Jozsef Patvarczki
Comprehensive exam
Due August 24th, 2010

Subject: Distributed Database Systems

Q1) **Map Reduce and Distributed Databases**

Map Reduce (Hadoop) is a popular framework for conducting data processing in a parallel manner. It requires us to take the data "out of" the database in order to process it efficiently, yet promises to achieve high-performance and scalable data processing by exploiting the power of a compute cluster (cloud) with ease. More recently, a marriage of MapReduce with DBMS Technologies has been touted as the new approach towards achieving both performance on the one hand as well as scalability and flexibility on the other hand. One example of such a hybrid architecture is HadoopDB by Abouzeid et al. [1].

a) First review the key issues that must be tackled with designing such a hybrid combining map-reduce style processing with database technology.
b) Then, compare this sort of novel hybrid architecture against more traditional distributed database systems' architectures (including the one that you are developing for your dissertation).
c) Will these novel systems replace the traditional distributed database products, or do they have different scopes of applicability? Explain your answer.
d) Which architecture is likely to scale to larger node clusters, and why? Which is likely to perform more seamlessly under failure of nodes within the cluster?
e) Which is expected to hold up better under update-heavy applications versus analytics (mostly read-only) applications?
f) Conduct a literature study to compare and contrast these technologies, starting with the HadoopDB paper [1].

[1] HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads, A Abouzeid, K BajdaPawlikowski, D. Abadi, A. Silberschatz, A. Rasin, VLDB'2009.

a) MapReduce [Dean & S. Ghemawat, 2004] is a programming model with an associated implementation to process and generate huge amount of data in a large scale (hundreds and thousands nodes) heterogeneous shared-nothing environment. Shared-nothing environment deploys the servers with their own local disk and memory connected with high speed network. MapReduce is a simple model consisting only two functions: map and reduce. Users have to write these functions to produce key-value pairs based on the available input data. It uses a distributed file [Borthakur, 2009] system where the input data is partitioned and stored on multiple nodes of the cluster. The map function works like a filter or transformer operator that is applied on the input data set. The output of the map operator is a set of intermediate key-value pairs stored on the local disk of the node. These intermediate key-value pairs are partitioned into R disjoint buckets based on a hash function on the key of each output record. In the second phase R instances of the Reduce function are executed. The input files for Rs are transferred through the network from the local disk of the previous nodes where the Map function saved them. Reduce function processes and combines the input records and writes them back to the distributed file system in an output file. Parallel databases use shared-nothing infrastructures in a cluster environment and it can execute queries in parallel using multiple nodes. One big difference is that parallel databases support SQL (Structured Query Language) and standard relational database tables. MapReduce has no pre-defined schema and it allows the data to be in any format. Because the data can be in any format the system does not provide e.g. built in indexes. However, HadoopDB [Abouzeid et al., 2009] (figure 1) combines the two approaches into one and targets the performance and scalability of a parallel database and the fault-tolerance feature of the flexible MapReduce to achieve better structured data processing. It uses Hadoop [Hadoop09] the open source implementation of MapReduce to parallelize the queries across nodes. Scheduling and job tracking is managed by Hadoop task coordinator (JobTracker and TaskTracker). HadoopDB provides a front-end for users to process SQL queries. Queries are created using SQL-like query language (HiveQL [Thusoo, 2009]) and translated into MapReduce jobs by the help of the extended version of Hive warehousing solution [Thusoo, 2009], called SMS planner. HadoopDB uses PostgreSQL [PostgreSQL09] as a database layer that processes the translated SQL queries. To design a hybrid infrastructure like HadoopDB multiple key issues should be considered at different levels.

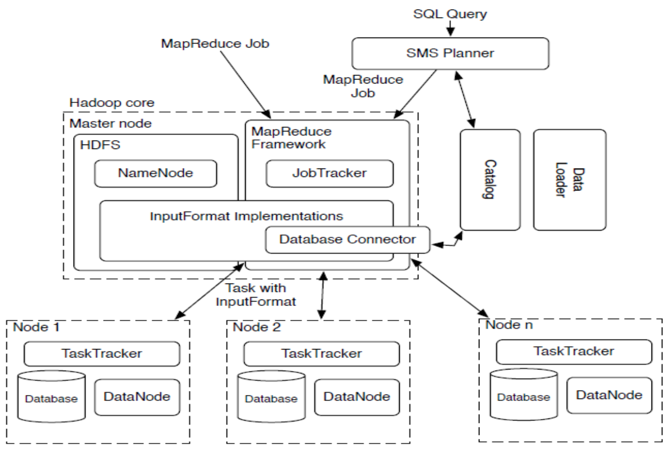

Figure 1: Architecture of the HadoopDB [Abouzeid et al., 2009]

To process extremely large dataset in a large-scale (thousand of nodes) shared-nothing environment where analytical workload performs heavy table scans scalability, performance, and availability are important factors. The amount of the analyzable data is growing and requires more and more computational nodes to complete the data analysis within a reasonable amount of time. Parallel databases can scale-up well if the number of the involved nodes is small. They assume a homogenous set of machines, but in a large scale this assumption fails. In a heterogeneous environment the probability of a possible server failure is high. Google reported 1.2 failures/analysis job [Dean & Ghemawat, 2004] for MapReduce. One of the first key issues is the data distribution. A highly scalable distributed file system is necessary to handle large amount of data and to eliminate a possible performance bottleneck. This file system should be fast and fault-tolerant for a possible node failure. HDFS (Hadoop Distributed File System) stores the data in fixed size blocks and distributed across multiple nodes. A central service (NameNode) maintains information about the location and size of each chunk. The parallel query optimizer of a parallel database always sends the query to the node where the data is located. MapReduce always moves the data where the computation is performed. This hybrid infrastructure takes this one step further. It loads the data from HDFS to the PostgreSQL nodes by the help of a dataloader that enforces two hashing phases. It utilizes the databases by dividing the data into as many chunks as the number of nodes. In the second phase it divides them further into chunks and loads each chunk into a separate database using a node. This structure is distributed thereafter. It can happen that some tables are collocated and partitioned on the query's attribute. In this case the requested operator can be handled by the database layer (PostgreSQL) directly. The implemented service (dataloader) should be fast and accurate to perform the two phase hashing, data re-partitioning, and loading the chunks. The SQL query interpreter and translator can be a performance bottleneck as well. This part of the system is responsible to optimize the query plans and translate SQL queries into MapReduce relational operators. The translator should be able to translate MapReduce relational operators back to SQL queries and utilize the parallel database at the database layer. Moreover, the SMS planner of the system can be a huge single-point-of-failure. The SMS planner extends Hive and it is responsible to update MapReduce's central information bank with the location of the database tables. It also scans the generated MapReduce jobs to determine the partitioning key in the DAG. This is a key component of the HadoopDB. If this component is not designed perfectly the system cannot interact with MapReduce. Fault tolerance is another key issue. If MapReduce is combined with parallel databases then it provides a more robust and sophisticated failure mechanism. If a job fails during execution the MapReduce framework can continue the failed job on the same node since output files of the Map function are kept on the local disk. If the node has a hardware failure then the system can re-schedule the task on a different node automatically. Parallel databases will not save intermediate results to disk and cannot continue the job execution. In the case of a possible node failure the database is capable to commit transactions successfully (e.g. log based commit). As we have already mentioned above, parallel databases are not designed for usage in a large-scale heterogeneous environment. Concurrent queries, node disk fragmentations, or corrupted data parts can decrease the performance of the system. MapReduce can handle these problems as well. It can schedule parallel execution of the same task on different node if it detects that the data processing is slow. Furthermore it can catch the process before it terminates due to a bad data segment and re-schedule the task using data from a different location. These are all important designing issues. Further design issues are load sharing within the system and locality tracking. Load sharing should balance the load equally utilizing all the nodes. Locality keeps the slave processes close to the master process to reduce the communication overhead. MapReduce uses master-slave topology to process the job. The master monitors the slave processes. Finally, this complex system should be flexible and allow users to write their own user defined functions which can be executer parallel utilizing the databases.

b) There is a conceptual difference between parallel database management systems (DBMS), MapReduce based systems (Hadoop), and hybrid systems like HadoopDB. In the case of DBMS user can state what he/she wants in SQL language. In the case of MapReduce like systems the user can present an algorithm to specify what he/she wants in a low level programming language [Pavlo et al., 2009]. HadoopDB hides the latest one from the users and provides the DBMS flexibility to the data analyst. In general there are a couple of differences between this MapReduce based hybrid system and other parallel databases. DBMSs have pre-defined table schemas with rows and columns. MapReduce does not have any pre-defined schemas so the user has to create them. Once such a schema is defined then the next task is to make sure no specific constraints are violated by the programmers. DBMS provides this by default. DBMSs have the capability to create indexes (B-Tree/Hash based) on specific columns to speed up scan functions. MapReduce does not have any built-in indexes. MapReduce introduces additional network traffic and disk accesses with the Reduce function that transfers and groups data parts together. Users can implement different functions in the Map and the Reduce parts, but DBMSs support user-defined functions, which can be executed in parallel. MapReduce HadoopDB combines all features of the DBMS with MapReduce to create a hybrid system that is good for analytical purposes. Vertica Analytic Database [Vertica, 2008] utilizes cheap shared-nothing commodity hardware and it is designed for large scale data warehouses. It uses a column store architecture where each column is independently stored on different nodes. It applies vertical partitioning on the original dataset to create multiple partitions that can be replicated across cluster nodes (figure 2). It is mostly for read intensive analytical applications where the system has to access a subset of columns. Vertica's optimizer is designed to operate on this column partitioned architecture to reduce I/O cost dramatically. It employs various data compression techniques to minimize the space requirements of the columns. The optimizer stores views of the table data in projections. The projection can contain a subset of the columns of a table or multiple tables to support materializing joins. Projections are created automatically by Vertica to support ad-hoc queries. To avoid node failure Vertica creates k+1 copies of the projections (k is the total number of nodes) and fully replicate them. In the case of a failure, it automatically switches to the next available instance. It has a built in automatic physical database design tool that creates these projections automatically and targets star (fact and dimension tables where fact tables are range partitioned across the nodes and dimension tables are replicated) or snowflake (normalization of dimension tables) schemas for automatic design.

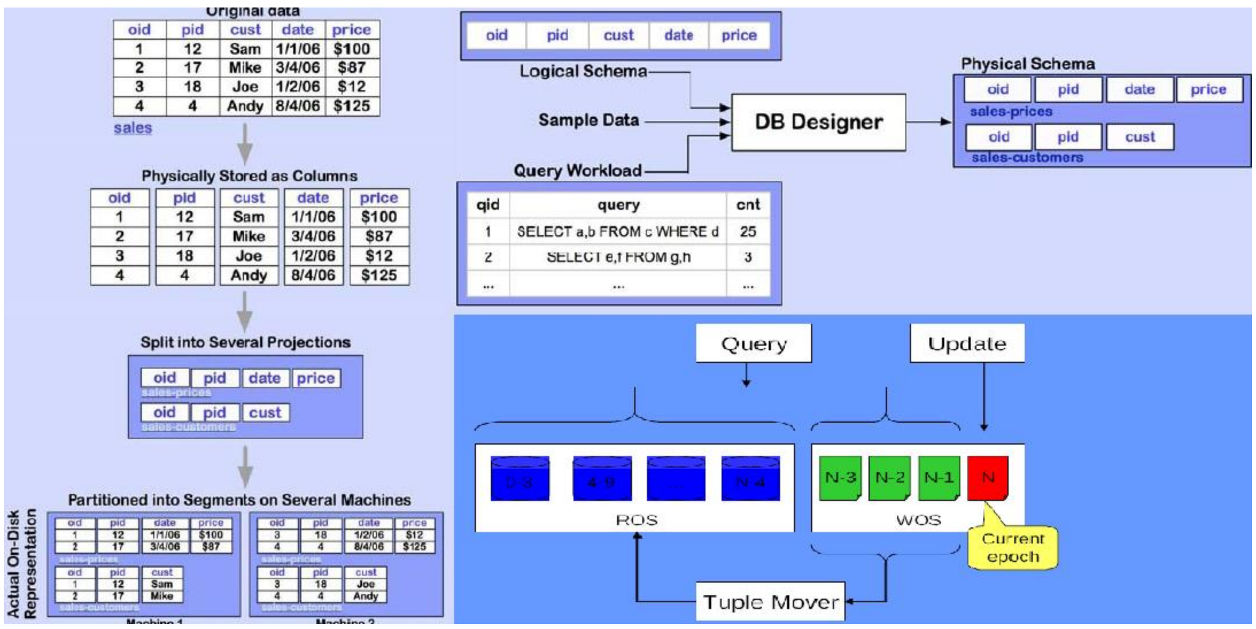

Figure 2: Architecture of Vertica [Vertica, 2008]

The hybrid storage module caches all updates to a memory segment called WSO (Write-optimized Store). A tuple mover migrates recent updates to permanent storages periodically and the system uses snapshot isolation to keep the consistency with the current updates. HadoopDB is a more robust system that provides fully structured relational tables with SQL language support where the structured data can be optimized. Vertica has a built in automatic database design tool that can adjust the system performance creating a new projection or partitioning the data according to the preferable schema. Vertica has nice data compression techniques that outperform HadoopDB in an I/O intensive task. HadoopDB SMS planner pushes the different

SQL clauses into the database layer and it can benefit from the created table indexes. A problem can be that the data is not partitioned according to the requested key. In this case this hybrid infrastructure re-partitions the data tuhat can decrease the system performance significantly. In the case of a join task – that execution time can be crucial – the two systems have a big difference. Vertica with the data compression, indexing, and projection has a native built in join query support. HadoopDB SMS optimizer (based on Hive) does not have full support for joins and cost based optimization of the queries. HadoopDB can benefit from the join if both input datasets are sorted on the join key. Then it can push down the join to the database layer. Otherwise, re-partitioning of the input data is required that adds a significant overhead. Netezza [Davidson et al., 2006] parallel system is a two-tiered system that is capable to handle very large queries from multiple users. The first-tier is a high-performance multiprocessing unit that compiles queries and generates the query execution plans. It divides the queries into sub-tasks for parallel processing and utilizes the second tier's Snippet Processing Units (SPU) for execution. Netezza combines the two tiers into one and hides the complexity of the system while providing SQL interface to the users. In the case of HadoopDB indexing can speed-up the queries execution time. Netezza has no indexing feature because the query processing is done at the disk level. The proper distribution of the tables over SPUs is the key issue to achieve high performance. The system distributes the tables based on the fields that would be indexed by HadoopDB and each SPU can process its own set of data without intercommunication with other SPUs. Teradata [Clarke, 2000] has two main requirements: ensure that the data is available when it is requested and being able to access the information without significant delay. It uses a shared-nothing architecture where the data is assigned to each unit. Virtual Access Module Processes (VAMPs) is responsible for controlling the database processing. VAMP executes index scans, reads, join, etc. functions using its own independent file system. A difference is that Teradata supports single row manipulation, block manipulation and full table or sub-table manipulation as well. It distributes the data randomly utilizing all the nodes. It provides a single hash based partitioning algorithm that partitions the data equally across all VAMPs. The hash re-distribution is an automatic task in the background according to the required update, delete, or insert actions. Another big difference is that it has a built in dynamic statistics collector that dynamically increases the number of VAMPs upon high load and distributes the requests equally. It has a built in optimizer that can handle sophisticated queries, ad-hoc queries, and complex joins as well. Teradata supports direct data loading into the database and the system handles the partitioning, indexing, etc. automatically. IBM DB2 Data partitioning [Rao et al., 2002] is based on shared-nothing architecture as well. Their tool suggests possible partitions based on the given workload and the frequency of each SQL statement occurrence. Their system recommends the best operator (partitioning or full replication) for each table. The system computes a set of interesting candidates that can help to reduce the query cost (overall cost is a combination of different constraints: I/O, CPU, and communication cost). For IBM DB2 data partitioning selection strategy [Zilio et al., 1994] the main goal is - for a given static database schema and workload characteristic - to minimize the overall response time of the workload in multiple nodes. They introduce two main placement algorithms: the Independent relations and the Comb one. The Independent Relation considers each query attributes separately. For example, if there is a query 'select book from table1, table2 where table1.book_id = table2.book_id', then the algorithm considers table1.book_id as a partition key first and then table2.book_id. The Comb algorithm considers the combination of the keys, e.g. table1.book_id and table2.book_id together. After the partition key decision, the algorithm groups the relations together and determines the node to which to assign the partition key to based on their relation grouping technique. There are several other parallel databases available like Exadata (parallel database version of Oracle), MonetDB, ParAccel, InfoBright, Greenplum, NeoView, Dataupia, DATAllegro, Exasol, etc. that all combine different techniques to achieve better performance and reliability.

Our parallel database architecture [Patvarczki, 2010a, 2010b] is also based on shared-nothing community hardware where each node has its own CPU, disk, RAM, and file system. We specialized on Web-based application where the workload consists of a fixed number of query templates. This means the system does not face with ad-hoc and unexpected queries. Because we know all the query templates beforehand our system can pre-partition the data using different operators and pre-determined heuristics [Patvarczki et al. 2009]. Each node has a PostgreSQL database but the main difference is that we do not need to re-partition the data like HadoopDB since we do not have unexpected queries. We characterize the problem as an AI search over database layout. We iteratively minimize the total cost of the workload creating different database layout and increase the system throughput (model and partition first then load the data approach vs. load the data and re-partition). Moreover, our system has a built-in corpus with generalized machine learned rules to determine which operator (Horizontal partitioning, Vertical Partitioning, Replication, and De-normalization) is applicable and when. As soon as the layout is determined, the data is distributed across the server nodes. A central dispatcher – similar to HadoopDN catalog - maintains the statistics about the current layout (table descriptors, data part locations, etc.). Since we do not have an unexpected query, the data can be partitioned according to a pre-defined rule: each query should be answerable using a single node. This means that all the joins are pre-computed and the communication bottleneck (e.g. in the case of MapReduce the Reduce function moves the files and loads the data from multiple location) is eliminated. The central dispatcher can push each query into the database layer directly where the well defined schemas support indexing. We support INSERT INTO, UPDATE, and DELETE SQL statements natively (Hadoop with Hive does not – see question1 part e). We do not have additional failure detection mechanism, but the system is easily expandable with a full copy of the original database. Furthermore, our system needs an additional layer – possible integration with Hadoop - if it wants to scale-up to thousands of nodes.

c) They definitely have different scopes of applicability based on the requirements of a possible application. These systems are designed to load terabytes of structured data into their analytical database tier per day. They are mainly for data warehousing corporations. MapReduce is capable to scale-up to thousands of nodes, but a modern DBMS can handle the same size of data in the range of 1-2PB [Pavlo et al. 2009] and ~100 nodes are enough for the task. Facebook that connects 500 million users utilizes a parallel MySql cluster is the main database layer and it is enough to store the users' data [Kerner, 2010]. They do not use the databases for joins and complex queries. In addition Facebook has two Hadoop clusters (primary and secondary –where the secondary is the replica of the primary) with ~2200 servers and 36PB data per cluster. The Hadoop cluster is for analyzing users' habits and to compute site usage statistics using complex queries. They combined Hadoop with Hive to add SQL interface like HadoopDB did [Thusoo et al., 2010]. This clearly shows that the used technology depends on the application requirements. If the environment has analytical requirements combined with complex queries and 30-100PB (Facebook expects to store 50PB data by the end of this year) then hybrid infrastructure is a novel approach.

d) MapReduce like architectures can scale to larger node clusters more effectively. This architecture was created and designed to be effective with thousands of nodes in a shared-nothing heterogeneous environment. Originally, parallel databases were created to be effective with operating a small set of nodes in homogenous environment and with the assumption that a possible failure is an unlikely event [Abouzeid, 2009]. To achieve homogeneity at the scale of thousands machines is impossible. Moreover, as the number of the nodes is increasing the probability of a possible node failure is growing with it. The HadoopDB hybrid infrastructure has a two phase hashing provided by the dataloader. It utilizes the databases by dividing the data into as many chunks as the number of nodes. In the second phase it divides them further into chunks and loads each chunk into a separate database using a node. It is also responsible for re-partitioning the data on a specific key. In the case of a join query multiple input data need to be re-partitioned if they were not partitioned on the joining key before. This could significantly decrease the performance on a large-scale level.
MapReduce like architectures can perform more seamlessly under the failure of nodes within the cluster. It was designed for large scale heterogeneous cheap commodity hardware environment where a possible hardware failure is expected. Hadoop can re-start a failed map or reduce task on a different node if the task's node goes down. Parallel database has to restart the entire query processing. Vertica keeps an extra copy of every table segment and this backup copy is assigned to different nodes. In the case of a failure the replica is used. If a node becomes slow or fragmented Hadoop replicates the job on a different node and copies the data files with it. Most of the parallel databases have no built in straggler node detections and this can dramatically decrease the performance in large scale environment. HadoopDB does not copy over the data from the slow node. The new node connects to the struggler database and keeps adding extra load to the struggling server (figure 3).

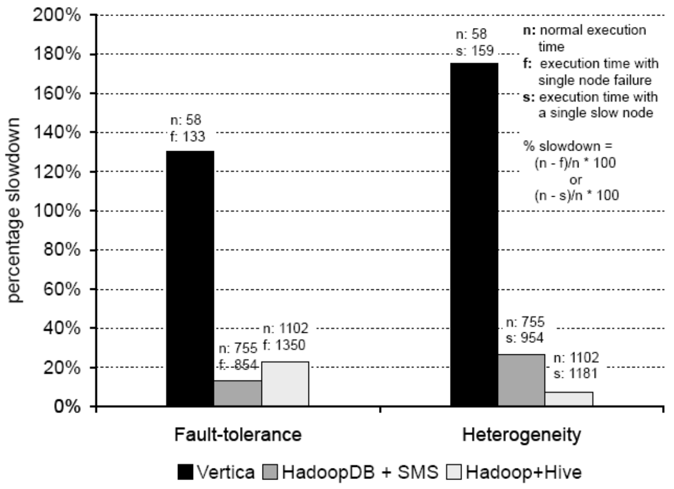

Figure 3: Fault tolerance and heterogeneity test [Abouzeid et al., 2009]

e) Hadoop connected with Hive currently does not support INSERT INTO, UPDATE, and DELETE statements [Thusoo et al., 2010]. This allows the system to handle read and write concurrencies without specific locking mechanism. An INSERT INTO statement overwrites the previous table or data partitions. However, Hadoop combined with HBase [Khetrapal & Ganesh, 2008] (the clone of Google BigTable) column oriented semi-structured data store provides additional features. It runs on the top of Hadoop HDFS file system and provides CRUD: Create, Read, Update, and Delete features. Parallel databases are expected to hold up better under update-heavy applications with their built in update propagation methods (e.g. snapshot isolation). Figure 3 shows the result of a simple selection task (SELECT pageURL, pageRank FROM Rankings WHERE pageRank > 10) using Vertica, a small row-based parallel database DBMS-X, HadoopDB, and Hadoop.

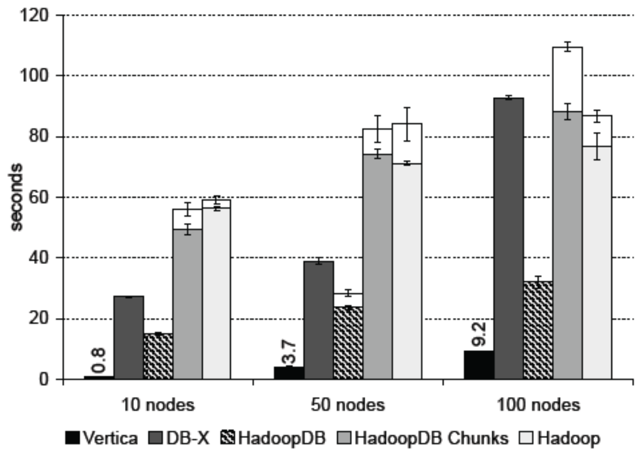

Figure 3: Aggregation task results [Abouzeid et al., 2009]

Hadoop uses only Map function that outputs the pageURL and pageRank key-value pairs. HadoopDB pushes the selection and the projection into the database layer (PostgreSQL). Parallel databases and the hybrid infrastructure benefit from the indexing possibilities of the database layer and outperform Hadoop. Vertica uses its own data compression techniques. As a conclusion, parallel databases are expected to hold up better under analytics applications.

f) As Google released and published its Distributed File System GFS (2003), an open-source version of GFS was implemented called HadoopDFS (HDFS). Google released MapReduce (Simplified Data Processing on Large Clusters 2004) and a similar open-source system (Hadoop) was published. Google released Bigtable (A Distributed Storage System for Structured Data 2006) and an open source version of Bigtable was created called HBase [Khetrapal & Ganesh, 2008]. Since the open-source versions of each software component are available free of charge, most of the database companies started to combine them with their own parallel database solutions to achieve a highly scalable system that is capable to process large amounts of data. HadoopDB is the combination of these open-source software components (HDFS and Hadoop) with its own extra modified version of Hive called SMS Planner and an additional database layer (PostgreSQL). As the amount of the available data is growing the need for complex analytic queries is increasing. HadoopDB targets the performance and scalability of a parallel database and the fault-tolerance feature of the flexible MapReduce to achieve better data processing. If we compare Hadoop with HadoopDB, we can realize that Hadoop was not designed for large number of structured data analysis and a parallel database e.g. Netezza parallel database [Davidson et al., 2006] (figure 4) outperforms Hadoop on the same job.

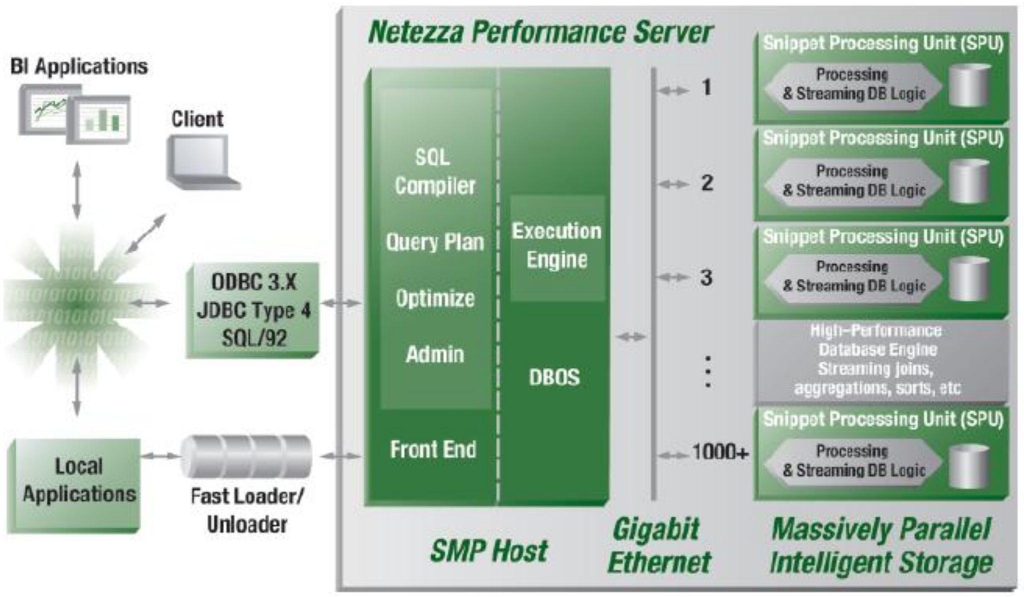
Netezza Performance Server
BI Applications
Client
ODBC 3.X
JDBC Type 4
SQL/92
Local Applications
Fast Loader/ Unloader
SQL Compiler
Query Plan
Optimize
Admin
Front End
Execution Engine
DBOS
SMP Host
1
2
3
1000+
Gigabit Ethernet
Snippet Processing Unit (SPU)
Processing & Streaming DB Logic
High-Performance Database Engine Streaming joins, aggregations, sorts, etc
Massively Parallel Intelligent Storage

Figure 4: Netezza architecture [Davidson et al., 2006]

Hadoop has no built in features similar to a parallel database that supports compressed data handling, defined table schemas, and materialized view features in default. HadoopDB utilizes the parallel databases' features and combines these features with the highly scalable and fault tolerant Hadoop. Netezza high-performance multiprocessing unit (see question 1 part b for details) compiles queries and generates the query execution plan. It divides the queries into sub-task for parallel processing. HadoopDB indexing can speed-up the queries execution time dramatically. Netezza has no indexing feature because the query processing is done at the disk level. It distributes the tables over computing units to achieve high performance. Netezza distributes the tables based on the fields that would be indexed by HadoopDB and each computing unit can process its own set of data without intercommunication with other units. Netezza does query planning in advance and assigns a part of the task to a computing unit based on the performance of the node (it is kind of common in the world of parallel databases). Teradata [Clarke, 2000] uses a shared-nothing architecture where the data is assigned to each unit.

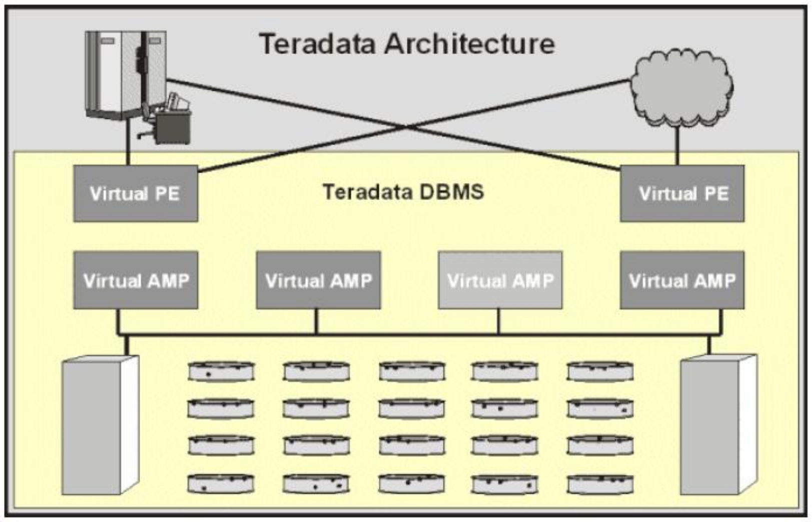

Figure 5: Teradata architecture [Clarke, 2000]

Virtual Access Module Processes (VAMPs) is responsible for controlling the database processing. Teradata supports single row manipulation, block manipulation and full table or sub-table manipulation as well. It distributes the data randomly utilizing all the nodes and provides a single hash based partitioning algorithm that partitions the data equally across all VAMPs. The hash re-distribution is an automatic task in the background according to the required update, delete, or insert actions. A big difference is that the built in dynamic statistics collector - that dynamically increases the number of VAMPs upon high load - distributes the requests equally. It has a built in optimizer and can handle sophisticated queries, ad-hoc queries, and complex joins effectively. Current implementation of HadoopDB's SMS planner cannot handle simple joins effectively because of the possible data re-partitioning and it prefers to utilize MapReduce task. Teradata supports direct data loading into the database and it handles the partitioning, indexing, etc. automatically. Dataupia [Dataupia 2008a, 2008b] Satori Server is a rack mountable unit solution. The smallest unit can store up to 2TB and it can utilize Oracle, SQL Server, or IBM DB2 at the database layer. It creates a single pool of virtual data (figure 6) from the multiple databases and handles transactions with full ACID properties. Dataupia targets mixed workloads including complex joins and complex aggregates and processes them with 10 seconds refresh rates.

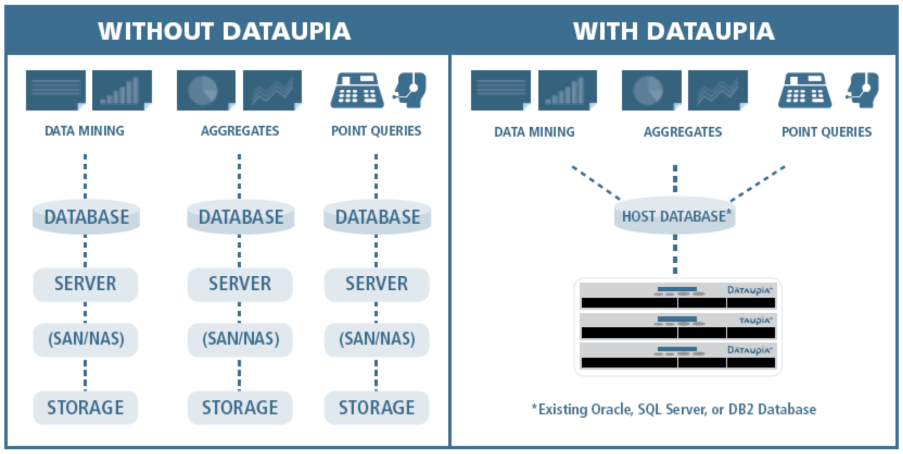

Figure 6: Dataupia architecture [Dataupia, 2008a]

Sybase IQ [Howard, 2005] is a column based – like Vertica (see question 1 part b) - analytics parallel database. It is different than the previous parallel databases because it stores the data in columns instead of rows and provides real-time analysis as well. It is similar to others because it supports SQL language, database schemas, and indexing, but it has its own specialized indexes for e.g. time series analysis. Bit-wise indexing allows computing aggregations on-the-fly (pre-computing). For analytical queries indexes can be added to the existing layout to support analytical applications. It separates read and write nodes to execute procedures in parallel. This technique is useful for data aggregators or re-sellers because Sybase IQ can assign these nodes for a particular account. It also has a pre-defined schema called Relational Datacube (Rcube) that provides significant benefits comparing to the star-schema and reduces the complexity of star and snowflake schemas managing them under one table with the same shared key. The big difference is that this engine focuses to run large number of queries in parallel, rather than optimizing on how to split the query and run each part of the same query in parallel. Another difference is that unbalanced partitions are rare because of the same number of fields in each column. Figure 7 presents the architecture of Sybase. One node is dedicated to manage, update, and own the database and the single writer node prevents the necessary locking mechanism. Sybase has fair failure detection mechanisms without full automatic failover switch. HadoopDB with its advanced failover strategies outperforms Sybase IQ in the case of hardware error or node slowness.

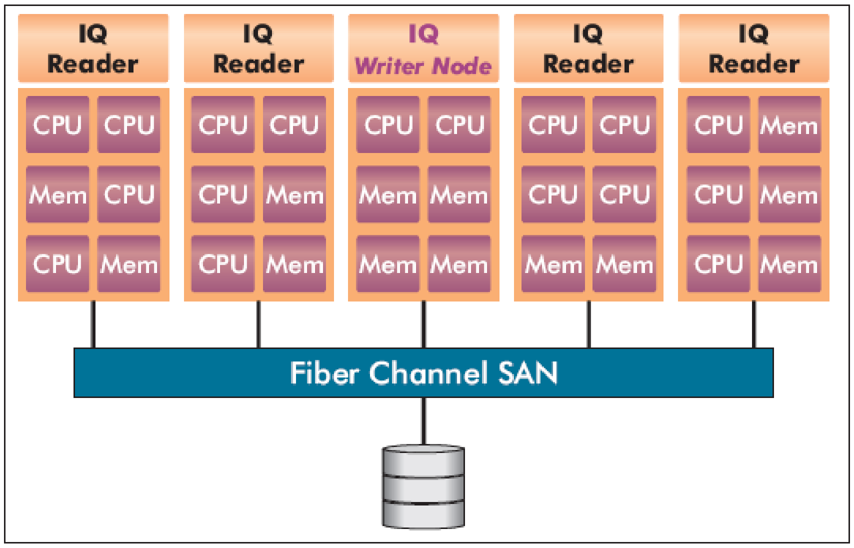

Figure 7: Sybase IQ Multiplex architecture [Howard, 2005]

The main goal of ParAccel [MacFarland, 2010] Analytic Database (PAD) is to scale-out for analytics applications. It focuses not just on fast data retrieval but handling the complex queries and complex joins. Comparing with others it uses a specific communication protocol for interconnecting the nodes. It also stores the data in columns. It adds encryption feature for data retrieval. If we compare the locking method of ParAccel with other parallel databases we can see its snapshot isolation process. ParAccel engine avoids locking by an internal version of the data using a transaction ID at table level that support full ACID integrity. ParAccel and HadoopDB both uses Postgres parser and optimizer for analytic operations. ParAccel added its own optimizer called OMNE for column based optimizations. HP Neoview [Winter, 2009] parallel database uses HP standard servers and storages. Neoview parallel transporter can perform parallel SQL inserts and updates while it is capable to handle server online queries. A separate instance of the database software runs on each node like HadoopDB's PostgreSQL instances. It supports hash partitioning of the tables using a specified partitioning key to form clusters of tables. Each partition is structured in B-tree table structure where the partition key can be the same as the hash key of the cluster (it is similar to HadoopDB's partition method). An addition to the other parallel databases systems is that it supports secondary B-tree indices on multiple columns for fast and frequent data retrieval queries that do not involve cluster columns (figure 8). With its unique indirect pointer based index structure the index re-organization of the tables is really easy. Huge difference that Neoview SQL optimizer is cost based and not rule based like HadoopDB. This engine is carefully designed to support mixed workloads like Dataupia but Neoview has its own single-point-of-failure elimination technique. Each disk drive is mirrored (Raid 1) and is accessible separately from different nodes. It has a similar task re-scheduling method to MapReduce that is capable to move the failed query to a different node and continue the processing without restarting all the related queries.

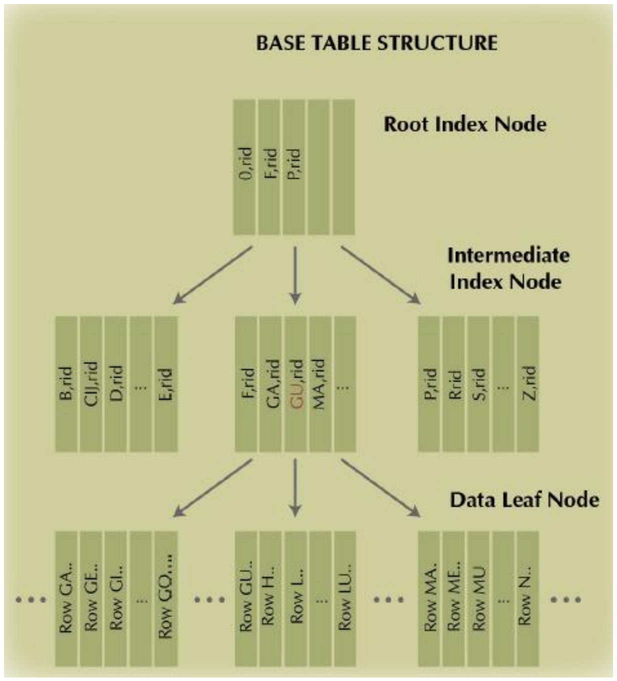

Figure 8: Neoview clustered data structure [Winter, 2009]

Oracle Exadata [Oracle, 2010] is designed to scale-out to different levels of performance. The architecture pushes as much SQL work as it can to the Exadata cells. If we compare Exadata with other architectures then the data retrieval method gives unique benefit. The storage returns only the requested rows and columns that fulfill the requirements of a query rather than querying the entire table itself (figure 9).

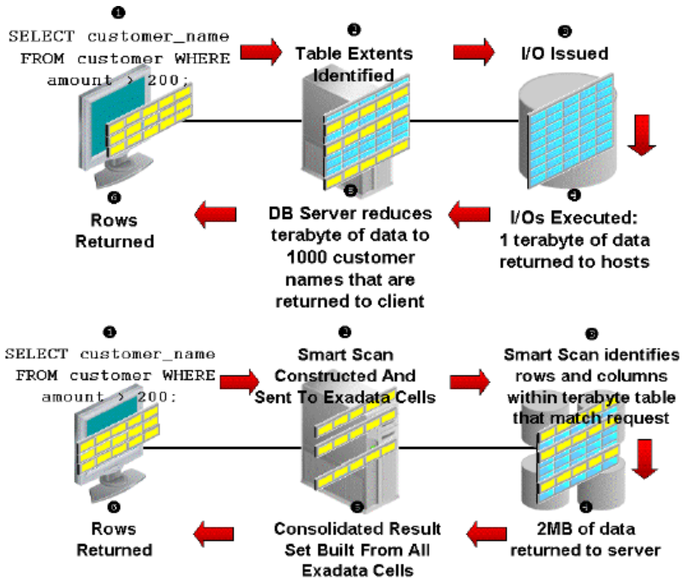
SELECT customer_name FROM customer WHERE amount > 200;
Table Extents Identified
I/O Issued
Rows Returned
DB Server reduces terabyte of data to 1000 customer names that are returned to client
I/Os Executed: 1 terabyte of data returned to hosts
SELECT customer_name FROM customer WHERE amount > 200;
Smart Scan Constructed And Sent To Exadata Cells
Smart Scan identifies rows and columns within terabyte table that match request
Rows Returned
Consolidated Result Set Built From All Exadata Cells
2MB of data returned to server

Figure 9: Parallel databases vs. Exadata data retrieval method [Oracle, 2010]

Another difference is the compression technique that Exadata uses. It decompresses only the rows and columns that are being returned to the client increasing the performance. It also has built in algorithms (checksums, block locations, alignment error, etc.) to find corrupted data and prevent that being saved to the storage. Hadoop detects the corrupted data while the Map or Reduce function crashes deterministically. Hadoop implements a sequence number with a signal handler that can catch different errors. This signal handler is processed by the master process with a special data packet. Master can skip the bad data based on this signal handler. Exadata supports "Flashback" history view of the records. A query can search historical data and perform change analysis. Greenplum [Greenplum, 2008] database is a massively parallel processing database where the server is based on PostgreSQL (figure 10). The servers behave like Dataupia's virtual pool and they form a single logical database with extended features like parallel external data loading, storage enhancement, etc. It has a dedicated master server where the clients can connect and submit SQL statements through PostgreSQL. This master is responsible for coordinating the tasks among other groups of database segments. These segments have the user data and the master has the global system catalog with metadata information.

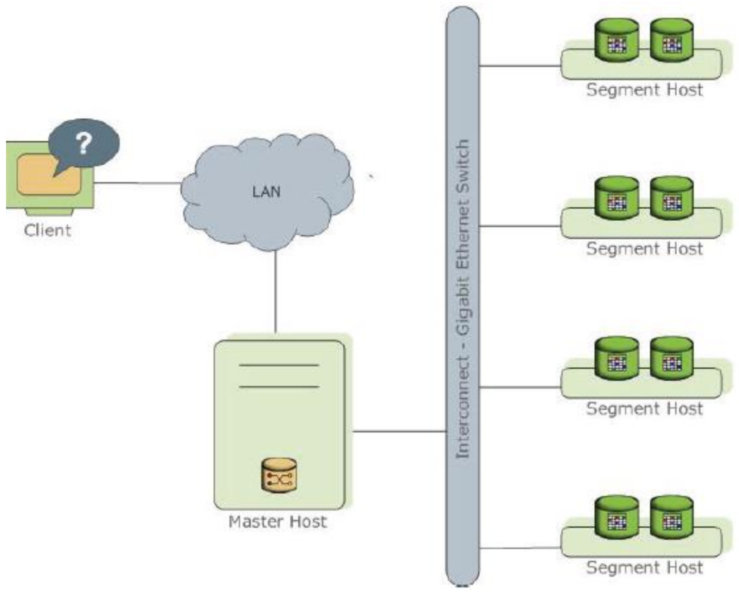

Figure 10: Greenplum architecture [Greenplum, 2008]

Greenplum implemented an intercommunication network between segments using User Datagram Protocol (UDP) to be able to scale-up and increase the number of segments. The mirroring of each segment is not automatic. The system administrator can set-up this feature with the possible fail-over methods (continue, fail, etc.). The master can be mirrored and the information between the failed master and the backup master is synced using log files. The segments contain collection of rows for each distributed table (figure 11) using a hashing algorithm. Each database operation is executed in parallel using the required segments simultaneously. As an addition to other parallel databases Greenplum has its own database operator called motion. Motion moves tuples around during the query processing.

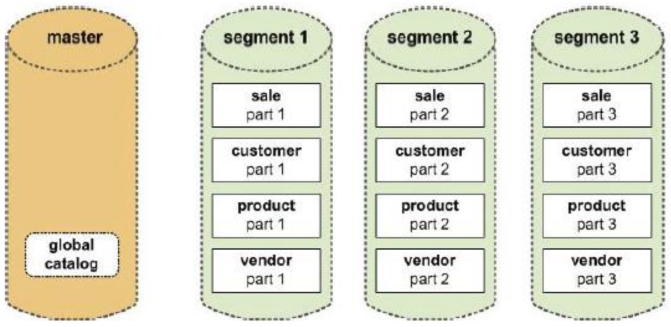

Figure 11: Greenplum data distribution [Greenplum, 2008]

In the case of a join the re-partitioning is not required because each segment will get the query plan and a redistributed motion moves tuples between the two segments to complete the join. This solution is different than the previous databases' methods.

MapReduce does not support real-time queries, random accesses, and the major features of a relational database engine. This is why different new storage systems were implemented and combined together. It runs over a distributed file system (GFS/HDFS) or a special storage system like HBase [Khetrapal & Ganesh, 2008] or Bigtable [Chang et al., 2008] to fulfill the real-time queries and ad-hoc accesses. PNUTS [Cooper et al., 2008] is Yahoo!'s hosted row based data serving platform. There is a significant difference between PNUTS and the other systems: PNUTS is a geographically distributed database system. It supports record-level mastering that handles most of the queries locally and makes all high latency operations asynchronous. Synchronous communication is not an option because to write data into multiple copies around the world can take a lot of time that exceeds the Web request handling time. The query language of PNUTS has a capability to handle selections and projections from a single table and updates/deletes queries with primary key only. The system is designed to handle simple reads and writes of a single record or form a group of records. It does not force constraints and it cannot handle complex ad-hoc queries. Write call gives ACID semantics as a transaction involving a single row. The schemas are flexible and rows can have arbitrary number of columns. PNUTS supports per-record timeline consistency. One of the replicas is designated as a master (which replica receives the most updates becomes the master) and all updated for a record are forwarded to the master. If a master fails to handle the request of a specific record the system tries to start a new master. If it is not possible then the query fails. The data is written to the master copy that propagates the records to the message broker. The message broker is responsible for replicating the data. The tables are horizontally partitioned into "tablets". The assignment of the tablets to a server is very flexible. The system can dynamically move the tablets from one server to another. The scatter-gather engine is responsible for splitting the multi record request into sub-requests and execute them in parallel. It resembles the results and propagates them back to the client. The system does not have a traditional database log nor does it archive data. It maintains a redo log that is capable to replay lost updates. The geographically replicated data provides additional protection. It can scale up to 10s of sites with 1000 nodes per site. HBase and Hypertable [Khetrapal & Ganesh, 2008] are the open-source implementation of Google’s Bigtable distributed storage system. Bigtable provides massive data organization by primary key and efficient data querying. It has the ability to store structured or semi-structured data without any specific schema. Dynamo [DeCandia et al., 2008] Amazon’s highly available key-vale store is a distributed storage system as well. However, its main focus is on write queries and Bigtable focuses on retrieval ones. All Bigtable like systems assume that writes are negligible. HBase defines the tables as regions where the region is specified by its name and startKey. Bigtable identifies them by its startKey and endKey. Each region can be on different nodes and consist of multiple Hadoop replicated HDFS files and blocks. It defines two special tables: root and meta. They are responsible for storing schema information and region locations. HBase is a fully consistent model compared to PNUTS. It has a columnar storage model that is ordered on rows and supports dynamic adjustment of the data layout. It has no transaction support and the replication mode is asynchronous. It can scale-up to 1000s of nodes. HBase has good support for analytic queries combined with Hadoop. Dynamo (figure 12) partitions and replicates the data using hashing and object versioning. The consistency is maintained by a complex double hashing method. It does not have pre-defined schema structures similarly to PNUTS. Amazon prefers high availability over consistency and it does not support ACID properties. This is common in the case of large distributed storage systems. Dynamo creates an always writable data store that should be highly available for writes (not like Bigtable or PNUTS). To execute a query it applies a sloppy quorum method.

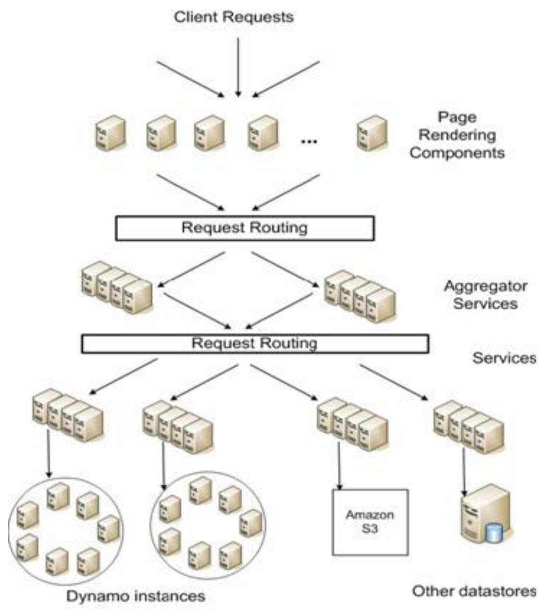

Figure 12: Dynamo architecture

R/W is the minimum number of nodes that must participate to execute a Read or Write operation successfully. It sets R + W > N that defines a quorum like system (N: number of nodes).

Q2) **Column-stores, Row-stores, and H-stores**
Assume you are working for a social networking company that serve many millions of users and 100 of PB of data on a regular basis. This system should be able to: (1) store the data, (2) support sophisticated report generation to identify the most active users, the hottest topics, and emerging trends in usage, and/or (3) allow for the users to interact in real-time through your site via chat, exchange of messages and blurbs. Your job is to do the background market research for your boss to provide him with a detailed analysis of the different data stores currently available, most notably, Column-stores, Row-stores, and H-stores, and to provide a summary of their respective features, key technical innovations and scopes of applicability. In your report to your boss, you must recommend which of the solutions you would pick - depending on how often or with what priority any of the 3 tasks above is to be undertaken. Support your final recommendation of technology selection carefully with solid technical arguments.

To design our system we have to select a framework which is capable to handle and process extremely large data sets, has support for frequent analytics queries and "live" features, highly scalable, and eliminates a possible single-point-of-failure at different levels. First, we have to consider different types of databases to build our system on a stable core. We can select column-store, row-store, or H-store databases.

- Commonly known Column-store databases

Choice a) MonetDB [Vermeij et al., 2008, Manegold et al. 2009] is an open-source high-performance column-store database. It combines the column store approach with a virtual fragmented storage model where the execution of the queries is CPU tuned. The key feature of the column based storage is that only a sub-set of a table is required during query processing. The engine can store every column in a separate table. This decreased table size helps to increase the performance because tuples can be stored in a disk-block. To convert a row-store database into a column-store a direct mapping can be applied using Vertical Partitioning. MonetDB has its own XQuery engine that is capable to query XML documents. Maybe, it is a good solution for storing blurbs in XML. The server uses MAL (MonetDB Assembly Language) algebra language to simplify SQL queries. A front-end compiler compiles SQL queries to MAL plans. This adds a multi level query optimization structure to the engine. The satellite project MonetDB/X100 [] adds extra performance on the top the column-store with vector processing and light-weight data compression. MonetDB architecture (figure 13) uses Decomposed Storage Module (DSM) to represent the vertical fragmentation storing each column in a separate table called Binary Association Table (BAT) with <head, tail> key pairs. Head or surrogate is the object identifier and the tail is the exact value. BAT is programmed via MAL. Complex expressions are broken into BAT algebra operators and each operator bulk process (single operation on the entire values of a column) and entire column. MonetDB also provides materialized views (similar to Vertica's projections) to combine different columns into one virtual table.

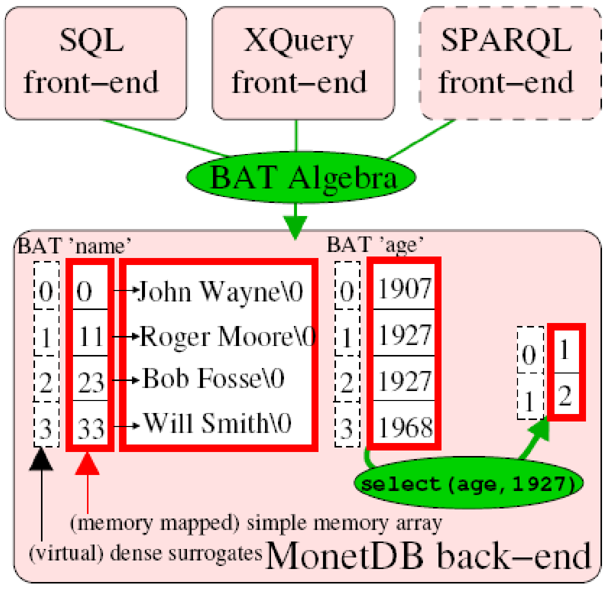

Figure 13: MonetDB architecture [Manegold et al., 2009]

Choice b) C-Store [Stonebraker et al., 2005] stores a group of columns sorted on different attributes. Similarly to Vertica it introduces projections as a group of columns sorted on the same attribute. The same column can exist in multiple projections. C-Store tupple mover transforms the readable column-store into an update and inserts supported writable approach. Tupple mover operates between a Writable Store (WS) and a Read-optimized Store (RS). C-Store implements the updates as an insert and a delete operation. WS is capable to handle frequent inserts and updates. This can be a positive feature of the chat application where frequent messages can be stored in the database (if we want to store them in the database instead of a distributed filesystem or in memory). This architecture supports ad-hoc queries and transactions as well. In the case of a possible transaction support C-Store can be a good choice. C-Store provides snapshot isolation and timestamps each element upon insert. C-Store heavily compresses the content of the columns to further increase the performance (for further techniques e.g. Late Materialization, Block Iteration, Invisible Join see Q3). The C-Store query language is SQL and it supports primary-key/foreign-key assignments. Storage Allocator allocates segments to nodes. To recover from a possible failure it applies K-safety (Vertica uses the same technique) that maintains sufficient replicas so that K sites can fail within a time frame without a total system crash.

Choice c) Vertica Analytic Database [Vertica, 2008] utilizes cheap shared-nothing commodity hardware and it is designed for large scale data warehouses (figure 2). It is mostly for read intensive analytical applications where the system accesses a subset of columns. Vertica's optimizer employs various data compression techniques to minimize the space requirements of the columns. The optimizer stores views of the table data in projections. The projection can contain a subset of the columns of a table or multiple tables to support materializing joins.

Projections are created automatically by Vertica to support ad-hoc queries. To avoid node failure Vertica creates k+1 copies of the projections (k is the total number of nodes) and fully replicate them. In the case of a failure, it automatically switches to the next available instance. It has a built in automatic physical database design tool that creates these projections automatically and targets star (fact and dimension tables where fact tables are range partitioned across the nodes and dimension tables are replicated) or snowflake (normalization of dimension tables) schemas for automatic design.

Choice d) The main goal of the ParAccel [MacFarland, 2010] Analytic Database (PAD) is to scale-out for analytics applications. It focuses not just on fast data retrieval but handles the complex queries and complex joins (in our cases this is an important feature to fulfill the second request) as well. Compared with others it uses a specific communication protocol for interconnecting the nodes. It adds encryption feature for data retrieval that can hide the content of the chat messages and keep the user privacy. If we compare the locking method of ParAccel with other parallel databases we can see its snapshot isolation process. ParAccel engine avoids locking by an internal version of the data using a transaction ID at table level that support full ACID integrity. ParAccel utilizes PostgreSQL parser and optimizer for analytic operations to achieve the best performance.

Choice e) Sybase IQ [Howard, 2005] is an analytics parallel column-store that provides real-time analysis as well (again, in our case this is really important). Similarly to others because it supports SQL language, database schemas, and indexing but it has its own specialized indexes for e.g. time series analysis. Time-series analysis can be significant for determining the emerging trends in usage. Bit-wise indexing allows computing aggregations on-the-fly (pre-computing). For analytical queries indexes can be added to the existing layout to support analytical applications. It separates read and write nodes to execute procedures in parallel. The big difference - compared to the previous choices - is that this engine focuses on how to run large number of queries in parallel, rather than optimizing on how to split the query and run each part of the same query in parallel. Sybase has fair failure detection mechanisms without full automatic failover switch (See question 1 part f for more details on Sybase IQ).

Choice f) InfoBright [Slezak & Eastwood, 2009] has two versions: community edition (open-source) and enterprise edition. The system focuses on analytical ad-hoc queries. It partitions the data into Rough Rows and labels them with value information on data columns. InfoBright creates objects corresponding to these Rough Rows and labels. InfoBright utilizes MySQL [MySQL2010] with MyISAM [MyISAM2010] engine to store table information (e.g. user permissions, table definitions) (figure 14). It handles complex queries using MySQL parser and optimizer. It defines Knowledge Grid between the query engine and the data that stores Rough information. Knowledge Grid includes data pack nodes that are linked to Data Packs. Columns are divided into 64K values (Data Pack). A Data Pack is linked with the Knowledge Grid that stores the metadata. InfoBright has a different concept than the previous engines. The

Knowledge Grid applies Rough approximations for the queries to assemble the required data parts and to minimize the data decompression.

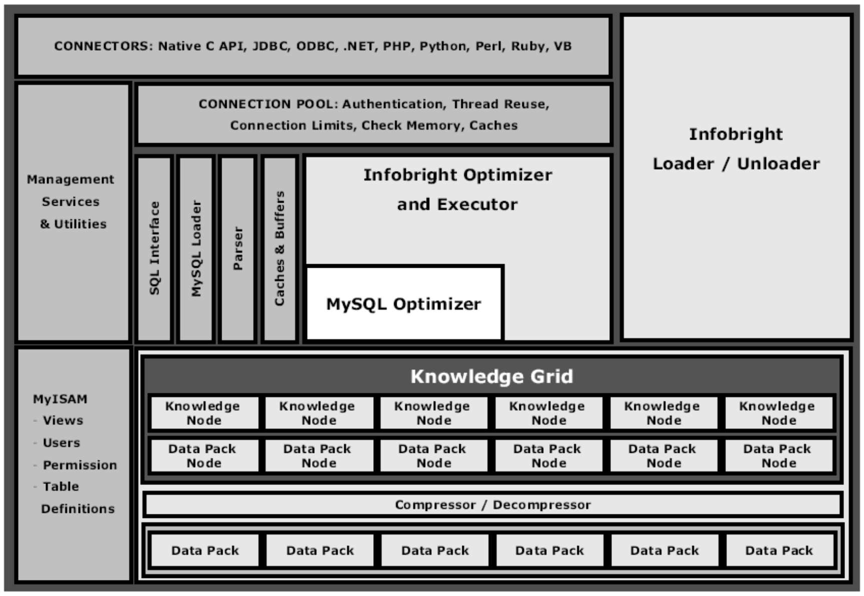
CONNECTORS: Native C API, JDBC, ODBC, .NET, PHP, Python, Perl, Ruby, VB
Infobright Loader / Unloader
Management Services & Utilities
CONNECTION POOL: Authentication, Thread Reuse, Connection Limits, Check Memory, Caches
SQL Interface
MySQL Loader
Parser
Caches & Buffers
Infobright Optimizer and Executor
MySQL Optimizer
MyISAM
- Views
- Users
- Permission
- Table Definitions
Knowledge Grid
Knowledge Node
Data Pack Node
Compressor / Decompressor
Data Pack

Figure 14: InfoBright architecture [Slezak & Eastwood, 2009]

A big difference is that the Knowledge Grid is a statistical descriptor of the records and it handles the indexes of the loaded data (index creation is not required on a table in InfoBright). InfoBright has a great data compression ratio (10-to-1) that is similar to Vertica. InfoBright query optimizer connects to the Knowledge Grid to collect the location of each data pack needed to serve the query. Queries can be exceptionally fast if the required number of the data packs is small.

- Commonly known Row-store databases

Netezza (choice g), Teradata (choice h), Dataupia (choice i), IBM DB2 (choice j), Oracle Exadata (choice k), NeoView (choice l), and Greenplum (choice m) (for database descriptions see question 1 part a and f).

Choice n) MySQL – InnoDB Storage Engine

InnoDB [InnoDB2010] is fully ACID (Atomicity, Consistency, Isolation, Durability) compliant storage engine for MySQL that uses two-phase commit and transaction logs. It stores data in clustered indexes based on primary keys and supports primary-key/foreign-key assignments similarly to C-Store. This engine has its own buffer pool for caching data in RAM. It has its own tablespace to store tables and indexes. InnoDB can horizontally partition the tables into separate files based on a hash key. InnoDB Hot Backup feature allows the DBA (Database Administrator) to backup a running database without interrupting the query processing. It has a double-write buffer and automatic checksum feature to prevent the system from corrupted data and hardware failures. It implements row-level locking for inserts and updates. The engine is fast for writes and updates because of the row-level locking. If the project processes large amount of data and requires transaction support then InnoDB is a perfect solution. It can consume a lot of RAM.

- H-store database

H-Store [Stonebraker et al., 2007, Kallman et al., 2008] is designed to run in RAM and makes the assumption that most of the On-line transaction processing (OLTP) tasks are short running. It is ACID compliant by running numerous RAM based replicated copies. H-Store does not write to the disk synchronously and the system recovery is based on other RAM copies. It claims that disk oriented indexing is not required if the database is in memory. H-Store operates over shared-nothing distributed machines where the data is entirely loaded into the memory. H-Store requires the complete workload to be given with all the transaction classes and available nodes. It also assumes that the group of tables that the transaction operates on is also known in advance. H-Store replicates the content of each table and updates them periodically. Transactions are timestamped like in the case of C-Store.

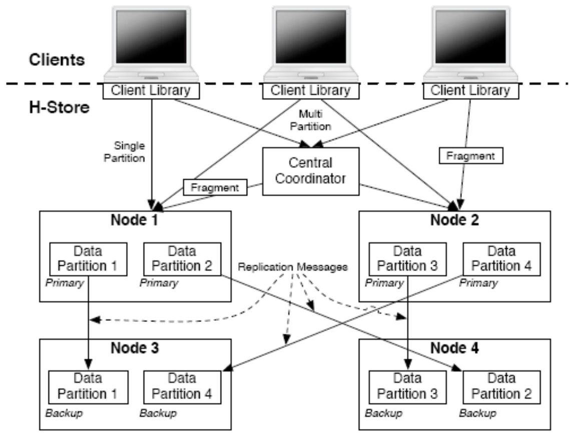

Figure 15: H-Store architecture [Jones et al., 2010]

The system has three main processes (figure 15). Every read-only table is replicated across the site and all tables horizontally partitioned into four partitions. These partitions are stored on two different sites. The data is stored in partitions and the central coordinator is responsible for distributing transactions [Jones et al., 2010]. The transactions are processed in stored procedures. A transaction is Single-sited if the queries can be executed using a single site of the cluster (site is where the application connects to execute a transaction). One-shot Transaction’s individual queries will be executed on only one site. The system applies K-safety technique to replicas similarly to Vertica or C-Store but here K=2 because of the possible memory bottleneck. H-Store can be 80 times faster than the fellow databases.

- Designed system architectures

I. Significance factor: Store Data

System 1: Hadoop and Hadoop Distributed File System (HDFS), Hive, and HBase (see question 1 part a for Hadoop, HDFS, Hive and see part e-f for HBase)

Why factor: HadoopDB (open-source) targets the performance and scalability of a parallel database and the fault-tolerance feature of the flexible MapReduce to achieve better structured data processing. HadoopDB parallelizes the queries across nodes. A parallel database itself is not designed for a large scale heterogeneous environment (thousands of nodes). Concurrent queries, node disk fragmentations, or corrupted data parts can decrease the performance of the system. HadoopDB can handle these problems well. It can schedule parallel execution of the same task on different node if it detects that the data processing is slow. Furthermore it can catch the process before it terminates because of a bad data segment and re-schedule the task using data from different location. One of the first key issues is the data distribution of the system. A highly scalable distributed file system is necessary to handle large amounts of data and to eliminate a possible performance bottleneck. This file system should be fast and fault-tolerant for a possible node failure. HDFS (Hadoop Distributed File System) stores the data in fixed size blocks and distributes it across multiple nodes. Hive adds an SQL interface to the system to be able to easily communicate without writing Map and Reduce functions. Hadoop connected with Hive currently does not support INSERT INTO, UPDATE, and DELETE statements (our application has apply insert, update, delete queries as well). However, Hadoop combined with HBase (the clone of Google BigTable) column oriented semi-structured data store provides additional features. It runs on the top of Hadoop HDFS file system and provides CRUD: Create, Read, Update, and Delete features for the application. Also, combining the system with HBase allows handling read and writing concurrencies without specific locking mechanism. HBase is also good support for possible analytic queries.

Additional system components: Application server and Web server

System 2: Using an Online Transaction Processing System (OLTP)
MySQL InnoDB Engine (middle-end, choice n) or H-Store (high-end) combined with Memcache chaching technology to cache frequent queries.

Why factor: H-Store is extremely fast and handles transaction processing well. Because it is a Web-based application we can know the workload, transaction classes, and table schemas beforehand. These are the required inputs using H-Base memory store. Memcache can easily speed-up the retrieval queries. MySQL InnoDB is an optional choice if we do not know the workload but we need transaction support.

Additional system components: Application server and Web server

II. Significance factor: sophisticated report generation to identify the most active users, the hottest topics, and emerging trends in usage (analytical queries)

System 1: Hadoop and Hadoop Distributed File System (HDFS), Hive, and a choice of an Analytical database

Analytical Database: row based choice: Teradata (choice h) or Netezza (choice g), column based choice: Vertica (choice c), ParAccel (choice d), or InfoBright (choice f)

Why factor: same as in the case of I/System 1. Database should support transactions, analytical queries, writes, updates, and deletes.

Additional system components: Application server and Web server

III. Significance factor: system allows for the users to interact in real-time through the site via chat, exchange of messages and blurbs

System 1: Hadoop and Hadoop Distributed File System (HDFS), Hive, and MonetDB (choice a) or H-Store

Why factor: MonetDB combines the column store approach with a virtual fragmented storage model where the execution of the queries is CPU tuned. This can be really handy because the load of the chat messages is changing frequently. The engine can store every column in a separate table. This decreased table size helps to increase the performance because tuples can be stored in a disk-block. We can expect lot of chat messages. MonetDB has its own XQuery engine that is capable to query XML document. Maybe, it is a good solution for storing chat messages and blurbs in XML. The server uses MAL (MonetDB Assembly Language) algebra language to simplify SQL queries. A front-end compiler compiles SQL queries to MAL plans. This adds a multi level query optimization structure to the engine to retrieve XML information quickly. H-Store is a good choice as well. In this case, all messages are stored in memory and users can quickly retrieve and modify them.

Note: We can store the messages on a simple distributed file system as well using Hadoop, HDFS, Hive, and H-Base.

Additional system components: Application server and Web server

I+II+III. All of them are significant: In this case we propose a multi level architecture.

Level 1 (analytical queries): Hadoop and Hadoop Distributed File System (HDFS), Hive, and a choice of an Analytical database

Analytical Database: row based choice: Teradata (choice h) or Netezza (choice g), column based choice: Vertica (choice c), ParAccel (choice d), or InfoBright (choice f)

Level 2 (mostly retrieval queries, chat messages, and blurbs): Using an Online Transaction Processing System (OLTP)
MonetDB (choice a) or H-Store combined with Memcache chaching technology to cache frequent queries.

Additional system components: Application server and Web server

Q3) **Data-servers for Data-Intensive Web Site Management**
As part of your dissertation research, you have been developing data partitioning and replication recommendation techniques to increase the efficiency of query processing. Consider which aspects of your technology, if any, were to change, and if so how, if you were to replace your assumption of utilizing a row-store relational database (such as, Oracle) as underlying DBMS by a column-store relational database (such as Vertica)?

Column oriented databases (Vertica, Sybase IQ, MonetDB, MonetDB/X100, C-Store, Paraccel, etc.) store each database table column separately with attribute values compressed. Processing the sub-set of the tables' columns makes the performance faster.
We assume that the original database (origin) and the new databases (targets) are row-store relational databases. If the origin is a column store system and the targets are column store structures as well we have to change the aspects of our technology. We utilize four operators: Replication, Horizontal Partitioning, Vertical Partitioning, and De-normalization. First of all, Vertical Partitioning is the most straightforward way to convert a row-based system into a column-based one if each relation is fully vertically partitioned [Khoshafian at al., 1987]. Our system can apply replication and replicate required columns based on the query workload. For example if a retrieval query frequency is high the column could be fully replicated. If the system has a node failure then mirrored columns can be utilized. De-normalization can be useful to create projections. In the case of Vertica the optimizer stores views of the table data in projections. The projection can contain a subset of the columns of a table or multiple tables to support materializing joins. With projections we can get all the required columns together to complete possible joins. Moreover, full replication can be applied on projections to avoid a possible system failure (system can switch to the next available instance automatically). Horizontal Partitioning can be an interesting operator. If we apply range or hash based

Horizontal Partitioning on a column we will create more columns based on the specific key. This could be really powerful if it is combined with a correct data compression method to further reduce the size of the columns and increase the total system performance. Range based Horizontal Partitioning can be combined with the data header information of a column. This means the header of the column contains a range descriptor about the stored data interval. Queries can easily ignore the column if the predicate has different intervals than the column header has. In both cases we need a pointer net that points from one partition or range to the next one of the same column. The query router of the system needs to be modified as well. The router should consider advanced techniques such as Late Materialization, Block Iteration, Invisible Join, and data compression, related to column-store to fulfill our requirements that each query should be answerable by a single node. Late Materialization is where the columns can be joined together to form rows as late as they can in the query plan [Abadi et al., 2007]. Block Iteration [Zukowski et al., 2005] where blocks of values can be retrieved from a column and passed to the next operator (IBM DB2). Invisible join converts a join into a set of predicates that can be faster and it can operate on columns.
If our origin is a row-oriented and our targets are column oriented databases then before any modification we have to convert the row-oriented database into column-store applying Vertical Partitioning operator to create column view.

References

1. Jeffrey Dean, Sanjay Ghemawat, and Google Inc. Mapreduce: simplified data processing on large clusters. In In OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design and Implementation, 2004.

2. Dhruba Borthakur (2008). HDFS Architecture. Apache Software Foundation. http://hadoop.apache.org/common/docs/current/hdfs_design.pdf

3. Azza Abouzeid, Kamil B. Pawlikowski, Daniel J. Abadi, Alexander Rasin, and Avi Silberschatz. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. PVLDB, 2(1):922-933, 2009.

4. Hadoop (2009). Web Page. http://hadoop.apache.org/core/

5. Ashish Thusoo, Joydeep S. Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wycko_, and Raghotham Murthy. Hive: a warehousing solution over a map reduce framework. Proc. VLDB Endow., 2(2):1626-1629, 2009.

6. PostgreSQL (2009). Web Page. http://www.**postgresql**.org/

7. Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. A comparison of approaches to large-scale data analysis. In SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data, pages 165-178, New York, NY, USA, 2009. ACM.

8. Vertica, 2009. The Vertica Analytic Database – Introducing a New Era in DBMS Performance and Efficiency.
http://www.redhat.com/solutions/intelligence/collateral/vertica_new_era_in_dbms_performance.pdf

9. George S. Davidson, Kevin W. Boyack, Ron A. Zacharski, Stephen C. Helmreich, and Jim R. Cowie. Data-Centric Computing with the Netezza Architecture. SANDIA REPORTSAND2006 3640 Unlimited Release Printed April 2006,
http://www.netezza.com/documents/whitepapers/Sandia_Labs_White_Paper_July_06.pdf

10. Sue Clarke. Teradata. Butler Group Research Paper. October 2000,
http://www.teradata.com/library/pdf/butler_100101.pdf

11. Rao, J., Zhang, C., Lohman, G., and Megiddo, N. Automating Physical Database Design in a Parallel Database. Proceedings of the ACM SIGMOD 2002
12. Zilio, D., Jhingran, A., Padmanabhan, S. Partitioning Key Selection for Shared-Nothing Parallel Database System. IBM Research Report RC 19820. 1994

13a. Patvarczki, J. "Developing an Architecture to Search for When Different Parallelization Operations are effective: An attempt to apply machine learning to database parallelization", Ph.D. proposal, 2010, http://users.wpi.edu/~patvarcz/PhD_Proposal_Jozsef.pdf

13b. Patvarczki J. "Developing an Architecture to Search for When Different Parallelization Operations are effective: An attempt to apply machine learning to database parallelization: Addition", Ph.D. proposal, 2010, http://users.wpi.edu/~patvarcz/WebApplications.pdf

14 Jozsef Patvarczki, Murali Mani, and Neil Heffernan, "Performance Driven Database Design for Scalable Web Applications", Advances in Databases and Information Systems, In J. Grundspenkis, T. Morzy & G. Vossen (Eds) Advances in Databases and Information Systems Springer-Verlag: Berlin. pp 43-58

15. Sean Michael Kerner. Inside Facebook's Open Source Infrastructure. 2010.
http://www.developer.com/open/article.php/3894566/Inside-Facebooks-Open-Source-Infrastructure.htm

16. Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu and Raghotham Murthy. Hive – A Petabyte Scale Data Warehouse Using Hadoop. Facebook Data Infrastructure Team report, 2010.
http://i.stanford.edu/~ragho/hive-icde2010.pdf

17. Ankur Khetrapal & Vinay Ganesh. HBase and Hypertable for large scale distributed storage systems A performance evaluation for Open Source BigTable Implementations. 2008. http://www.uavindia.com/ankur/downloads/HypertableHBaseEval2.pdf

18. Dataupia (2008a). Dataupia Satori Server. http://www.dataupia.com/pdfs/productoverview/Dataupia%20Product%20Overview.pdf

19. Dataupia Satori Server with Dynamic Aggregation (2008b). http://www.dataupia.com/pdfs/productoverview/Dataupia%20Satori%20Servier%20with%20Dynamic%20Aggregation%205_19_2008.pdf

20. Philip Howard. Sybase IQ. An evaluation by Bloor Research. 2005. http://www.sybase.com/content/1035804/SybaseIQ_bloor-report.pdf

21. Anne MacFarland. The Speed of ParAccel's Data Warehousing Solution Changes the Economics of Business Insight. The Clipper Group Navigator, Report #TCG2010008, 2010. http://www.clipper.com/research/TCG2010008.pdf

22. Richard Winter. HP Neoview Architecture and Performance. WinterCorp White paper, Waltham, MA. 2009. http://h20195.www2.hp.com/v2/GetPDF.aspx/4AA2-6924ENW.pdf

23. Oracle (2010). A Technical Overview of the Sun Oracle Database Machine and Exadata Storage Server. An Oracle White Paper. http://www.oracle.com/technetwork/database/exadata/exadata-technical-whitepaper-134575.pdf

24. Greenplum (2008). Greenplum Database 3.2 Administrator Guide. http://docs.huihoo.com/greenplum/GPDB-3.2-AdminGuide.pdf

25. Khetrapal, Ankur; Ganesh, Vinay. (2008). HBase and Hypertable for large scale distributed storage systems. http://www.uavindia.com/ankur/downloads/HypertableHBaseEval2.pdf

26. Fay Chang, Je_rey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. (2008) Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst., 26(2):1-26, June

27 .Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans A. Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. (2008) PNUTS: Yahoo!'s hosted data serving platform. Proc. VLDB Endow., 1(2):1277-1288

28. Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels (2007). Dynamo: amazon's highly available key-value store. SIGOPS Oper. Syst. Rev., 41(6):205-220

29. Maarten Vermeij, Wilko Quak, Martin Kersten, Niels Nes. MonetDB, a novel spatial column-store DBMS. Inge Netterberg and Serena Coetzee (Eds.); Academic Proceedings of the 2008 Free and Open Source for Geospatial (FOSS4G) Conference, OSGeo, pp. 193-199. http://www.gdmc.nl/publications/2008/MonetDB.pdf

30. Manegold, S., Kersten, M. L., and Boncz, P. 2009. Database architecture evolution: mammals flourished long before dinosaurs became extinct. *Proc. VLDB Endow.* 2, 2 (Aug. 2009), 1648-1653.

31. Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., O'Neil, P., Rasin, A., Tran, N., and Zdonik, S. 2005. C-store: a column-oriented DBMS. In *Proceedings of the 31st international Conference on Very Large Data Bases* (Trondheim, Norway, August 30 - September 02, 2005). Very Large Data Bases. VLDB Endowment, 553-564.

32. Dominik Slezak and Victoria Eastwood. **Data Warehouse Technology by Infobright.** *SIGMOD'09,* June 29–July 2, 2009, Providence, Rhode Island, USA

33. MySQL (2010), MySQL 5.5, http://dev.mysql.com/tech-resources/articles/introduction-to-mysql-55.html

34. MyISAM (2010) MySQL 5.5, http://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html

35. InnoDB (2010) MySQL 5.5, http://dev.mysql.com/doc/refman/5.5/en/innodb.html

36. Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N., and Helland, P. 2007. The end of an architectural era: (it's time for a complete rewrite). In *Proceedings of the 33rd international Conference on Very Large Data Bases* (Vienna, Austria, September 23 - 27, 2007). Very Large Data Bases. VLDB Endowment, 1150-1160.

37. Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., Jones, E. P., Madden, S., Stonebraker, M., Zhang, Y., Hugg, J., and Abadi, D. J. 2008. H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.* 1, 2 (Aug. 2008), 1496-1499.

38. Jones, E. P., Abadi, D. J., and Madden, S. 2010. Low overhead concurrency control for partitioned main memory databases. In *Proceedings of the 2010 international Conference on Management of Data* (Indianapolis, Indiana, USA, June 06 - 10, 2010). SIGMOD '10. ACM, New York, NY, 603-614.

39. S. Khoshafian, G. Copeland, T. Jagodis, H. Boral, and P. Valduriez. A query processing strategy for the decomposed storage model. In ICDE, pages 636-643, 1987.

40. D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden. Materialization strategies in a column-oriented DBMS. I ICDE, pages 466–475, 2007.

41. M. Zukowski, P. A. Boncz, N. Nes, and S. Heman. MonetDB/X100 - A DBMS In The CPU Cache. IEEE Data Engineering Bulletin, 28(2):17–22, June 2005.