Data Management in Cloud Computing

Ms. Y.Sheeba[1] Mr. D.Jayakuar[2]
[1]Department of Computer Application [2]Department of Computer Science and Engineering
[1,2]IFET College Of Engineering

Abstract— To some, cloud computing seems to be little more than a marketing umbrella, encompassing topics such as distributed computing, grid computing, utility computing, and software-as-a-service, that have already received significant research focus and commercial implementation. Nonetheless, there exist an increasing number of large companies that are offering cloud computing infrastructure products and services that do not entirely resemble the visions of these individual component topics. In this paper, the limitations and opportunities of deploying data management issues on the emerging cloud computing platforms are discussed. It is speculated that large scale data analysis tasks, decision support systems, and application specific data marts are more likely to take advantage of cloud computing platforms than operational, transactional database systems. A list of features that a DBMS designed for large scale data analysis tasks running on cloud computing platforms should contain is presented. Some currently available open source and commercial database options that can be used to perform such analysis tasks are discussed, and it is concluded that none of these options, as presently architected, match the requisite features. The need for a new DBMS, designed specifically for cloud computing environments is also discussed in this paper.

Key words: Cloud Computing, Data Management, Software-as-a-service, Cloud DBMS

I. INTRODUCTION

Though not everyone agrees on the exact definition of cloud computing [23], most agree the vision encompasses a general shift of computer processing, storage, and software delivery away from the desktop and local servers, across the network, and into next generation data centers hosted by large infrastructure companies such as Amazon, Google, Yahoo, Microsoft, or Sun. Just as the electric grid revolutionized access to electricity one hundred years ago, freeing corporations from having to generate their own power, and enabling them to focus on their business differentiators, cloud computing is hailed as revolutionizing IT, freeing corporations from large IT capital investments, and enabling them to plug into extremely powerful computing resources over the network. Data management applications are potential candidates for deployment in the cloud. This is because an on-premises enterprise database system typically comes with a large, sometimes prohibitive up-front cost, both inhardware and in software. For many companies (especially for start-ups and medium-sized businesses), the pay-as-you-go cloud computing model, along with having someone else worrying about maintaining the hardware, is very attractive. In this way, cloud computing is reminiscent of the application service provider (ASP) and database-as-a-service (DaaS) paradigms. In practice, cloud computing platforms, like those offered by Amazon Web Services, AT&T's Synaptic Hosting, AppNexus, GoGrid, Rackspace Cloud Hosting, and to an extent, the HP/Yahoo/Intel Cloud Computing Testbed, and the IBM/Google cloud initiative, work differently than ASPsand DaaS. Instead of owning, installing, and maintaining the database software for you (often in a multi-tenancy architecture), cloud computing vendors typically maintain little more than the hardware, and give customers a set of virtual machines in which to install their own software. Resource availability is typically elastic, with a seemingly infinite amount compute power and storage available on demand, in a pay-only-for-what-you-use pricing model. This article explores the advantages and disadvantages of deploying database systems in the cloud. We look at how the typical properties of commercially available cloud computing platforms affect the choice of data management applications to deploy in the cloud. Due to the ever-increasing need for more analysis over more data in today's corporate world, along with an architectural match in currently available deployment options, we conclude that read-mostly analytical data management applications are better suited for deployment in the cloud than transactional data management applications. We thus outline a research agenda for large scale data analysis in the cloud, showing why currently available systems are not ideally-suited for cloud deployment, and arguing that there is a need for a newly designed DBMS, architected specifically for cloud computing platforms.

II. DATA MANAGEMENT IN THE CLOUD

Before discussing in Section III the features a database system must implement for it to run well in the cloud, in this section we attempt to narrow the scope of potential database applications to consider for cloud deployment. Our goal in this section is to decide which data management applications are best suited for deployment on top of cloud computing infrastructure. In order to do this, we first discuss three characteristics of a cloud computing environment that are most pertinent to the ensuing discussion.

A. Cloud characterstics

Compute power is elastic, but only if workload is parallelizable. One of the oft-cited advantages of cloud computing is its elasticity in the face of changing conditions. For example, during seasonal or unexpected spikes in demand for a product retailed by an e-commerce company, or during an exponential growth phase for a social networking Website, additional computational resources can be allocated on the fly to handle the increased demand in mere minutes (instead of the many days it can take to procure the space and capital equipment needed to expand the computational resources in-house). Similarly, in this environment, one only pays for what one needs, so increased resources can be obtained to handle spikes in load and then released once the spike has subsided. However, getting additional computational resources is not as simple as a magic upgrade to a bigger, more powerful machine on the

fly (with commensurate increases in CPUs, memory, and local storage); rather, the additional resources are typically obtained by allocating additional server instances to a task. For example, Amazon's Elastic Compute Cloud (EC2) apportions computing resources in small, large, and extra large virtual private server instances, the largest of which contains no more than four cores.

If an application is unable to take advantage of the additional server instances by offloading some of its required work to the new instances which run in parallel with the old instances, then having the additional server instances available will not be much help.

In general, applications designed to run on top of a shared-nothing architecture (where a set of independent machines accomplish a task with minimal resource overlap) are well suited for such an environment. Some cloud computing products, such as Google's App Engine, provide not only a cloud computing infrastructure, but also a complete software stack with a restricted API so that software developers are forced to write programs that can run in a shared-nothing environment and thus facilitate elastic scaling. Data is stored at an un-trusted host. Although it may not seem to make business sense for a cloud computing hosting company to violate the privacy of its customers and access data without permission, such a possibility nevertheless makes some potential customers nervous. In general, moving data off premises increases the number of potential security risks, and appropriate precautions must be made. Furthermore, although the name "cloud computing" gives the impression that the computing and storage resources are being delivered from a celestial location, the fact is, of course, that the data is physically located in a particular country and is subject to local rules and regulations. For example, in the United States, the US Patriot Act allows the government to demand access to the data stored on any computer; if the data is being hosted by a third party, the data is to be handed over without the knowledge or permission of the company or person using the hosting service . Since most cloud computing vendors give the customer little control over where data is stored (e.g., Amazon S3 only allows a customer to choose between US and EU data storage options), the customer has little choice but to assume the worst and that unless the data is encrypted using a key not located at the host, the data may be accessed by a third party without the customer's knowledge.

B. Data management applications in the cloud

The above described cloud characteristics have clear consequences on the choice of what data management applications to move into the cloud. In this section we describe the suitability of moving the two largest components of the data management market into the cloud: transactional data management and analytical data management.

1) Transactional data management

By "transactional data management", we refer to the bread-and-butter of the database industry, databases that back banking, airline reservation, online e-commerce, and supply chain management applications. These applications typically rely on the ACID guarantees that databases provide, and tend to be fairly write-intensive. We speculate that transactional data management applications are not likely to be deployed in the cloud, at least in the near future, for the following reasons: Transactional data management systems do not typically use a shared-nothing architecture. The transactional database market is dominated by Oracle, IBM DB2, Microsoft SQL Server, and Sybase [20]. Of these four products, neither Microsoft SQL Server nor Sybase can be deployed using a shared-nothing architecture. IBM released a shared-nothing implementation of DB2 in the mid-1990s which is now available as a "Database Partitioning Feature" (DPF) add-on to their flagship product , but is designed to help scale analytical applications running on data warehouses, not transactional data management . Oracle had no shared-nothing implementation until very recently (September 2008 with the release of the Oracle Database Machine that uses a shared-nothing architecture at the storage layer), but again, this implementation is designed only to be used for data warehouses.

Implementing a transactional database system using shared nothing architecture is non-trivial, since data is partitioned across sites and, in general, transactions cannot be restricted to accessing data from a single site. These results in complex distributed locking and commit protocols, and in data being shipped over the network leading to increased latency and potential network bandwidth bottlenecks. Furthermore the main benefit of a shared-nothing architecture is its scalability [15]; however this advantage is less relevant for transactional data processing for which the overwhelming majority of deployments are less than 1 TB in size [24].

It is hard to maintain ACID guarantees in the face of data replication over large geographic distances. The CAP theorem [10] shows that a shared-data system can only choose at most two out of three properties: consistency, availability, and tolerance to partitions. When data is replicated over a wide area, this essentially leaves just consistency and availability for a system to choose between. Thus, the 'C' (consistency) part of ACID is typically compromised to yield reasonable system availability.

In order to get a sense of the inherent issues in building a replicated database over a wide area network, it is interesting to note the design approaches of some recent systems. Amazon's SimpleDB [2] and Yahoo's PNUTS [6] both implement shared-nothing databases over a wide-area network, but overcome the difficulties of distributed replication by relaxing the ACID guarantees of the system. In particular, they weaken the consistency model by implementing various forms of eventual/timeline consistency so that all replicas do not have to agree on the current value of a stored value (avoiding distributed commit protocols). Similarly, the research done by Brantner et. al. found that they needed to relax consistency and isolation guarantees in the database they built on top of Amazon's S3 storage layer [3]. Google's Bigtable [5] implements a replicated shared-nothing database, but does not offer a complete relational API and weakens the 'A' (atomicity) guarantee from ACID. In particular, it is a simple read/write store; general purpose transactions are not implemented (the only atomic actions are read-modify-write sequences on data stored under a single row key). SimpleDB and Microsoft SQL Server Data Services work similarly. The H-Store project [24] aims to build wide-area shared-nothing

transactional database that adheres to strict ACID guarantees by using careful data base design to minimize the number of transactions that access data from multiple partitions; however, the project remains in the vision stage, and the feasibility of the approach on a real-world dataset and query workload has yet to be demonstrated.

There are enormous risks in storing transactional data on an un-trusted host. Transactional databases typically contain the complete set of operational data needed to power mission-critical business processes. This data includes detail at the lowest granularity, and often in clouds sensitive information such as customer data or credit card numbers. Any increase in potential security breaches or privacy violations is typically unacceptable.

We thus conclude that transactional data management applications are not well suited for cloud deployment. Despite this, there are a couple of companies that will sell you a transactional database that can run in Amazon's cloud: EnterpriseDB's Postgres Plus Advanced Server and Oracle. However, there has yet to be any published case studies of customers successfully implementing a mission critical transactional database using these cloud products and, at least in Oracle's case, the cloud version seems to be mainly intended for database backup [18].

2) Analytical data management

By "analytical data management", we refer to applications that query a data store for use in business planning, problem solving, and decision support. Historical data along with data from multiple operational databases are all typically involved in the analysis. Consequently, the scale of analytical data management systems is generally larger than transactional systems (whereas 1TB is large for transactional systems, analytical systems are increasingly crossing the petabyte barrier [25, 7]). Furthermore, analytical systems tend to be read-mostly (or read-only), with occasional batch inserts. Analytical data management consists of $3.98 billion [26] of the $14.6 billion database market [20] (27%) and is growing at a rate of 10.3% annually [26]. We speculate that analytical data management systems are well-suited to run in a cloud environment, and will be among the first data management applications to be deployed in the cloud, for the following reasons: Shared-nothing architecture is a good match for analytical data management. Teradata, Netezza, Green-plum, DATAllegro (recently acquired by Microsoft), Vertica, and Aster Data all use a shared-nothing architecture (at least in the storage layer) in their analytical DBMS products, with IBM DB2 and recently Oracle also adding shared-nothing analytical products. The ever increasing amount of data involved in data analysis workloads is the primary driver behind the choice of a shared-nothing architecture, as the architecture is widely believed to scale the best [15]. Furthermore, data analysis workloads tend to consist of many large scan scans, multidimensional aggregations, and star schema joins, all of which are fairly easy to parallelize across nodes in a shared-nothing network. Finally, the infrequent writes in the workload eliminates the need for complex distributed locking and commit protocols.

ACID guarantees are typically not needed. The infrequent writes in analytical database workloads, along with the fact that it is usually sufficient to perform the analysis on a recent snapshot of the data (rather than on up-to-the-second most recent data) makes the 'A', 'C', and 'I' (atomicity, consistency, and isolation) of ACID easy to obtain. Hence the consistency tradeoffs that need to be made as a result of the distributed replication of data in transactional databases are not problematic for analytical databases. Particularly sensitive data can often be left out of the analysis. In many cases, it is possible to identify the data that would be most damaging should it be accessed by a third party, and either leave it out of the analytical data store, include it only after applying an anonymization function, or include it only after encrypting it. Furthermore, less granular versions of the data can be analyzed instead of the lowest level, most detailed data.

We conclude that the characteristics of the data and workloads of typical analytical data management applications are well-suited for cloud deployment. The elastic compute and storage resource availability of the cloud is easily leveraged by a shared-nothing architecture, while the security risks can be somewhat alleviated. In particular, we expect the cloud to be a preferred deployment option for data warehouses for medium-sized businesses (especially those that do not currently have data warehouses due to the large up-front capital expenditures needed to get a data warehouse project off the ground), for sudden or short-term business intelligence projects that arise due to rapidly changing business conditions (e.g., a retail store analyzing purchasing patterns in the aftermath of a hurricane), and for customer-facing data marts that contain a window of data warehouse data intended to be viewed by the public (for which data security is not an issue). Data Analysis in the Cloud Now that we have settled on analytic database systems as a likely segment of the DBMS market to move into the cloud, we explore various currently available software solutions to perform the data analysis. We focus on two classes of software solutions: MapReduce-like software, and commercially available shared-nothing parallel databases. Before looking at these classes of solutions in detail, we first list some desired properties and features that these solutions should ideally have.

III. CLOUD DBMS WISH LIST

A. Efficiency:

Given that cloud computing pricing is structured in a way so that you pay for only what you use, the price increases linearly with the requisite storage, network bandwidth, and compute power. Hence, if data analysis software product A requires an order of magnitude more compute units than data analysis software product B to perform the same task, then product A will cost (approximately) an order of magnitude more than B. Efficient software has a direct effect on the bottom line.

B. Fault tolerance:

Fault tolerance in the context of analytical data workloads is measured differently than fault tolerance in the context of

transactional workloads. For transactional workloads, a fault tolerant DBMS can recover from a failure without losing any data or updates from recently committed transactions, and in the context of distributed databases, can successfully commit transactions and make progress on a workload even in the face of worker node failure. For read-only queries in analytical workloads, there are no write transactions to commit, nor updates to lose upon node failure. Hence, a fault tolerant analytical DBMS is simply one that does not have to restart a query if one of the nodes involved in query processing fails. Given the large of amount of data that needs to be accessed for deep analytical queries, combined with the relatively weak compute capacity of a typical cloud compute server instance (e.g., a default compute unit on Amazon's EC2 service is the equivalent of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor), complex queries can involve hundreds (even thousands) of server instances and can take hours to complete. Furthermore, clouds are typically built on top of cheap, commodity hardware, for which failure is not uncommon. Consequently, the probability of a failure occurring during a long-running data analysis task is relatively high; Google, for example, reports an average of 1.2 failures per analysis job [7]. If a query must restart each time a node fails, then long, complex queries are difficult to complete.

C. *Ability to run in a heterogeneous environment:*

The performance of cloud compute nodes is often not consistent, with some nodes attaining orders of magnitude worse performance than other nodes. There are a variety of reasons why this could occur, ranging from hardware failure causing degraded performance on a node [22], to an instance being unable to access the second core on a dual-core machine, to contention for non-virtualized resources. If the amount of work needed to execute a query is equally divided amongst the cloud compute nodes, then there is a danger that the time to complete the query will be approximately equal to time for the slowest compute node to complete its assigned task. A node observing degraded performance would thus have a disproportionate affect on total query latency. A system designed to run in a heterogeneous environment would take appropriate measures to prevent this from occurring.

D. *Ability to operate on encrypted data:*

As mentioned in *Analytical Data Management* sensitive data may be encrypted before being uploaded to the cloud. In order to prevent unauthorized access to the sensitive data, any application running in the cloud should not have the ability to directly decrypt the data before accessing it. However, shipping entire tables or columns out of the cloud for decryption is bandwidth intensive. Hence, the ability of the data analysis system to operate directly on encrypted data (such as in [10, 20, 18, 23, 28]) so that a smaller amount of data needs to ultimately be shipped elsewhere to be decrypted could significantly improve performance.

E. *Ability to interface with business intelligence products:*

There are a variety of customer-facing business intelligence tools that work with database software and aid in the visualization, query generation, result dash-boarding, and advanced data analysis. These tools are an important part of the analytical data management picture since business analysts are often not technically advanced and do not feel comfortable interfacing with the database software directly. These tools typically interface with the database using ODBC or JDBC, so database software that want to work these products must accept SQL queries over these connections. Using these desired properties of our cloud data analysis software, we now examine how close two currently available solutions come to attaining these properties: MapReduce-like software, and commercially available shared-nothing parallel databases.

IV. MAPREDUCE-LIKE SOFTWARE

MapReduce [7] and related software such as the open source Hadoop [12], useful extensions [21], and Microsoft's Dryad/SCOPE stack [4] are all designed to automate the parallelization of large scale dataanalysis workloads. Although DeWitt and Stonebraker took a lot of criticism for comparing MapReduce to database systems in their recent controversial blog posting [8] (many believe that such a comparison is apples-to-oranges), a comparison is warranted since MapReduce (and its derivatives) is in fact a useful tool for performing data analysis in the cloud . The MapReduce programming model and framework implementation satisfies many of the previously stated desired properties: Fault Tolerance.

MapReduce is designed with fault tolerance as a high priority. A data analysis job is divided into many small tasks and upon a failure tasks assigned to a failed machine are transparently reassigned to another machine. Care is taken to make sure that partially executed tasks are not doubly accounted for in the final query result. In a set of experiments in the original Map Reduce paper, it was shown that explicitly killing 200 out of 1746 worker processes involved in a MapReduce job resulted in only a 5% degradation in query performance [7].

A. *Ability to run in a heterogeneous environment:*

MapReduce is also carefully designed to run in a heterogeneous environment. Towards the end of a MapReduce job, tasks that are still in progress get redundantly executed on other machines, and a task is marked as completed as soon as either the primary or the backup execution has completed. This limits the effect that "straggler" machines can have on total query time, as backup executions of the tasks assigned to these machines will complete first. In a set of experiments in the original MapReduce paper, it was shown that backup task execution improves query performance by 44% by alleviating the adverse affect caused by slower machines.

B. *Ability to operate on encrypted data:*

Neither MapReduce, nor its derivatives, come with a native ability to operate on encrypted data. Such ability would have to be provided using user-defined code.

C. Ability to interface with business intelligence products:

Since MapReduce is not intended to be a database system, it is not SQL compliant and thus it does not easily interface with existing business intelligence products.

D. Efficiency:

The efficiency and raw performance of MapReduce is a matter of debate. A close inspection of the experimental results presented in the MapReduce paper [7] would seem to indicate that there is room for performance improvement. In this query, 1TB of data is read off of the 3600 disks in the cluster (in parallel) and a very simple pattern search is performed. Disk should clearly be the bottleneck resource since the string is rare, so query results do not need to be shipped over the network, and the query is computationally trivial. Despite these observations, the entire grep query takes 150 seconds to complete. If one divides the 1TB of data by the 3600 disks and 150 seconds to run the query, the average throughput with which data is being read is less than 2 MB/s/disk. At peak performance, MapReduce was reading data at 32GB/s which is less than 10MB/s/disk. Given the long start-up time to get to peak performance, and the fact that peak performance is four to six times slower than how fast disks in the cluster could actually be read, there indeed is room for improvement. Other benchmarks [27] (albeit not performed up to the standards of publishable academic rigor) have also shown MapReduce to be about an order of magnitude slower than alternative systems. Much of the performance issues of MapReduce and its derivative systems can be attributed to the fact that they were not initially designed to be used as complete, end-to-end data analysis systems over structured data. Their target use cases include scanning through a large set of documents produced from a web crawler and producing a web index over them [7]. In these applications, the input data is often unstructured and a brute force scan strategy over all of the data is usually optimal. MapReduce then helps automate the parallelization of the data scanning and application of user defined functions as the data is being scanned. For more traditional data analysis workloads of the type discussed in *Analytical Data Management* that work with data produced from business operational data stores, the data is far more structured. Furthermore, the queries tend to access only a subset of this data (e.g., breakdown the profits of stores located in the Northeast). Using data structures that help accelerate access to needed entities (such as indexes) and dimensions (such as column-stores), and data structures that pre-calculate common requests (such as materialized views) often outperform abrute-force scan execution strategy.

Many argue that the lack of these "helper" data structures in MapReduce is a feature, not a limitation. These additional structures generally require the data to be loaded into to data analysis system before it can be used. This means that someone needs to spend time thinking about what schema to use for the data, define the schema and load the data into it, and decide what helper data structures to create (of course self-managing/self-tuning systems can somewhat alleviate this burden). In contrast, MapReduce can immediately read data off of the file system and answer queries on-the-fly without any kind of loading stage. Nonetheless, at the complexity cost of adding a loading stage, indexes, columns, and materialized views unquestionably can improve performance of many types of queries. If these data structures are utilized to improve the performance of multiple queries, then the one-time cost of their creation is easily outweighed by the benefit each time they are used. The absence of a loading phase into MapReduce has additional performance implications beyond precluding the use of helper data structures. During data load, data can be compressed on disk. This can improve performance, even for brute-force scans, by reducing the I/O time for subsequent data accesses. Furthermore, since data is not loaded in advance, MapReduce needs to perform data parsing at runtime (using user-defined code) each time the data is accessed, instead of parsing the data just once at load time.

The bottom line is that the performance of MapReduce is dependent on the applications that it is used for. For complex analysis of unstructured data (which MapReduce was initially designed for) where brute-force scans is the right execution strategy, MapReduce is likely a good fit. But for the multi-billion dollar business-oriented data analysis market, MapReduce can be wildly inefficient.

V. SHARED-NOTHING PARALLEL DATABASES

A more obvious fit for data analysis in the cloud are the commercially available shared-nothing parallel databases, such as Teradata, Netezza, IBM DB2, reenplum, DATAllegro, Vertica, and Aster Data, that already hold a reasonable market share for on-premises large scale data analysis [26]. DB2, Greenplum, Vertica, and Aster Data are perhaps the most natural fit since they sell software-only products that could theoretically run in the data centers hosted by cloud computing providers. Vertica already markets a version of its product designed to run in Amazon's cloud [25]. Parallel databases implement a largely complimentary set of properties from our wish list relative to MapReduce-like software:

A. Ability to interface with business intelligence products

Given that the business intelligence products are designed to work on top of databases, this property essentially comes for free. More mature databases, such as DB2, tend to have carefully optimized and certified interfaces with a multitude of BI products.

B. Efficiency

At the cost of the additional complexity in the loading phase discussed in *MapReduce-like software*, parallel databases implement indexes, materialized views, and compression to improve query performance.

C. Fault tolerance

Most parallel database systems restart a query upon a failure. This is because they are generally designed for environments where queries take no more than a few hours and run on no more than a few hundred machines. Failures are relatively rare in such an environment, so an occasional query restart is not problematic. In contrast, in a cloud computing environment, where machines tend to be cheaper, less reliable, less powerful, and more numerous, failures are more common. Not all parallel databases, however, restart a query upon a failure; Aster Data reportedly has a demo

showing a query continuing to make progress as worker nodes involved in the query are killed [17].

D. Ability to run in a heterogeneous environment

Parallel databases are generally designed to run on homogeneous equipment and are susceptible to significantly degraded performance if a small subset of nodes in the parallel cluster is performing particularly poorly.

E. Ability to operate on encrypted data

Commercially available parallel databases have not caught up to (and do not implement) the recent research results on operating directly on encrypted data. In some cases simple operations (such as moving or copying encrypted data) are supported, but advanced operations, such as performing aggregations on encrypted data, is not directly supported. It should be noted, however, that it is possible to hand-code encryption support using user defined functions.

VI. A CALL FOR A HYBRID SOLUTION

It is now clear that neither MapReduce-like software, nor parallel databases are ideal solutions for data analysis in the cloud. While neither option satisfactorily meets all five of our desired properties, each property (except the primitive ability to operate on encrypted data) is met by at least one of the two options. Hence, a hybrid solution that combines the fault tolerance, heterogeneous cluster, and ease of use out-of-the-box capabilities of MapReduce with the efficiency, performance, and tool plugability of shared-nothing parallel database systems could have a significant impact on the cloud database market.

There has been some recent work on bringing together ideas from MapReduce and database systems; however, this work focuses mainly on language and interface issues. The Pig project at Yahoo [21] and the SCOPE project at Microsoft [4] aim to integrate declarative query constructs from the database community into MapReduce-like software to allow greater data independence, code reusability, and automatic query optimization. Greenplum and Aster Data have added the ability to write MapReduce functions (instead of, or in addition to, SQL) over data stored in their parallel database products [13]. Although these four projects are without question an important step in the direction of a hybrid solution, there remains a need for a hybrid solution at the systems level in addition to at the language level. One interesting research question that would stem from such a hybrid integration project would be how to combine the ease-of-use out-of-the-box advantages of MapReduce-like software with the efficiency and shared-work advantages that come with loading data and creating performance enhancing data structures. Incremental algorithms are called for, where data can initially be read directly off of the file system out-of-the-box, but each time data is accessed, progress is made towards the many activities surrounding a DBMS load (compression, index and materialized view creation, etc.).

Another interesting research question is how to balance the tradeoffs between fault tolerance and performance. Maximizing fault tolerance typically means carefully checkpointing intermediate results, but this usually comes at a performance cost (e.g., the rate which data can be read off disk in the sort benchmark from the original MapReduce paper is half of full capacity since the same disks are being used to write out intermediate Map output). A system that can adjust its levels of fault tolerance on the fly given an observed failure rate could be one way to handle the tradeoff. The bottom line is that there is both interesting research and engineering work to be done in creating a hybrid MapReduce/parallel database system.

VII. CONCLUSION

Data management issues of now-a-days' in cloud computing platform are well studied and reported. The opportunities and limitations in data management at cloud computing platform are studied and documented.

REFERENCES

[1] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. InProc. of SIGMOD, pages 563–574, 2004.
[2] Amazon Web Services. SimpleDB. Web Page. http://aws.amazon.com/simpledb/.
[3] M. Brantner, D. Florescu, D. Graf, D. Kossmann, and T. Kraska. Building a Database on S3. In Proc. of SIGMOD, pages 251–264, 2008.
[4] R. Chaiken, B. Jenkins, P.-A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. Scope: Easy and efficient parallel processing of massive data sets. InProc. of VLDB , 2008.
[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach , M. Burrows, Chandra, A. Fikes, and R. E. Gruber. Bigtable: a distributed storage system for structured data. In Proceedings of OSDI , 2006.
[6] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!s hosted data serving platform. In Proceedings of VLDB , 2008.
[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. pages 137–150, December 2004.
[8] D. DeWitt and M. Stonebraker. MapReduce: A major step backwards. DatabaseColumn Blog. http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html
[9] T. Ge and S. Zdonik. Answering aggregation queries in a secure system model. In Proc. of VLDB , pages 519–530, 2007.
[10] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51–59, 2002.

