I don't see a reason why a decent implementation can't happen in such time frame of these or other performance-critical features (personally I think that having a world-class optimizer is a harder problem).
Anyway, I think the opposite will happen. Spark will become the default, one-stop-shop for all types of computation. Any competing MPP database will have to convince that running some queries faster is worth the hassle of replacing Spark and its richness, or worth adding another component to the big data stack.

For reference, Impala's perspective of benchmarking Spark 1.5 vs. Impala 2.3 – nothing as dramatic as the good old "everything is 100x faster than hive" days, and it is two significant Spark releases behind.
http://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3-uniquely-delivers-analytic-database-performance/

0x0FFF Post author
June 20, 2016 at 1:16 pm

First, *"running some queries faster is worth the hassle"* – you should tell this to all NoSQL and NewSQL solutions that replaced MSSQL, Oracle and MySQL, I believe they won't agree with you

Second, Cloudera's article states Impala is 7x faster for interactive queries and 3x for others. Spark can still get faster and it is getting faster, but it would still be behind Impala IMO. However, it depends on implementation – badly implemented MPP would be worse than well implemented Batch, but I assume both Impala and Spark has good code quality and optimizations, and it would held for at least 2 years, so Spark would be slower here. And also, don't forget that Impala is at least 3x behind native MPP RDBMS solutions, just because of the penalty introduced by HDFS. So in total it is around 6x faster than MPP RDBMSs. All in all, 6x is not that much, but when you have a choice between deploying 10-machine cluster and 60-machine cluster you will look at 6x difference under a different angle

Third, if we speak about MPP RDBMSs specifically, it is not only about performance, but also about semantics – updates, ACID, transactions and the other good old RDBMS stuff

Baofeng Zhang
June 18, 2016 at 3:27 am

Very interesting point of view on the future of spark. I basically agree with you on many opinions, including what you discussing with Xin.

First, I want to mention the streaming things. Although Spark 2.0 introduced Structured Streaming, and if we truly know about streaming, it is obvious that the model is incomplete compared to Google DataFlow, which is the state of the art model as far as I can see in streaming. Besides, a pipelined runtime suits streaming for it gains latency(on record level). Spark is okay to build a prototype that does pipeline shuffle across stages, which seems not "Batch" any more, but it loses its advantage of the RDD abstraction. Also, it is due to the RDD, increment computing is unnatural to implement in spark streaming, which is provided as "updateStateByKey" previously. In fact, incremental update operation on state in streaming windows is complicated and important for streaming. Checkpoint is also a "Batch" mode in streaming 😕

Then, goes to the SparkSQL. I understand that RDD already becomes a "physical plan" representation and SparkSQL is not only SQL but the new runtime for spark execution. It is similar to DBMS, which it calls the single machine execution part the "Runtime", which including RBO/CBO, codegen things. I also understand why Batch vs. MPP does not gap that much, basically, it just two implementation of distributed runtime(mainly the task scheduling, data transferring, etc.), so SparkSQL is trying to integrate the good parts from DBMS into its execution model.

IMO, MPPs is model-based faster. It is not that bad if MPPs is fast, because MPP has its own disadvantage, which SparkSQL may not have. I am going to try to build a hybrid distributed runtime, which combines DAG and MPP, so that OLAP and OLTP is able to co-exist in one runtime. I will build the runtime in a streaming way, which means tasks are long-running and pre-launched. I am looking forward to see what you and Xin will write about "Batch and MPP" article 🙂

0x0FFF Post author
June 18, 2016 at 7:56 am

Great, thank you!
But a word of caution – be careful with combining OLTP and OLAP: it is not possible to make a good solution storing the data spindle HDDs for obvious reasons. But targeting SSD-only or RAM-only would be fine

Pingback: 为什么说Spark SQL远远超越了MPP SQL | 神刀安全网

gatorsmile
June 18, 2016 at 6:11 am

Based on my understanding, Spark SQL is a general-purpose data processing/analytics engine.

IMO, its future is like smartphones. You know, smartphones are not the best camera, not the best game console, not the best eBook reader, not the best music/video players. However, now, how many people are still buying camera, game console, MP3, and MP4? Here, no need to explain why.

In addition, Spark SQL is open. Many vendors are supporting it. It is like Android ecosystem. Because it is open, Android dominates the smartphone market. [However, the iPhone maker, Apple, is one of the biggest corporations. :)]

That is why I am very confident about the future of Spark.

0x0FFF Post author
June 18, 2016 at 8:04 am

You might be right here, it is all about chances. Another person mentioned in the comments that it is not always that the best system wins
I'm also pretty confident in Spark future – it is already too big to fall. What I wanted to highlight is that its direction is imperfect IMO and collides with MPP RDBMS market. However, Hadoop has spent around 10 years trying to get a share in this market, and I admit it more or less failed in doing so.

Zhan Zhang
June 19, 2016 at 6:48 am

Very interesting read and discussion. My understanding of spark is:
1. To developers, it is like programming language, like Java/c++, but with distributed computation native supported.
2. For end user, it is a tool for telling, SQL or machine learning.

The 1st is the real powerful part of spark, as @Reynold indicated, on top of it, spark can keep evolving.

Regarding the streaming, in my point of view, it is in the right direction. Catalyst engine have the foundation built, it is completely possible to change the underlying rdd dag batch processing to rdd dag streaming processing to realize the unification of batching and streaming system, although I do not know whether spark wil move to that direction.

With these said, comparing spark with mpp or other system does not cover the whole story as such comparison wil focus the the 2nd part but miss the 1st part.

gatorsmile
June 20, 2016 at 8:46 pm

I like the talk by Michael. It shows the vision and the direction for Spark streaming.
https://www.youtube.com/watch?v=SRlJs1VfGt8&feature=youtu.be

Really expect what Matei will invent for streaming.
