Chapter 5: Large-Scale Dataflow Engines

Introduced by Peter Bailis

Selected Readings:

Jeff Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *OSDI*, 2004.

Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu. DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. *OSDI*, 2008.

Of the many developments in data management over the past decade, MapReduce and subsequent large-scale data processing systems have been among the most disruptive and the most controversial. Cheap commodity storage and rising data volumes led many Internet service vendors to discard conventional database systems and data warehouses and build custom, home-grown engines instead. Google's string of publications on their large-scale systems, including Google File System [10], MapReduce, Chubby [6], and BigTable [7], are perhaps the most famous and influential in the market. In almost all cases, these new, homegrown systems implemented a small subset of the features found in conventional databases, including high-level languages, query optimizers, and efficient execution strategies. However, these systems and the resulting open source Hadoop ecosystem proved highly popular with many developers. This led to considerable investment, marketing, research interest, and development on these platforms, which, today are in flux, but, as an ecosystem, have come to resemble traditional data warehouses—with some important modifications. We reflect on these trends here.

History and Successors

Our first reading is the original Google MapReduce paper from 2004. MapReduce was a library built for simplifying parallel, distributed computation over distributed data at Google's scale—particularly, the batch rebuild of web search indexes from crawled pages. It is unlikely that, at the time, a traditional data warehouse could have handled this workload. However, compared to a conventional data warehouse, MapReduce provides a very low-level interface (two-stage dataflow) that is closely tied to a fault-tolerant execution strategy (intermediate materialization between two-stage dataflow). Equally importantly, MapReduce was designed as a library for parallel programming rather than an end-to-end data warehousing solution; for example, MapReduce delegates storage to Google File System. At the time, members of the database community decried the architecture as simplistic, inefficient, and of limited use [8].

While the original MapReduce paper was released in 2003, there was relatively little additional activity external to Google until 2006, when Yahoo! open-sourced the Hadoop MapReduce implementation. Subsequently, there was an explosion of interest: within a year, a range of projects including Dryad (Microsoft) [15], Hive (Facebook) [26], Pig (Yahoo) [22] were all under development. These systems, which we will call post-MapReduce systems, gained considerable traction with developers—who were largely concentrated in Silicon Valley—as well as serious VC investment. A multitude of research spanning the systems, databases, and networking communities investigated issues including scheduling, straggler mitigation, fault tolerance, UDF query optimization, and alternative programming models [5].

Almost immediately, post-MapReduce systems expanded their interface and functionality to include more sophisticated declarative interfaces, query optimization strategies, and efficient runtimes. Today's post-MapReduce systems have come to implement a growing proportion of the feature set of conventional RDBMSs. The latest generation of data processing engines such as Spark [27], F1 [24], Impala [16], Tez [1], Naiad [21], Flink/Stratosphere [2], AsterixDB [3], and Drill [14] frequently i) expose higher-level query languages such as SQL, ii) more advanced execution strategies, including the ability to process general graphs of operators, and iii) use indexes and other functionality of structured input data sources when possible. In the Hadoop ecosystem, dataflow engines have become the substrate for a suite of higher-level functionality and declarative interfaces, including SQL [4, 26], graph processing [12, 19], and machine learning [11, 25]. There is also increasing interest in stream processing functionality, revisiting many of the concepts pioneered in the database community in the 2000s. A growing commercial and open source ecosystem has developed "connectors" to various structured and semi-structured data sources, catalog functionality (e.g., HCatalog), and data serving and limited transactional capabilities (e.g., HBase). Much of this functionality, such as the typical query optimizers in these frameworks, is rudimentary compared to many mature commercial databases but is quickly evolving.

DryadLINQ, our second selected reading for this section, is perhaps most interesting for its interface: a set of embedded language bindings for data processing that integrates

seamlessly with Microsoft's .NET LINQ to provide a parallelized collections library. DryadLINQ executes queries via the earlier Dryad system [15], which implemented a runtime for arbitrary dataflow graphs using a replay-based fault tolerance. While DryadLINQ still restricts programmers to a set of side-effect free dataset transformations (including "SQL-like" operations), it presents a considerably higher-level interface than Map Reduce. DryadLINQ's language integration, lightweight fault tolerance, and basic query optimization techniques proved influential in later dataflow systems, including Apache Spark [27] and Microsoft's Naiad [21].

Impact and Legacy

There are at least three lasting impacts of the MapReduce phenomenon that might not have occurred otherwise. These ideas are – like distributed dataflow itself – not necessarily novel, but the ecosystem of post-MapReduce dataflow and storage systems have broadly increased their impact:

1.) *Schema flexibility.* Perhaps most importantly, traditional data warehouse systems are walled gardens: ingested data is pristine, curated, and has structure. In contrast, MapReduce systems process arbitrarily structured data, whether clean or dirty, curated or not. There is no loading step. This means users can store data first and consider what to do with it later. Coupled with the fact that storage (e.g., in the Hadoop File System) is considerably cheaper than in a traditional data warehouse, users can afford to retain data for longer and longer. This is a major shift from traditional data warehouses and is a key factor behind the rise and gathering of "Big Data." A growing number of storage formats (e.g., Avro, Parquet, RCFile) marry semi-structured data and advances in storage such as columnar layouts. In contrast with XML, this newest wave of semi-structured data is even more flexible. As a result, extract-transform-load (ETL) tasks are major workload for post-MapReduce engines. It is difficult to overstate the impact of schema flexibility on the modern practice of data management at all levels, from analyst to programmer and analytics vendor, and we believe it will become even more important in the future. However, this heterogeneity is not free: curating such "data lakes" is expensive (much more than storage) and is a topic we consider in depth in Chapter 12.

2.) *Interface flexibility.* Today, most all users interact with Big Data engines in SQL-like languages. However, these engines also allow users to program using a combination of paradigms. For example, an organization might use imperative code to perform file parsing, SQL to project a column, and machine learning subroutines to cluster the results – all within a single framework. Tight, idiomatic language integration as in DryadLINQ is commonplace, further improving programmability. While traditional database engines historically supported user-defined functions, these new engines' interfaces make user-defined computations simpler to express and also make it easier to integrate the results of user-defined computations with the results of queries expressed using traditional relational constructs like SQL. Interface flexibility and integration is a strong selling point for data analytics offerings; the ability to combine ETL, analytics, and post-processing in a single system is remarkably convenient to programmers — but not necessarily to users of traditional BI tools, which make use of traditional JDBC interfaces.

3.) *Architectural flexibility.* A common critique of RDBMSs is that their architecture is too tightly coupled: storage, query processing, memory management, transaction processing, and so on are closely intertwined, with a lack of clear interfaces between them in practice. In contrast, as a result of its bottom-up development, the Hadoop ecosystem has effectively built a data warehouse as a series of modules. Today, organizations can write and run programs against the raw file system (e.g., HDFS), any number of dataflow engines (e.g., Spark), using advanced analytics packages (e.g., GraphLab [18], Parameter Server [17]), or via SQL (e.g., Impala [16]). This flexibility adds performance overhead, but the ability to mix and match components and analytics packages is unprecedented at this scale. This architectural flexibility is perhaps most interesting to systems builders and vendors, who have additional degrees of freedom in designing their infrastructure offerings.

To summarize, a dominant theme in today's distributed data management infrastructure is flexibility and heterogeneity: of storage formats, of computation paradigms, and of systems implementations. Of these, storage format heterogeneity is probably the highest impact by an order of magnitude or more, simply because it impacts novices, experts, and architects alike. In contrast, heterogeneity of computation paradigms most impacts experts and architects, while heterogeneity of systems implementations most impacts architects. All three are relevant and exciting developments for database research, with lingering questions regarding market impact and longevity.

Looking Ahead

In a sense, MapReduce was a short-lived, extreme architecture that blew open a design space. The architecture was simple and highly scalable, and its success in the open source domain led many to realize that there was demand for alternative solutions and the principle of flexibility that it embodied (not to mention a market opportunity for cheaper data warehousing solutions based on open source). The resulting interest is still surprising to many and is due to many factors, including community zeitgeist, clever marketing, eco-

nomics, and technology shifts. It is interesting to consider which differences between these new systems and RDBMSs are fundamental and which are due to engineering improvements.

Today, there is still debate about the appropriate architecture for large-scale data processing. As an example, Rasmussen et al. provide a strong argument for why intermediate fault tolerance is not necessary except in very large (100+ node) clusters [23]. As another example, McSherry et al. have colorfully illustrated that many workloads can be efficiently processed using a single server (or thread!), eliminating the need for distribution at all [20]. Recently, systems such as the GraphLab project [18] suggested that domain-specific systems are necessary for performance; later work, including Grail [9] and GraphX [12], argued this need not be the case. A further wave of recent proposals have also suggested new interfaces and systems for stream processing, graph processing, asynchronous programming, and general-purpose machine learning. Are these specialized systems actually required, or can one analytics engine rule them all? Time will tell, but I perceive a push towards consolidation.

Finally, we would be remiss not to mention Spark, which is only six years old but is increasingly popular with developers and is very well supported both by VC-backed startups (e.g., Databricks) and by established firms such as Cloudera and IBM. While we have included DryadLINQ as an example of a post-MapReduce system due to its historical significance and technical depth, the Spark paper [27], written in the early days of the project, and recent extensions including SparkSQL [4], are worthwhile additional reads. Like Hadoop, Spark rallied major interest at a relatively early stage of maturity. Today, Spark still has a ways to go before its feature set rivals that of a traditional data warehouse. However, its feature set is rapidly growing and expectations of Spark as the successor to MapReduce in the Hadoop ecosystem are high; for example, Cloudera is working to replace MapReduce with Spark in the Hadoop ecosystem [13]. Time will tell whether these expectations are accurate; in the meantime, the gaps between traditional warehouses and post-MapReduce systems are quickly closing, resulting in systems that are as good at data warehousing as traditional systems, but also much more.

Commentary: Michael Stonebraker

26 October 2015

Recently, there has been considerable interest in data analytics as part of the marketing moniker "big data". Historically, this meant business intelligence (BI) analytics and was serviced by BI applications (Cognos, Business Objects, etc.) talking to a relational data warehouse (such as Teradata, Vertica, Red Shift, Greenplum, etc.). More recently it has become associated with "data science". In this context, let's start ten years ago with Map-Reduce, which was purpose-built by Google to support their web crawl data base. Then, the marketing guys took over with the basic argument: "Google is smart; Map-Reduce is Google's next big thing, so it must be good". Cloudera, Hortonworks and Facebook were in the vanguard in hyping Map-Reduce (and its open source look-alike Hadoop). A few years ago, the market was abuzz drinking the Map-Reduce koolaid. About the same time, Google stopped using Map-Reduce for the application that it was purpose-built for, moving instead to Big Table. With a delay of about 5 years, the rest of the world is seeing what Google figured out earlier; Map-Reduce is not an architecture with any broad scale applicability.

In effect Map-Reduce suffers from the following two problems:

1. It is inappropriate as a platform on which to build data warehouse products. There is no interface inside any commercial data warehouse product which looks like Map-Reduce, and for good reason. Hence, DBMSs do not want this sort of platform.
2. It is inappropriate as a platform on which to build distributed applications. Not only is the Map-Reduce interface not flexible enough for distributed applications but also a message passing system that uses the file system is way too slow to be interesting.

Of course, that has not stopped the Map-Reduce vendors. They have simply rebranded their platform to be HDFS (a file system) and have built products based on HDFS that do not include Map-Reduce. For example, Cloudera has recently introduced Impala, which is a SQL engine, not built on Map-Reduce. In point of fact, Impala does not really use HDFS either, choosing to drill through that layer to read and write the underlying local Linux files directly. HortonWorks and Facebook have similar projects underway. As such the Map-Reduce crowd has turned into a SQL crowd and Map-Reduce, as an interface, is history. Of course, HDFS has serious problems when used by a SQL engine, so it is not clear that it will have any legs, but that remains to be seen. In any case, the Map-Reduce-HDFS market will merge with the SQL-data warehouse market; and may the best systems prevail. In summary, Map-Reduce has failed as a distributed systems platform, and vendors are using HDFS as a file system under data warehouse products.

This brings us to Spark. The original argument for Spark is that it is a faster version of Map-Reduce. It is a main memory platform with a fast message passing interface. Hence, it should not suffer from the performance problems of Map-Reduce when used for distributed applications. However, according to Spark's lead author Matei Zaharia, more than 70% of the Spark accesses are through SparkSQL. In effect, Spark is being used as a SQL engine, not as a distributed applications platform! In this context Spark has an identity problem. If it is a SQL platform, then it needs some mechanism for persistence, indexing, sharing of main memory between users, meta data catalogs, etc. to be competitive in the SQL/data warehouse space. It seems likely that Spark will turn into a data warehouse platform, following Hadoop along the same path.

On the other hand, 30% of Spark accesses are not to SparkSQL and are primarily from Scala. Presumably this is a distributed computing load. In this context, Spark is a reasonable distributed computing platform. However, there are a few issues to consider. First, the average data scientist does a mixture of data management and analytics. Higher performance comes from tightly coupling the two. In Spark there is no such coupling, since Spark's data formats are not necessarily common across these two tasks. Second, Spark is main memory-only (at least for now). Scalability requirements will presumably get this fixed over time. As such, it will be interesting to see how Spark evolves off into the future.

In summary, I would like to offer the following takeaways:

- Just because Google thinks something is a good idea does not mean you should adopt it.
- Disbelieve all marketing spin, and figure out what benefit any given product actually has. This should be especially applied to performance claims.
- The community of programmers has a love affair with "the next shiny object". This is likely to create "churn" in your organization, as the "half-life" of shiny objects may be quite short.

References

[1] Apache Tez. `https://tez.apache.org/`.

[2] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, et al. The Stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964, 2014.

[3] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. Borkar, Y. Bu, M. Carey, I. Cetindil, M. Cheelangi, K. Faraaz, et al. Asterixdb: A scalable, open source bdms. In *VLDB*, 2014.

[4] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark SQL: Relational data processing in spark. In *SIGMOD*, 2015.

[5] S. Babu and H. Herodotou. Massively parallel databases and MapReduce systems. *Foundations and Trends in Databases*, 5(1):1–104, 2013.

[6] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI*, 2006.

[7] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. In *OSDI*, 2006.

[8] D. DeWitt and M. Stonebraker. Mapreduce: A major step backwards. *The Database Column*, 2008.

[9] J. Fan, A. Gerald, S. Raj, and J. M. Patel. The case against specialized graph analytics engines. In *CIDR*, 2015.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *SOSP*, 2003.

[11] A. Ghoting, R. Krishnamurthy, E. Pednault, B. Reinwald, V. Sindhwani, S. Tatikonda, Y. Tian, and S. Vaithyanathan. Systemml: Declarative machine learning on mapreduce. In *ICDE*, 2011.

[12] J. E. Gonzales, R. S. Xin, D. Crankshaw, A. Dave, M. J. Franklin, and I. Stoica. Graphx: Unifying data-parallel and graph-parallel analytics. In *OSDI*, 2014.

[13] D. Harris. Forbes: Why Cloudera is saying 'Goodbye, MapReduce' and 'Hello, Spark', 2015. `http://fortune.com/2015/09/09/cloudera-spark-mapreduce/`.

[14] M. Hausenblas and J. Nadeau. Apache Drill: Interactive ad-hoc analysis at scale. *Big Data*, 1(2):100–104, 2013.

[15] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *EuroSys*, 2007.

[16] M. Kornacker, A. Behm, V. Bittorf, T. Bobrovytsky, C. Ching, A. Choi, J. Erickson, M. Grund, D. Hecht, M. Jacobs, et al. Impala: A modern, open-source sql engine for hadoop. In *CIDR*, 2015.

[17] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

[18] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. In *VLDB*, 2012.

[19] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, 2010.

[20] F. McSherry, M. Isard, and D. G. Murray. Scalability! But at what COST? In *HotOS*, 2015.

[21] D. G. Murray, F. McSherry, R. Isaacs, M. Isard, P. Barham, and M. Abadi. Naiad: A timely dataflow system. In *SOSP*, 2013.

[22] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD*, 2008.

[23] A. Rasmussen, V. T. Lam, M. Conley, G. Porter, R. Kapoor, and A. Vahdat. Themis: An i/o-efficient mapreduce. In *SoCC*, 2012.

[24] J. Shute, R. Vingralek, B. Samwel, B. Handy, C. Whipkey, E. Rollins, M. Oancea, K. Littlefield, D. Menestrina, S. Ellner, et al. F1: A distributed sql database that scales. In *VLDB*, 2013.

[25] E. R. Sparks, A. Talwalkar, V. Smith, J. Kottalam, X. Pan, J. Gonzalez, M. J. Franklin, M. Jordan, T. Kraska, et al. Mli: An api for distributed machine learning. In *ICDM*, 2013.

[26] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. In *VLDB*, 2009.

[27] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.