The Database Column
A multi-author blog on database technology and innovation.

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | TrackBacks (1)
[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we'll begin here with our views on MapReduce. This is a good time to discuss it, since the recent trade press has been filled with news of the revolution of so-called "cloud computing." This paradigm entails harnessing large numbers of (low-end) processors working in parallel to solve a computing problem. In effect, this suggests constructing a data center by lining up a large number of "jelly beans" rather than utilizing a much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to teach students how to program such clusters using a software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the MapReduce framework.

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represents a paradigm shift in the development of scalable, data-intensive applications. MapReduce may be a good idea for writing certain types of general-purpose computations, but to the database community, it is:

1. A giant step backward in the programming paradigm for large-scale data intensive applications
2. A sub-optimal implementation, in that it uses brute force instead of indexing
3. Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago
4. Missing most of the features that are routinely included in current DBMS
5. Incompatible with all of the tools DBMS users have come to depend on

First, we will briefly discuss what MapReduce is; then we will go into more detail about our five reactions listed above.

What is MapReduce?

The basic idea of MapReduce is straightforward. It consists of two programs that the user writes called *map* and *reduce* plus a framework for executing a possibly large number of instances of each program on a compute cluster.

The map program reads a set of "records" from an input file, does any desired filtering and/or transformations, and then outputs a set of records of the form (key, data). As the map program produces output records, a "split" function partitions the records into M disjoint buckets by applying a function to the key of each output record. This split function is typically a hash function, though any deterministic function will suffice. When a bucket fills, it is written to disk. The map program terminates with M output files, one for each bucket.

In general, there are multiple instances of the map program running on different nodes of a compute cluster. Each map instance is given a distinct portion of the input file by the MapReduce scheduler to process. If N nodes participate in the map phase, then there are M files on disk storage at each of N nodes, for a total of $N * M$ files; $F_{i,j},\ 1 \le i \le N,\ 1 \le j \le M$.

The key thing to observe is that all map instances use the same hash function. Hence, all output records with the same hash value will be in corresponding output files.

The second phase of a MapReduce job executes M instances of the reduce program, R_j, $1 \le j \le M$. The input for each reduce instance R_j consists of the files $F_{i,j},\ 1 \le i \le N$. Again notice that all output records from the map phase with the same hash value will be consumed by the same reduce instance -- no matter which map instance produced them. After being collected by the map-reduce framework, the input records to a reduce instance are grouped on their keys (by sorting or hashing) and feed to the reduce program. Like the map program, the reduce program is an arbitrary computation in a general-purpose language. Hence, it can do anything it wants with its records. For example, it might compute some additional function over other data fields in the record. Each reduce instance can write records to an output file, which forms part of the "answer" to a MapReduce computation.

To draw an analogy to SQL, map is like the *group-by* clause of an aggregate query. Reduce is analogous to the *aggregate* function (e.g., average) that is computed over all the rows with the same group-by attribute.

We now turn to the five concerns we have with this computing paradigm.

1. MapReduce is a step backwards in database access

As a data processing paradigm, MapReduce represents a giant step backwards. The database community has learned the following three lessons from the 40 years that have unfolded since IBM first released IMS in 1968.

- Schemas are good.
- Separation of the schema from the application is good.

- High-level access languages are good.

MapReduce has learned none of these lessons and represents a throw back to the 1960s, before modern DBMSs were invented.

The DBMS community learned the importance of schemas, whereby the fields and their data types are recorded in storage. More importantly, the run-time system of the DBMS can ensure that input records obey this schema. This is the best way to keep an application from adding "garbage" to a data set. MapReduce has no such functionality, and there are no controls to keep garbage out of its data sets. A corrupted MapReduce dataset can actually silently break all the MapReduce applications that use that dataset.

It is also crucial to separate the schema from the application program. If a programmer wants to write a new application against a data set, he or she must discover the record structure. In modern DBMSs, the schema is stored in a collection of system catalogs and can be queried (in SQL) by any user to uncover such structure. In contrast, when the schema does not exist or is buried in an application program, the programmer must discover the structure by an examination of the code. Not only is this a very tedious exercise, but also the programmer must find the source code for the application. This latter tedium is forced onto every MapReduce programmer, since there are no system catalogs recording the structure of records -- if any such structure exists.

During the 1970s the DBMS community engaged in a "great debate" between the relational advocates and the Codasyl advocates. One of the key issues was whether a DBMS access program should be written:

- By stating what you want - rather than presenting an algorithm for how to get it (relational view)
- By presenting an algorithm for data access (Codasyl view)

The result is now ancient history, but the entire world saw the value of high-level languages and relational systems prevailed. Programs in high-level languages are easier to write, easier to modify, and easier for a new person to understand. Codasyl was rightly criticized for being "the assembly language of DBMS access." A MapReduce programmer is analogous to a Codasyl programmer -- he or she is writing in a low-level language performing low-level record manipulation. Nobody advocates returning to assembly language; similarly nobody should be forced to program in MapReduce.

MapReduce advocates might counter this argument by claiming that the datasets they are targeting have no schema. We dismiss this assertion. In extracting a key from the input data set, the map function is relying on the existence of at least one data field in each input record. The same holds for a reduce function that computes some value from the records it receives to process.

Writing MapReduce applications on top of Google's BigTable (or Hadoop's HBase) does not really change the situation significantly. By using a self-describing tuple format (row key, column name, {values}) different tuples within the same table can actually have different schemas. In addition, BigTable and HBase do not provide logical independence, for example with a view mechanism. Views significantly simplify keeping applications running when the logical schema changes.

2. MapReduce is a poor implementation

All modern DBMSs use hash or B-tree indexes to accelerate access to data. If one is looking for a subset of the records (e.g., those employees with a salary of 10,000 or those in the shoe department), then one can often use an index to advantage to cut down the scope of the **search** by one to two orders of magnitude. In addition, there is a query optimizer to decide whether to use an index or perform a brute-force sequential **search**.

MapReduce has no indexes and therefore has only brute force as a processing option. It will be creamed whenever an index is the better access mechanism.

One could argue that value of MapReduce is automatically providing parallel execution on a grid of computers. This feature was explored by the DBMS research community in the 1980s, and multiple prototypes were built including Gamma [2,3], Bubba [4], and Grace [5]. Commercialization of these ideas occurred in the late 1980s with systems such as Teradata.

In summary to this first point, there have been high-performance, commercial, grid-oriented SQL engines (with schemas and indexing) for the past 20 years. MapReduce does not fare well when compared with such systems.

There are also some lower-level implementation issues with MapReduce, specifically skew and data interchange.

One factor that MapReduce advocates seem to have overlooked is the issue of skew. As described in "Parallel Database System: The Future of High Performance Database Systems," [6] skew is a huge impediment to achieving successful scale-up in parallel query systems. The problem occurs in the map phase when there is wide variance in the distribution of records with the same key. This variance, in turn, causes some reduce instances to take much longer to run than others, resulting in the execution time for the computation being the running time of the slowest reduce instance. The parallel database community has studied this problem extensively and has developed solutions that the MapReduce community might want to adopt.

There is a second serious performance problem that gets glossed over by the MapReduce proponents. Recall that each of the *N* map instances produces *M* output files -- each destined for a different reduce instance. These files are written to a disk local to the computer used to run the map instance. If *N* is 1,000 and *M* is 500, the map phase produces 500,000 local files. When the reduce phase starts, each of the 500 reduce instances needs to read its 1,000 input files and must use a protocol like FTP to "pull" each of its input files from the nodes on which the map instances were run. With 100s of reduce instances running simultaneously, it is inevitable that two or more reduce instances will attempt to read their input files from the same map node simultaneously -- inducing large numbers of disk seeks and slowing the effective disk transfer rate by more than a factor of 20. This is why parallel database systems do not materialize their split files and use push (to sockets) instead of pull. Since much of the excellent fault-tolerance that MapReduce obtains depends on materializing its split files, it is not clear whether the MapReduce framework could be successfully modified to use the push paradigm instead.

Given the experimental evaluations to date, we have serious doubts about how well MapReduce applications can scale. Moreover, the MapReduce implementers would do well to study the last 25 years of parallel DBMS research literature.

3. MapReduce is not novel

The MapReduce community seems to feel that they have discovered an entirely new paradigm for processing large data sets. In actuality, the techniques employed by MapReduce are more than 20 years old. The idea of partitioning a large data set into smaller partitions was first proposed in "Application of Hash to Data Base Machine and Its Architecture" [11] as the basis for a new type of join algorithm. In "Multiprocessor Hash-Based Join Algorithms," [7], Gerber demonstrated how Kitsuregawa's techniques could be extended to execute joins in parallel on a shared-nothing [8] cluster using a combination of partitioned tables, partitioned execution, and hash based splitting. DeWitt [2] showed how these techniques could be adopted to execute aggregates with and without group by clauses in parallel. DeWitt and Gray [6] described parallel database systems and how they process queries. Shatdal and Naughton [9] explored alternative strategies for executing aggregates in parallel.

Teradata has been selling a commercial DBMS utilizing all of these techniques for more than 20 years; exactly the techniques that the MapReduce crowd claims to have invented.

While MapReduce advocates will undoubtedly assert that being able to write MapReduce functions is what differentiates their software from a parallel SQL implementation, we would remind them that POSTGRES supported user-defined functions and user-defined aggregates in the mid 1980s. Essentially, all modern database systems have provided such functionality for quite a while, starting with the Illustra engine around 1995.

4. MapReduce is missing features

All of the following features are routinely provided by modern DBMSs, and all are missing from MapReduce:

- **Bulk loader** -- to transform input data in files into a desired format and load it into a DBMS
- **Indexing** -- as noted above
- **Updates** -- to change the data in the data base
- **Transactions** -- to support parallel update and recovery from failures during update
- **Integrity constraints** -- to help keep garbage out of the data base
- **Referential integrity** -- again, to help keep garbage out of the data base
- **Views** -- so the schema can change without having to rewrite the application program

In summary, MapReduce provides only a sliver of the functionality found in modern DBMSs.

5. MapReduce is incompatible with the DBMS tools

A modern SQL DBMS has available all of the following classes of tools:

- **Report writers** (e.g., Crystal reports) to prepare reports for human visualization

- **Business intelligence tools** (e.g., Business Objects or Cognos) to enable ad-hoc querying of large data warehouses
- **Data mining tools** (e.g., Oracle Data Mining or IBM DB2 Intelligent Miner) to allow a user to discover structure in large data sets
- **Replication tools** (e.g., Golden Gate) to allow a user to replicate data from on DBMS to another
- **Database design tools** (e.g., Embarcadero) to assist the user in constructing a data base.

MapReduce cannot use these tools and has none of its own. Until it becomes SQL-compatible or until someone writes all of these tools, MapReduce will remain very difficult to use in an end-to-end task.

In Summary

It is exciting to see a much larger community engaged in the design and implementation of scalable query processing techniques. We, however, assert that they should not overlook the lessons of more than 40 years of database technology -- in particular the many advantages that a data model, physical and logical data independence, and a declarative query language, such as SQL, bring to the design, implementation, and maintenance of application programs. Moreover, computer science communities tend to be insular and do not read the literature of other communities. We would encourage the wider community to examine the parallel DBMS literature of the last 25 years. Last, before MapReduce can measure up to modern DBMSs, there is a large collection of unmet features and required tools that must be added.

We fully understand that database systems are not without their problems. The database community recognizes that database systems are too "hard" to use and is working to solve this problem. The database community can also learn something valuable from the excellent fault-tolerance that MapReduce provides its applications. Finally we note that some database researchers are beginning to explore using the MapReduce framework as the basis for building scalable database systems. The Pig[10] project at Yahoo! Research is one such effort.

References

[1] "MapReduce: Simplified Data Processing on Large Clusters," Jeff Dean and Sanjay Ghemawat, Proceedings of the 2004 OSDI Conference, 2004.

[2] "The Gamma Database Machine Project," DeWitt, et. al., IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990.

[4] "Gamma - A High Performance Dataflow Database Machine," DeWitt, D, R. Gerber, G. Graefe, M. Heytens, K. Kumar, and M. Muralikrishna, Proceedings of the 1986 VLDB Conference, 1986.

[5] "Prototyping Bubba, A Highly Parallel Database System," Boral, et. al., IEEE Transactions on Knowledge and Data Engineering,Vol. 2, No. 1, March 1990.

[6] "Parallel Database System: The Future of High Performance Database Systems," David J. DeWitt and

Jim Gray, CACM, Vol. 35, No. 6, June 1992.

[7] "Multiprocessor Hash-Based Join Algorithms," David J. DeWitt and Robert H. Gerber, Proceedings of the 1985 VLDB Conference, 1985.

[8] "The Case for Shared-Nothing," Michael Stonebraker, Data Engineering Bulletin, Vol. 9, No. 1, 1986.

[9] "Adaptive Parallel Aggregation Algorithms," Ambuj Shatdal and Jeffrey F. Naughton, Proceedings of the 1995 SIGMOD Conference, 1995.

[10] "Pig", Chris Olston, http://research.yahoo.com/project/90

[11] "Application of Hash to Data Base Machine and Its Architecture," Masaru Kitsuregawa, Hidehiko Tanaka, Tohru Moto-Oka, New Generation Comput. 1(1): 63-74 (1983)

Categories

Database architecture , Database history , Database innovation

Tags

- database performance
- DeWitt
- MapReduce
- Stonebraker

1 TrackBacks

Listed below are links to blogs that reference this entry: MapReduce: A major step backwards.

TrackBack URL for this entry: http://www.databasecolumn.com/blog/mt-tb.cgi/26

» This is one of the web's most interesting stories on Fri 18th Jan 2008 from purrl.net |** urls that purr **|

These are the web's most talked about URLs on Fri 18th Jan 2008. The current winner is .. Read More

Tracked on January 18, 2008 12:09 AM

44 Comments

Ronn Brashear said:

As an MR advocate, I can agree with several of the above points. Certainly, MR development shouldn't ignore previous research, nor should it be constrained by it. MR is directed at a different problem from the modern DBMS.

For example, using MR to rapidly identify small subsets of data is a bad idea. However MR is a good large-data manipulation tool - something for which DBs are notoriously bad. A grid DB's indexing offers no advantage when computing page rank of the internet for example. Indices are pure overhead in that situation.

I am concerned the authors are suggesting that introducing MR into academia is a bad idea since *that is where most of the previous literature is well understood*. Some of the best improvements to MR lately have been based on distributing reductions ala Monet's continuous near-neighbor load distribution. To say MR doesn't have high level languages/tools/optimizations is short-sighted. Pig, Sawzall, and others functional languages are in development. Additional tools, research, and optimization will follow. Presenting MR as a research topic will enable that growth.

For engineers, the underlying issue is picking the right tool for the job. RDB versus Flat Files versus MQL versus MR smacks of the same "religious" debates between Java versus C++ versus Ruby versus assembly and is generally a waste of effort. A good engineer understands the specific problem space, examines the potential solutions, and picks the right tool for the job.

January 17, 2008 6:52 PM
ade said:

You seem to be under the impression that MapReduce is a database. It's merely a mechanism for using lots of machines to process very large data sets. You seem to be arguing that MapReduce would be better (for some value of better) if it were a data warehouse product along the lines of TeraData. Unfortunately the resulting tool would be less effective as a general purpose mechanism for processing very large data sets.

You seem to have made a category error in this article: http://en.wikipedia.org/wiki/Category_mistake

January 17, 2008 7:37 PM
Joe Hellerstein said:

As a wise philosopher once said, *Be a lover, not a fighter!*

Technically, I agree with much of this article, especially the history lesson on parallel data processing. (I'd even go one further on you w.r.t.the fault-tolerance aspects of MapReduce, where I wish they had acknowledged Mehul Shah's work on FLuX, which provides fault-tolerance, load balancing *and* pipelined processing.)

But none of that matters. It's all about hearts and minds, and if the DB industry adopts the attitude in this article, it's back to what I said at HPTS back in 2001: We Lose. See especially slides 5-10.

January 17, 2008 8:19 PM
Toby DiPasquale said:

I would point out that Google created MapReduce because some (most?) of the inputs into the GFS to be used with MapReduce were already "garbage", in that they are semi-structured data (e.g. the Web). As such, a schema would have been too restrictive for them to have made the progress in data mining that they have. (this ignores their entry into the column-oriented database space, BigTable) Also, the "key" you speak of in Web data turns out to be the URL of the document, something not inherent to the data at all.

For Google and others like them, a custom solution is sometimes preferable to what the industry standard practice is doing. Recall the time during which Google created this model: the storage on commodity PCs was much smaller than it was today, especially in relation to the size of the Web, and they needed considerably more horsepower to perform the machinations of a **search** engine than was available on even the largest machines of the day. Thus, they were forced them to create a system that managed large numbers of machines to work effectively in concert. No such system existed at that time in the database world. Oracle was limited to 32 nodes at that time.

Also, MapReduce is an extremely simple programming model and allows even interns to produce useful programs for data mining and reporting. But, those same interns are likely *not* working on the inputs to said system. This works for them because the data MapReduce is working on is read-only.

GFS + MapReduce is missing a lot of the features you mention on purpose but some are available in BigTable (views, in particular, just like your Vertica database). The entire point of the GFS/MapReduce system from its original design was to build the inverted index of the Web, so I'd hardly say that it lacks indexing. Google itself is the index. As well, you can update data in GFS after writing, although this is discouraged. (its right there in the GFS paper)

As to your "doubts" as to how well MapReduce can scale, I'm having trouble believing that you could honestly have such doubts. Google has dozens of clusters in the tens of thousands of machines, all of which crawl and index the Web independently of each other at regular intervals with GFS, MapReduce and BigTable. MapReduce was designed with horizontal scalability in mind, first and foremost. You can complain legitimately about the performance (of Hadoop, anyway) but the scalability is there in spades.

Finally, Google has a high-level language to access this data called Sawzall and the Hadoop community is forging its own language for this purpose called Pig. (http://incubator.apache.org/pig/)

I feel as if this post has misrepresented the MapReduce model and the problems it was designed to solve. No one at Google or working on Hadoop would tell you that MapReduce is a replacement for a generic RDBMS but is rather designed for a specific set of issues and constraints. This is a valid method of doing work and does not deserve to be criticized out of context.

January 17, 2008 11:11 PM
JS said:

While I don't want to sound critical, how can you make such an absurd claim as: "Given the experimental evaluations to date, we have serious doubts about how well MapReduce applications can scale." The most recent information from Google is that they're running MapReduce on 20 petabytes of data a day(1). That's larger than any other data sets I'm aware of. MapReduce is also designed to deal with heterogeneous data sets, something not compatible with the relational data model where uniformity of records is expected.

(1) "MapReduce: simplified data processing on large clusters" in Communications of the ACM Volume 51 , Issue 1 (January 2008)

January 17, 2008 11:18 PM
Daniel said:

So what do YOU propose as a valid schema and index system, using Teradata or a similar DBMS, for what is essentially a full-text indexing system? And how much would such a system cost (including yearly support contracts) Google?

They're looking through unstructured data for specific words that match a query, and they're doing it under pretty strict OLTP constraints. Nevermind that mapReduce is run all the time to update the pagerank.

So what do you propose, other than hand waving? This is a entirely different problem domain than DBs solve. Page searching doesn't even require most of ACID or worry about constraints or foreign keys.

January 17, 2008 11:35 PM
Greg Jorgensen said:

When I finished reading the article I was thinking that the authors did not understand MapReduce or the idea of data in the cloud ... if you change "MapReduce" to "SimpleDB" the original article almost makes sense.

More at Relational Database Experts Jump The MapReduce Shark.

January 18, 2008 12:10 AM
Sorin Gherman said:

> Missing most of the features that are routinely included in current DBMS
> Incompatible with all of the tools DBMS users have come to depend on

With really large datasets and distributed sytems the RDBMS paradigms stop working, and that's where a system like Mapreduce is needed.
Distributed systems *interfaces* are dumb and simple-minded on purpose: there is no way to index data arbitrarily as in DBMS, and do arbirary joins on data like in SQL.
With really large and open data, one has to shift their paradigm away from DBMS and SQL and arbirary indexes: these simply don't work in distributed systems.

January 18, 2008 12:16 AM
DAR said:

Hmmmm ... although I appreciate you guys addressing this topic (I was the one who suggested it), frankly this column left me scratching my head a bit. It looks to me like there's some incorrect information here about some key aspects of these distributed DB's.

First off, from everything I've read, it appears that they are a completely separate technology from map/reduce. (For example, if you read the Google BigTable paper at http://labs.google.com/papers/bigtable-osdi06.pdf, the only place they even seem to mention map/reduce is where they state that a BigTable database can be used as a data source for an external map/reduce job.)

Second, BigTable *does* have indexes ("The map is indexed by a row key, column key, and a timestamp") and therefore *doesn't* require brute force.

So although some of your criticisms might be valid here (e.g., lack of schemas, lack of high level language, lack of advanced features and tool support, etc.) several others seem somewhat off-base.

I have no axe to grind here. (I don't work for Google or anything.) I'm just a developer who finds this stuff interesting and wants to learn more about it. But it just feels to me like these DB's didn't really get a fair comparison here.

January 18, 2008 12:43 AM

Robert Weisman said:

MapReduce is not a DBMS. It is a framework for developing distributed systems. I can't quite imagine why such a framework would have indexes, or why the concepts of updates, integrity, transactions, views, or bulk loaders would even be meaningful. God forbid, one could even write a MapReduce to read from a DBMS. Some of these criticisms may be valid with Bigtable, but then Bigtable is probably the only DBMS designed to handle petabytes of data.

January 18, 2008 1:25 AM
Ilya Haykinson said:

I think your arguments are solid in that MapReduce represents a step backwards compared to a traditional DBMS. However, I believe that your overall point is severely off the mark, since I believe that your comparison is not entirely fair.

MapReduce is not a database framework. Instead, it's a computational framework. Unlike a database, it does not offer storage of data, or transactions, or indeed any sort of a query language. To see MapReduce as a database system is to seriously miscategorize its use.

Additionally, you write that there's a question of MapReduce's ability to scale. I think that Google's track record here is a great testament to the system's abilities: with hundreds of thousands of nodes, the system seems to be battle-tested.

Instead of looking at MapReduce you may want to look at BigTable -- Google's database technology. I think that it's a more fair comparison and probably deserves a thorough review.

January 18, 2008 2:33 AM
Joe Developer said:

Mapreduce's charm is, I gather, a combination of
excellent fault tolerance and utter
simplicity. There isn't a whiff of database
about it, it's more like a simple pipe.
And sometimes a pipe is all you need.

January 18, 2008 3:57 AM
gasper_k said:

Hi,

MapReduce isn't meant to replace a RDB; it doesn't need indices, ordering, grouping and practically everything you wrote in items 4 and 5. MapReduce is design to work by iterating over data in the given order and producing an output. Also, you can have a schema and data validation just as well, so again it doesn't fall short.

As for item 3, MapReduce not being novel is hardly an argument against it, is it?

Basically, what you're saying is, you should use MapReduce because:
- it doesn't have features it doesn't need,
- it isn't a new concept,
- some databases (not many, though) can do what it does,
- requires a lower level view of the application and data (this one being the only solid argument).

Not being an advocate for MapReduce, but your argumentation fails on many levels.

Best regards,
Gasper

January 18, 2008 4:14 AM
James A. said:

By stating what you want - rather than presenting an algorithm for how to get it (relational view)

We use mapreduce to obtain results and perform computation that RDBMS cannot perform. For example, what relational command would you use to extract all urls from four billion documents then collect into separate lists all urls matching certain patterns (movie-like urls, image-like urls, html-like urls) by site? The site should be determined by host-level chunking the rules of which are a combination of data and programming logic.

This hypothetical task (no mapreduce in Google that I know of performs it, although there are ones like it but much more complex) is simple for a mapreduce, would be impossible to write in a relational query-language. What would be more impossible for an RDBMS is performing this calculation on 30 TB of input data in under three hours for less than $5 million capital invested.

[Section on skew]

As if we don't have devices that solve this problem. Simply selecting a uniformly distributed key scheme is enough to get around it.

Given the experimental evaluations to date, we have serious doubts about how well MapReduce applications can scale.

Have you experimentally evaluated mapreduce at all? If so, how?

The MapReduce community seems to feel that they have discovered an entirely new paradigm for processing large data sets.

This would be a wonderful thing to have a citation for. Mostly the sense I get is that MapReduce allows complex processing on large datasets without the programming difficulty required in other mediums. Novelty is not on the feature list.

MapReduce is missing features

You seem to not have noticed that mapreduce is not a DBMS. Same goes for section five.

Overall this article represents a profound and surprising misunderstanding of what MapReduce is. Perhaps next time they evaluate a product the authors could be bothered to learn what that product is?

January 18, 2008 5:36 AM
steppres said:

Perhaps you guys should read this...

Relational Database Experts Jump The MapReduce Shark

At least *somebody* know what they're talking about when it comes to MapReduce.

January 18, 2008 6:09 AM
masukomi said:

Our friend the Typical Programmer points out why this article shows a complete mis-understanding of what MapReduce is or is for. Amongst other things, that it's not intended, in any way, to be a system for storing and managing structured data, and thus **not a database.**

http://typicalprogrammer.com/programming/mapreduce/

January 18, 2008 8:02 AM
Tom Ritchford said:

I respectfully disagree. I use MapReduce every day and *for what it does* it's the best.

There are two advantages that you missed.

1. If you set it up properly, the records get sorted and appear in the reducer in sorted order, for free!

but even more important:

2. MapReduce is extremely light.

It doesn't mean that a MapReduce won't use a lot of machines -- it means that you can run a MapReduce you already have on brand-new data in a few minutes, and you can write a brand-new one, run it and get good output in an afternoon -- because you don't have to load up a database.

January 18, 2008 11:13 AM
Chris Olston said:

It is unfortunate that so many people are focusing on superficial distinctions between map-reduce and databases. The point here is that if you examine the underlying techniques, there are very strong similarities (although there are some key differences as well).

If we can focus on the areas of overlap, we can foster a much more productive relationship. There's no need to be adversarial on this, as Joe Hellerstein points out. Both communities bring ideas and experiences that can benefit the other.

January 18, 2008 12:21 PM
NAC said:

This is the basic problem with database people. They view everything
as a database, and that everything must be done in/with a database.

MapReduce is for a different class of problems. eg if you were
using MR to process large quantities of small image tiles, how
would you forumulate that in teradata, what advantage does an
RDBMS bring?

Relational databases are as much an inhibitor to modern application
development as they are an asset. Hence the massive use of
complex object relational mappers to work around their problems.

All technologies have strengths and weaknesses, MR and databases included. These are two different things, don't confuse them.

January 18, 2008 4:32 PM
Chris G said:

I think you could write an equivalent article:
"Airplanes: A major step backward"
You could say something like, "roads are good, everyone knows that, why would you throw them away?"
You could talk about fuel consumption, possible crashes, etc.
For the knock out punch, give an example why cars are definitely better. Describe taking your kids to school in a plane would be ridiculous; a car is so much better.
I don't think anyone seriously suggests replacing all databases with MR. It'd be a terrible solution for a small database. Seems your trying to sell it for something it's not. I do mean "selling" too. I see that Vertica Systems is behind the page. When Vertica is running Google, let me know. I'll get in line for the product.

January 18, 2008 5:05 PM
Greg Grasmehr said:

Interesting opinion and set of comments thus far; thanks to everyone who has provided input. Interesting reading to say the least.

January 18, 2008 5:19 PM
Stephan Wehner said:

Could one of the bonus points of MapReduce be that "It works" or "It doesn't cost much"?

Stephan

January 18, 2008 6:24 PM
mypalmike said:

Some interesting quotes from the article that show what I think is a fundamental misunderstanding of the subject matter:

"To draw an analogy to SQL, map is like the group-by clause of an aggregate *query*."

"[Relational databases have] a *query* optimizer to decide whether to use an index or perform a brute-force sequential **search**... MapReduce has no indexes and therefore has only brute force as a processing option."

"...skew is a huge impediment to achieving successful scale-up in parallel *query* systems"

"It is exciting to see a much larger community engaged in the design and implementation of scalable *query processing* techniques."

Apparently, the authors of this article believe that MapReduce is a process for *querying* unstructured data. It is not, or at least, I've never heard it being touted as such. It is a process for categorizing (Map) and aggregating (Reduce) the records in an unindexed data stream. In common use, the MapReduce process is run exactly *once* on a set of data. Really, the Google paper on the subject is quite clear on this.

January 18, 2008 7:17 PM
Steve Severance said:

Even google does not claim that the idea of map/reduce is novel. If you read the paper and watch the videos they have for interns it is clear that is grounded in functional programming. Map/reduce is simply an elegant implementation of those concepts that accomplish something very useful.

I agree with about all the other criticisms so I will not opine.

Steve

January 18, 2008 7:30 PM
ajfabb said:

Your statement that the MapReduce community claims to have invented this stuff is obviously wrong: anyone with a decent computer science background knows that Map and Reduce are ancient funtional programming primitives for operating on lists. The MR people are simply taking advantage of the trivial parallelization these primitives offer--which has been well known for decades.

January 18, 2008 7:38 PM
Alex Rasmussen said:

As one of the authors of Berkeley's MR-based curriculum, let me say that the choice of MapReduce as a framework was primarily motivated by its simplicity and its public exposure (OK, and the fact that Google was sponsoring it, but that's beside the point). We wanted to introduce students to the general idea of parallelism in a way that would get their attention - saying "Google uses this" definitely gets their attention.

We have implemented a glue layer between Berkeley's Scheme interpreter and Hadoop that allows students to express mapreductions in Scheme, a language with which they become familiar during the course. Just as we use Scheme as a tool for teaching students about the power of recursion, we use MapReduce as a tool for teaching students who don't know a whole ton about computer science yet about the power of parallelism. We don't present MapReduce as anything shockingly innovative - far from it. We show that MapReduce represents an application of operations with which students are already familiar (map and reduce) over a bunch of data in parallel, which makes both what they learned and what they're learning seem a lot more relevant.

January 19, 2008 12:03 AM
Ashwin said:

This article, coming as it does from such eminent folks from academia, just goes on to show how dogmatic some of the academic community has become. Or is it just that the earlier world / status-quo has been shattered so much that you can't bear the success of a newer system? This is truly what academia should NOT be.

To my knowledge (and I'm a CS PhD student at a very reputable university), the MapReduce folks have never claimed it is applicable to a wide variety of systems, even though in reality, it very well could be. This article starts with the assumption that MapReduce is a new DBMS, which has never been a claim. They seem to be genuinely trying to explore a new paradigm... At the same time, a lot of folks out in the real world are beginning to realize that relational schemas aren't really the cure for all diseases either as you seem to claim.

Or, as the cynic in me says, maybe this is a ploy to advance some of your other business ambitions?

January 19, 2008 12:27 AM
Mats Helander said:

I'm afraid I don't get it.

I must begin by stating that I have no clue on MapReduce other than what I grabbed from your blog post. Now, given that:

You say that a problem with MapReduce is that there's no schema. Then you go on to say that MapReduce is like Group By plus aggregates in SQL.

But a view or a stored procedure with Group By and aggregates have no schemas in an RDBMS either - right?? I agree that the definition of a view in SQL (a bunch of SQL statements) can be considered declarative (what) rather than imperative (how) in nature, but there's really no schema for it, is there? This may only go to reveal my poor understanding of RDBMS, but...you'd have to go in and check the SQL in the view to see what columns it returns, right? And so if we see MapReduce as a function over the data, much like stored procedures, there's really no issue with missing schema...or what am I missing?

Put another way, couldn't the MapReduce function work over an RDBMS that used a strict schema for all its data even while no such schema exists for describing neither the group by + aggregate functions in the stored procedures/views in the RDBMS nor the equivalent operations in the MapReduce layer?

/Mats

January 19, 2008 4:14 AM
Jim White said:

Which RDBMS allows computations to complete successfully in the presence of nodes that catch fire?

January 19, 2008 9:40 AM
Jim Kellerman said:

Quite a change in position for the authors. As recently as October 2007, Michael Stonebraker presented his paper The End of an Architectural Era (It's time for a Complete Rewrite at HPTS.

With respect to other comments about performance, Google's Map/Reduce framework has been around since 2003, and work on Bigtable started in 2004. Google's Distributed File System predated Map/Reduce. Since the initial publication of their papers on these topics, development of these systems has continued and Google now uses Bigtable to serve live data on their site which has changed the performance and availability requirements significantly. Hadoop is considerably younger (the project started in March of 2005) and HBase did not even exist a year ago.

January 19, 2008 12:12 PM
Denis Altudov said:

Kudos to the authors for providing a well-researched article. The point itself is debatable as evidenced by heated comments, but I think it is important for us to get both historical and adjacent field perspectives on modern technologies - reusing other people research is the cornerstone of progress. I also think that people who have been in the industry for a long time are uniquely positioned to give us this perspective and this article is one of the best ways Michael and his friends can contribute to advancement of the art.

I have certainly enjoyed it, more than the usual praises sung to column stores on this blog and posts like these is what is needed to turn this blog into wider new db technologies discussion forum.

January 19, 2008 2:23 PM
xbar said:

You are right that GFS + BigTable + MapReduce are not equivalent to an RDMBS + SQL + its ecosystem of tools and applications.

However, there is NO database out there (whether academic or commercial) that can scale to tens of thousands of processors and petabytes of unstructured/semi-structured data, and can run on commodity hardware (without any RAID/SAN etc), and can do so efficiently and cost-effectively without costing a fortune for licensing and support and administration. Not to mention the fact that it will take more than just some pie-in-the-sky academic research to actually engineer and implement a reliable query executor and optimizer that can translate arbitrary SQL into an efficient plan for this scale of operation.

You can quote the last 25 years of parallel database literature all you want, but the reality is that while GFS + BigTable + MapReduce may not be the ultimate answer, it is induced by the constraints of the problem that Google is working on: processing large datasets efficiently and cheaply.

January 19, 2008 4:02 PM
Rob McCool said:

What I don't quite understand is why these sorts of "damn kids better get off my lawn" essays keep getting written. Tenenbaum vs Torvalds is one example. Tenenbaum was both correct and irrelevant.

The views expressed here are both correct and irrelevant. If MapReduce is seeing widespread excitement and adoption, it's either because the 20 year old efforts described in this essay were before their time, or because the market rejected them for some reason.

It would be much more productive to either suggest how these techniques could be applied to MapReduce, in specific terms, or alternatively it would be productive to examine why the market has rejected solutions as described in this essay. Cost and complexity are two factors I think are worth a debate.

I worked in academia for a while, and I really liked Feigenbaum's approach to these sorts of things. His group was doing things in the 80's that are only today being widely understood and adopted. But instead of arguing about how primitive modern techniques were compared to his work, he always worked with the young upstarts who were exploring an area that was new to them but old to academia, and gently guided them in the right direction without judging or bragging.

January 20, 2008 12:58 PM
Ali Dasdan said:

MapReduce was not developed to replace DBMSs at all; it was developed to satisfy a need. As we reviewed in the map-reduce-merge paper,
search engine companies (ask, google, and yahoo) independently developed similar frameworks for simple, fast, and reliable processing of huge data sets. It is not the end, just a useful enabler.

MapReduce may have lots of deficiencies when viewed under a different light but we cannot ignore the fact that it works so successfully. In fact, the reasons for its success are very similar to the points made in Stonebroker's 2nd paper on "one size fits all?".

I think with more constructive support from the database and systems communities, it will mature into a better framework. There is already lots of work on improving and generalizing it, e.g., see dryad from microsoft, sawzall from google, pig from yahoo, hadoop from apache, map-reduce-merge (the paper cited above) from yahoo, and phoenix from stanford.

January 20, 2008 6:23 PM
Gordon Linoff said:

My background is in parallel processing, parallel databases, and data mining. I have recently written a book called "Data Analysis Using SQL and Excel", because I do strongly believe that relational databases can be used for sophisticated data analysis, even for non-programmers.

I found this posting because one company that I know of is in the process of deciding to use Hadoop and MapReduce for their data warehouse. For many of the reasons outlined by Profs Stonebreaker and Dewitt, I remain skeptical about this approach, although the particular company is determined to follow this path.

I do observe that there is considerable agreement between the original article and many of the critics. Everyone, including the original authors, state emphatically that the MapReduce framework is not a replacement, nor intended as a replacement, for relational databases.

And that may be true technically. However, in the marketplace, both are trying to solve the problem of analyzing large amounts of data. That is, both technologies compete in the market, even though they are quite different. In a similar way, when we have free time and money, we can play a computer game, go to a movie, read a book, or dine in a restaurant -- even though computer games, movies, books and restaurants are not traditionally thought of as "competing" against each other.

I do think that Profs Stonebreaker and Dewitt do miss some important points:

(1) For most analytic purposes, indexes are not useful. Full table scans should simply be assumed.

(2) The transactional integrity parts of relational technology are not generally useful for complex queries, since SELECT (as opposed to UPDATE or INSERT) is the most common SQL statement for analysis purposes.

(3) Although some databases do have analytic tools built-in, serious analysis generally use more serious statistical tools such as SAS, SPSS, S-Splus, or R.

MapReduce is a new paradigm (even if it is an old technology). Undoubtedly, it can solve many problems, in a fault tolerant way on large amounts of data. It is not a replacement for relational databases. It is a paradigm for developing parallel programs, not a general purpose solution for managing and querying data.

--gordon
Gordon S. Linoff, Founder
Data Miners, Inc.
gordon@data-miners.com
Author of:
"Data Analysis Using SQL and Excel"

"Data Mining Techniques for Marketing, Sales, and Customer Support"
"Mastering Data Mining: The Art and Science of Customer Relationship Management"
"Mining the Web"

January 22, 2008 2:40 PM
D L Childs said:

Its rather sad that such dialogues protecting the myth of index structure performance still persist. If one is ignorant of jet-engine technology then advocating prop-engine technology over hot-air-balloon technology should be expected. Though Selective Set Retrieval I/O (with 90-98% informationally dense I/Os) provides better performance than index structures (with 2-3% informationally dense I/O's) under ALL conditions, the following statement has to be true.

"Any system that has no indexes and therefore has only brute force as a processing option, will be creamed whenever an index is the better access mechanism."

The operative word here is "whenever". Notice also that the "therefore" depends on a selection from two choices, not a selection from three choices.

For a paper of possible interest on index structure antiquity, see http://xsp.xegesis.org/Pebbles.pdf.

January 23, 2008 12:40 PM
Zach said:

I think maybe you want to replace all instances of "MapReduce" with "SimpleDB" in this article. As Greg Jorgensen said, it almost makes sense that way.

January 23, 2008 3:16 PM
M. Ibrahim said:

Who said Map/Reduce is meant to be a relational database? Where did you get this idea from?

January 24, 2008 9:03 AM
Admin said:

Thanks for all your comments. If you are just reading this post, or are planning a comment, you might want to first read the authors' follow up post here and submit your comments to that post.

- The Database Column Editors

January 25, 2008 4:38 PM
K. Wu said:

I had a first-hand experience where MapReduce and BigTable was used to justify the choice of abandoning schema. This is definitely a unintended consequence of the rising awareness of MapReduce -- at least I hope it is just a unintended consequence. Having a schema for data is very important for efficiency of data processing. Abandoning schema will invariably require the client software somehow rediscover it.

March 16, 2008 12:42 AM
Anonymous said:

Not all MPP database vendors agree with Dewitt and Stonebraker. For example, Aster Data just launched the world's first In-Database MapReduce:

http://www.asterdata.com/product/mapreduce.html

August 25, 2008 8:44 PM
Michael Mullany said:

Here at Aster, we've just launched a sneak peek of our In-Database Map-Reduce capability with a whitepaper and demo.

Our firm belief is that the attractive thing about MapReduce is the programming model -- far superior to traditional database user-defined functions & PL/SQL because it's easy to learn but conceptually powerful/flexible. Why not put that programming model to work within an MPP relational database -- retaining the familiarity and powerful of SQL & the other benefits of RDBMS.

Love to hear your thoughts:
http://www.asterdata.com/blog/index.php/2008/08/25/announcing-in-database-mapreduce/

August 25, 2008 8:53 PM
Anonymous said:

See CloudBase-
http://cloudbase.sourceforge.net

It is a data warehouse system built on top of Hadoop's Map Reduce architecture that allows one to query Terabyte and Petabyte of data using ANSI SQL. It comes with a JDBC driver so one can use third party BI tools, reporting frameworks to directly connect to CloudBase.

CloudBase creates database system directly on flat files and converts input ANSI SQL expressions into map-reduce programs for processing flat files. It has an optimized algorithm to handle Joins and plans to support table indexing in next release.

December 29, 2008 1:45 PM
Horia Margarit said:

Map and Reduce are indeed general purpose functions, which means they can be implemented in a high-level language as well as a low-level language. So the paradigm itself is not akin to programming in Assembly, because you can write a MapReduce database in a high-level language if you so choose.

Furthermore, even if you pick a *very* high-level, general purpose language like Common Lisp, you can still control your data types and variable declarations by the use of a dedicated function, which you may call Schema.

This Schema function would be passed as an argument to the function which is in turn Mapped onto the Keys in your MapReduce example.

In short, I do not see any validity to your claims in point 1.

June 17, 2009 9:12 PM

Leave a comment

Name

Email Address

URL

Remember personal info?

Comments (You may use HTML tags for style)

Preview Submit

Search

Search

About this Entry

This page contains a single entry by David DeWitt published on *January 17, 2008 4:20 PM*.

Relational databases for storing and querying RDF was the previous entry in this blog.

MapReduce II is the next entry in this blog.

Find recent content on the main index or look in the archives to find all content.

Subscribe to this blog's feed

Powered by Movable Type Publishing Platform