CS 6332: Fall 2008
Systems for Large Data Review

Guozhang Wang

September 25, 2010

1 Data Services in the Cloud

1.1 MapReduce

MapReduce [10] gives us an appropriate model for distributed parallel computing. There are several features which are proved useful: 1) centralized job distribution. 2) Fault tolerance mechanism for both masters and workers. Although there is controversies about MapReduce capability to replace standard RDBMS [12, 13], it is reasonable that existing proposals to use MapReduce in relational data processing [39] do not manipulate very well for complicated queries.

Besides, MapReduce itself is not really user-friendly for most programmers and therefore may need some additional specific language or systems for its easy usage [30].

Lesson Learned: General solutions claimed for "all" classes of tasks are not likely to succeed, because unless it has a very general and nice model in it, it would be very complicated and hard to inefficient to use in practice. [23]

1.2 Bigtable

What is the difference between Bigtable and DBMS?

Bigtable [8] is data-oriented, while DBMS is query-oriented. Bigtable assumes simple read/write and seldom delete, and focus on scalability of data processing; DBMS assumes perfect data schema and focus on query optimization. Based on these differences, many features of DBMS is weakened or even discarded in Bigtable like transaction process and logical independence.

1.3 Google File System

What GFS [17] has told us?

1) Optimization towards certain applications – a) Most read (major sequential minor random); b) Most append write; c) Large files – can be very

efficient; On the other hand, method claimed for general usage "come and go".

2) For scalable distributed systems, it can at most achieve "relative concurrency" if efficiency is required. In other words, some design assumptions/decisions must be relaxed or sacrificed for certain features.

1.4 PNUTS

When availability and scalability comes to the stage, the designers in *Google* and *Yahoo!* chooses the similar decision: sacrifice concurrency and make simpler assumptions of the query loads. [9]

2 Parallel Database Systems

2.1 Gamma Database

Comparison between Gamma [14] and MapReduce:

1. *Application*:

   Gamma: Ad hoc queries with transactions, complex joins, and aggregates, require response time and concurrency.

   MapReduce: Batch workload with simple analysis process, require simplicity and scalability

2. *Machine*:

   Gamma: 32 machines, each one is precious.

   MapReduce: 1000+ machines, no one is ever more important.

3. *Design Decision*: Based on above differences.

   Gamma: Log/Lock on each machine; replication; schema information on Catalog Manager; query execution split (inter, intra) and optimization; query language compiler, etc (parallel ideas borrowed).

   MapReduce: "Black Box" manipulation, users take care of the schema; tolerate great nuber of workers' fault; split only on query rewrite; heavily depend on GFS for replication and consistency, etc.

Lesson Learned: Different task requirement and application environment affect the design decisions. For example, PIG is actually just standing between these two: 1) Simple query processing, but support rewrite/optimization. 2) Query tree (from Gamma), while scheduler is still simple. 3) "Relative Concurrency" requirement, replications.

3 Tradeoff between Performance and Accuracy

3.1 Isolation Level

In context of Concurrency Control, anomaly is "a bad thing", phenomenon is when "a bad thing is possible". In order that anomaly never happen, we need to somehow avoid phenomenon, which will necessarily affect the performance. Therefore different Isolation levels gives us multiple choice on how restrict we avoid phenomenon in expense of performance. [6, 5]

3.2 Recovery

To guarantee single machine recovery from fault, "Write-Ahead Logging" (WAL) is necessary; to be able to recovery from fault when previous recovery is executing, both redo and undone logs are also required. That is the essence of ARIES. [28]

For distributed data warehouses, other possibilities for recovery emerge as a result of query workload characteristics (read-only queries with smaller transactions of updates and deletions), distributed environments, data redundancy and networking efficiency. HARBOR [27] is one approach that re-examine and modify ARIES under this new class of applications. It depends on the following properties: 1) Network messaging is not too slow compared with the local disk read/write (require sites near each other), that take advantage of replication to replace stable logs and force-writes is better. 2) Queries are mostly read-only, little updates and deletions on most recent tuple, therefore system can utilize locality. One note is that on multi-insertion, ARIES' performance stays while HARBOR's running time goes up.

3.3 Paxos for Consensus

Absolute distributed consensus has been proved impossible although one process is faulty. [26, 16] However, this proof of impossible of "Completely Asynchronous" made assumptions too strong: 1) relative speed of processes is not known; 2) time-out mechanisms cannot be used; 3) dead process cannot be detected.

In practice, this situation is rarely to happen, especially the third assumption. By using timing mechanism more realistically, we can use Paxos [25] to achieve asynchronous consensus.

Due to the impossible proof, reliable consensus asynchronously is impossible because it will leads to either blocking (if use 2PC, 3PC, etc) or Non-Blocking (if use Paxos) but might never decide. By using timing mechanism we can eventually decide in Non-Blocking case.

3.4 Commit Protocol

A couple of new commit protocols can be proposed to optimize read-only transactions and some class of multisite update transactions. [29] The main idea is that under a distributed environment, we can simply get a better trade-off between costs of writes and network messaging of complicated commit protocols and costs of failure recovery process resulted from simpler commit protocols.

Chubby [7] is a distributed lock service intend for synchronization within Goggle's distributed systems. It has been proved to be useful for primary election, name service, repository of small amount of meta-data, etc. Chubby is designed for special application requirement in Google systems, like compatibility during transition, small file read/write, Google developer's programming familiarity, etc. From Chubby we can see how the system requirement and application environment can influence the key design decisions during the implementation process.

For large world-wide e-commerce platforms like Amazon, the reliability, availability, latency and throughput requirements are very stringent. Therefore, designers need to further tradeoff consistency and isolation for above system requirements. Dynamo [11], which is designed to suit Amazon's applications (simple read and write operations, and small target object files no relational schema, and relatively lower scalability requirement), provides useful insight that merging and tuning different existed techniques can meet very strict performance demands. By combining many decentralized techniques Dynamo successfully achieves high availability in distributed environment

4 Column Store

When database techniques were proposed and used in the 70's and 80's, the killer application is online transaction processing (OLTP), which is composed of mostly over-writes or updates on tuples. Therefore most major DBMS vendors implement row store architecture that aims at high writes performance. But the past ten years has witnessed a great new class of applications which has large amount of ad-hoc queries of the data but a relatively infrequent updates (or bulk loads) on the other hand. This new class of applications calls for read-optimized systems, and column store architecture is one promising approach. [36]

4.1 Features of Column Store

From column store we can observe many features that has been presented in many above section. This tells us how research ideas have been evolved

and inherited, and what we can learn from the history. We demonstrate this point by listing some of the "bullets" of column store:

1. *hybrid RS + WS*: from GFS and Bigtable.
2. *Redundant Storage*: from P2P, Distributed Database, etc.
3. *Column Storage*: from Vertical Partitioning.
4. *Compression* [2]
5. *Shared Nothing/Cluster/ "K-Safety"*: from Distributed Systems and Consensus Protocol.
6. *Snapshot Isolation*
7. *Segment Partition*: Since projection may still very large, and we have many machines, so shrink sizes further. Data Cracking to avoid too much sorting for insertions. [24, 21, 22]
8. *Query Processing*: perform directly on the column data instead of decompose "eagerly", trade CPU for I/O throughput. [19, 4]

Further experiments have shown that it might be possible to simulate a row-store in a column-store, but it will need better support for vertical partitioning at the storage layer, such as position-virtual-id, sorting, compression, etc. After the features (especially compression and late materialization) have been removed, column stores are just like row stores. [3, 20]

Lesson Learned: 1) Column stores first appear in IR, then imported into DB. Therefore people still believe in the future it will become an extension to current systems. 2) The trend of DB is that one day all the data will be in the main memory, and compression will be more and more important. Actually there are many complicated compression techniques in IR, how to import them into DB will be interesting.

4.2 Column Stores for Unstructured Data

Due to the features of column store, it can also be applied to many other unstructured data processing applications, such as XML, Semantic Web, etc. [1, 33] Experiments have shown that one of the main feature: vertical partitioning, does not scale very well although it does well for small query size. One note is that vertical partitioning is quite like shredding in XML (which is not inherited by the industry, but XQuery and XPath).

4.3 The Impact on RDBMS

The new DBMS markets with new application requirements, together with technology evolvement, result in the predicted demise of "one size fits all" systems. Further, current mainstream solutions for data model and querying language including Relational Model, SQL Language also need rethinking. [37, 18, 38]

5 Potpourri

5.1 Clustera

Instead of the "standard" job-queue-manager with different scheduling strategy that many cluster management systems use, Clustera [15] uses a Java application server with all state information about jobs, users, nodes, files, job execution history, etc stored in a relational database system. The idea of this "highly centralized" server is that the importance of recoding metadata information in the cluster server cannot be overstated. Although it might have a lower scalability compared with job-queue-manager architecture, it is a nice demonstration of academia research method.

6 Conclusion

- *9/11: How to read a paper?* When reading papers, not only focus on the contents but also focus on what authors have not say but have implied as their thread of thinking. Eg. in GFS, authors did not say their system is "relative" concurrency, but actually it is (as their assumption implied in the Introduction); we can find out that by asking "what happens when user require a page between two 'heartbeats', if there is stale page?" Ask questions to jump out of the papers' contents.

- *9/16: How to conduct experiments?* 1) Consider many different classes of "user scenario" for the proposed techniques with every one sharing some basic characters. Eg. In MapReduce evaluation [31], experiment demonstrate a lot of applications (word count, kmeans, PCA, etc) with different code size ratio. 2) Compare on different dimension (# of processor, # of data set, # of unit size), and indicate each components' run tim ratio when necessary. 3) If necessary, compare on different execution environment.

- *9/18: How to motivate the solution?* There need to have some real applications for conducted research. Eg. Social Network (already a lot of required applications) *v.s.* P2P Networks (central data storage still rules), and for bulk insertion in distributed ordered table [34], how frequently dense insertion will be applied since it seems that "append"

and "sparse insertion" are more common; Another example is using arrays in memory for database cracking [21], arrays are better for range queries but the experiment is focus on value query, then why not use lists in memory?

- *10/24: Discover ideas from history approaches.* Some old papers have hidden ideas which even authors themselves did not know about. One possible approach is to figure out in historical steps (proposals) to a certain problem, which is NECESSARY to the problem, and which is just "smart ideas" from smart people. A better understanding of historical approaches ("what comes by and goes by") gives you a deeper insight for the coming future solutions.

- *10/30: Combine/utilize existed techniques for certain problems with matched characteristics.* From Dynamo we can see many similar techniques applied in P2P systems and Distributed File systems. It does this because some of its application characteristics are similar with these systems. For example, its uses decentralized node partitioning and local routing from P2P systems [35] (small number of nodes), get/put interface and record version mechanism from PNUTS [9] (consistency can be sacrificed due to performance/availability requirement), etc.

- *11/4: Learn more in other fields.* A broad understanding of other fields (Systems, Machine Learning, Statistics, etc) can give a good guide on how well ones intuition goes and how to formalize/generalize your "ad hoc" solution into a more general framework.

- *11/11: Simple idea works best.* To avoid too much sorting before small query workload, "crack" the database.

- *11/13: Available systems shape research agenda.* The development of systems affects the research topic trends a lot. We can observe some topics that are hot 10 or 20 years ago becomes popular iterately due to development of networking/hardware/etc. For example, for Semantic Web, it is still in data collecting period, not query and analyzing period (What is the real-world workload? So far no one knows). But maybe in the future it will also have "Data Mining", "Data Integration", "Top-K Query Optimization", etc sub area on it.

- *11/18: Many techniques in original application are reminder for other applications.* For example, compression in column store stimulates other compressions in row-store, such as Rose [32]. Rose borrowed "LSM-Tree" and sequential I/O idea from IR literature, where the problem characteristics are the same: 1) High throughput writes, regardless of update patterns; 2) Write without read. It avoids random

I/O by paying for sequential scans (which is executed periodically), this is affordable since tense writes performance is more important. IR use this strategy to perform multi-way joins.

- *12/10: Do not "dream" about the industrial applications when conducting academia research.*

References

[1] Daniel J. Abadi, Adam Marcus 0002, Samuel Madden, and Katherine J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, pages 411–422, 2007.

[2] Daniel J. Abadi, Samuel Madden, and Miguel Ferreira. Integrating compression and execution in column-oriented database systems. In *SIGMOD Conference*, pages 671–682, 2006.

[3] Daniel J. Abadi, Samuel Madden, and Nabil Hachem. Column-stores vs. row-stores: how different are they really? In *SIGMOD Conference*, pages 967–980, 2008.

[4] Daniel J. Abadi, Daniel S. Myers, David J. DeWitt, and Samuel Madden. Materialization strategies in a column-oriented dbms. In *ICDE*, pages 466–475, 2007.

[5] Atul Adya, Barbara Liskov, and Patrick E. O'Neil. Generalized isolation level definitions. In *ICDE*, pages 67–78, 2000.

[6] Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, pages 1–10, 1995.

[7] Michael Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI*, pages 335–350, 2006.

[8] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *OSDI*, 2006.

[9] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. PNUTS: Yahoo!'s hosted data serving platform. In *VLDB*, 2008.

[10] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[11] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, pages 205–220, 2007.

[12] David Dewitt. MapReduce: A major step backwards. The Database Column, January 17, 2008. `http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html`.

[13] David Dewitt and Michael Stonebraker. MapReduce II. The Database Column, January 25, 2008. `http://www.databasecolumn.com/2008/01/mapreduce-continued.html`.

[14] David J. DeWitt, Shahram Ghandeharizadeh, Donovan Schneider, Allan Bricker, Hui-I Hsiao, and Rick Rasmussen. The Gamma database machine project. *IEEE Trans. Knowl. Data Eng*, 2(1), 1990.

[15] David J. DeWitt, Erik Paulson, Eric Robinson, Jeffrey F. Naughton, Joshua Royalty, Srinath Shankar, and Andrew Krioukov. Clustera: an integrated computation and data management system. *PVLDB*, 1(1):28–41, 2008.

[16] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[17] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *OSDI*, 2003.

[18] Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker. Oltp through the looking glass, and what we found there. In *SIGMOD Conference*, pages 981–992, 2008.

[19] Stavros Harizopoulos, Velen Liang, Daniel J. Abadi, and Samuel Madden. Performance tradeoffs in read-optimized databases. In *VLDB*, pages 487–498, 2006.

[20] Allison L. Holloway and David J. DeWitt. Read-optimized databases, in depth. *PVLDB*, 1(1):502–513, 2008.

[21] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Database cracking. In *CIDR*, pages 68–78, 2007.

[22] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Updating a cracked database. In *SIGMOD Conference*, pages 413–424, 2007.

[23] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *SIGOPS*, 2007.

[24] Milena Ivanova, Martin L. Kersten, and Niels Nes. Self-organizing strategies for a column-store database. In *EDBT*, pages 157–168, 2008.

[25] Leslie Lamport. Paxos made simple, fast, and byzantine. In *OPODIS*, pages 7–9, 2002.

[26] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[27] Edmond Lau and Samuel Madden. An integrated approach to recovery and high availability in an updatable, distributed data warehouse. In *VLDB*, pages 703–714, 2006.

[28] C. Mohan, Donald J. Haderle, Bruce G. Lindsay, Hamid Pirahesh, and Peter M. Schwarz. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. *ACM Trans. Database Syst.*, 17(1):94–162, 1992.

[29] C. Mohan, Bruce G. Lindsay, and Ron Obermarck. Transaction management in the R* distributed database management system. *ACM Trans. Database Syst.*, 11(4):378–396, 1986.

[30] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig Latin: A Not-So-Foreign language for data processing. In *SIGMOD*, 2008.

[31] Colby Ranger, Ramanan Raghuraman, Arun Penmetsa, Gary Bradski, and Christos Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. In *HPCA*, 2007.

[32] Russell Sears, Mark Callaghan, and Eric Brewer. Rose: compressed, log-structured replication. *PVLDB*, 1(1):526–537, 2008.

[33] Lefteris Sidirourgos, Romulo Goncalves, Martin L. Kersten, Niels Nes, and Stefan Manegold. Column-store support for rdf data management: not all swans are white. *PVLDB*, 1(2):1553–1563, 2008.

[34] Adam Silberstein, Brian F. Cooper, Utkarsh Srivastava, Erik Vee, Raghu Ramakrishnan, and Ramana Yerneni. Efficient bulk insertion into a distributed ordered table. In *SIGMOD*, 2008.

[35] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[36] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel

Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik. C-store: A column-oriented dbms. In *VLDB*, pages 553–564, 2005.

[37] Michael Stonebraker, Chuck Bear, Ugur Çetintemel, Mitch Cherniack, Tingjian Ge, Nabil Hachem, Stavros Harizopoulos, John Lifter, Jennie Rogers, and Stanley B. Zdonik. One size fits all? part 2: Benchmarking studies. In *CIDR*, pages 173–184, 2007.

[38] Michael Stonebraker, Samuel Madden, Daniel J. Abadi, Stavros Harizopoulos, Nabil Hachem, and Pat Helland. The end of an architectural era (it's time for a complete rewrite). In *VLDB*, pages 1150–1160, 2007.

[39] Hung-Chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D. Stott Parker. Map-Reduce-Merge: Simplified relational data processing on large clusters. In *SIGMOD*, 2007.