GEO DISTRIBUTED PARALLELIZATION PACTS IN MAP REDUCE FRAMEWORK IN MULTIPLE DATACENTER

C.Kirubanantham[1], C.Rajavenkateswaran[2]

[1]*Student, computer science engineering, Nandha College of Technology, India*
[2]*Assistant professor, Information Technology, Nandha College of Technology, India*

Abstract

MapReduce framework in Hadoop plays an important role in handling and processing big data. Hadoop is scalable that is it can reliably store and process petabytes. MapReduce works by dividing input files into chunks and processing these in a series of parallelizable steps. MapReduce framework offers a response to the problem by distributing computations among large sets of nodes. we concentrate on geographical distribution of data for sequential execution of MapReduce jobs to optimize the execution time. The fixed execution strategy of MapReduce program is not optimal for many task and as it does not know about the behavior of the functions. Thus, to overcome these issues, we are enhancing our proposed work with parallelization contracts. The parallelization contracts include input and output contract which includes the constraints and functions of data execution. These contracts help to capture a reasonable amount of semantics for executing any type of task with reduced time consumption.

Keywords— *Bigdata, Datacenter, Geo-distributed, Hadoop, MapReduce, PACT*

----------***----------

1. INTRODUCTION

Data comes from many sources such as social media websites, sensors to gather climate information, trajectory information, transaction records, other web site usage data etc. it is called Big data. The limits to what can be done are often times due to how much data can be processed in a given time-frame. Parallelization contract framework have become one of the toolkit for processing large datasets using cloud computing resources, and are provided by most cloud vendors. GEO-PACT, a system for efficiently processing geo-distributed big data GEO-PACT is a Hadoop based framework that can efficiently perform a sequence of Parallelization Contracts jobs on a geo-distributed dataset across multiple datacenters. Sequences of Parallelization Contracts jobs are executed on a given input by applying the first job on the given input, applying the second job on the output of the first job, and so on. GEO-PACT acts much like the atmosphere surrounding the clouds. The problem of executing geo-distributed Parallelization Contracts job sequences as arising in "cloud of clouds" scenarios is analyzed for job execution. Executing individual Parallelization Contracts jobs in each datacenter on corresponding inputs and then aggregating results is defined as multiple execution path. The datacenter with optimized execution path is selected for job execution.

2 HDFS

The Hadoop Distributed File System (HDFS). Provides redundant storage for massive amounts of data using inexpensive commodity hardware and serves as the large scale data storage system. The NameNode splits large files into fixed sized data blocks which are scattered across the cluster. Typically the data block size for the HDFS is conFigd as 64MB, but it can be conFigd by file system clients as per usage requirements. The data storage is of type write once/read many (WORM) and once written, the files can only be appended and cannot be modified to maintain data coherency. Since HDFS is built on commodity hardware, the machine fault rate is high. In order to make the system failure tolerant, data blocks are replicated across multiple DataNodes**.**

2.1 NameNode

NameNode is important for HDFS to store its metadata reliably. Furthermore, while the file data is accessed in a write once and read many model, the metadata structures (e.g., the names of files and directories) can be modified by a large number of clients concurrently. It is important that this information is never desynchronized. Therefore, it is all handled by a single machine, called the NameNode. The NameNode stores all the metadata for the file system. Because of the relatively low amount of metadata per file (Information about file locations in HDFS, Information about file ownership and permissions, Names of the individual blocks, Locations of the blocks of each file), all of this information can be stored in the main memory of the NameNode machine, allowing fast access to the metadata.

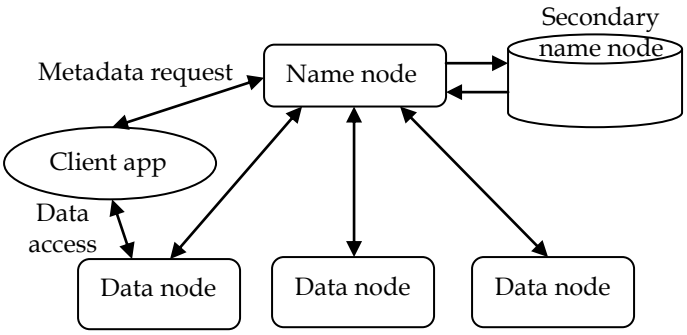

Fig 1.1 HDFS Architecture

2.2 Secondary NameNode

The Secondary NameNode is not a failover NameNode and it performs memory-intensive administrative functions for the NameNode. Secondary NameNode should run on a separate machine in a large installation. The secondary name-node is to perform periodic checkpoints. The secondary name-node periodically downloads current name-node image and edits log files, joins them into new image and uploads the new image back to the (primary and the only) name-node. If the name-node fails and it will restart on the same physical node then there is no need to shutdown data-nodes, just the name-node need to be restarted. If the old node cannot use anymore you will need to copy the latest image somewhere else. The latest image can be found either on the node that used to be the primary before failure if available; or on the secondary name-node.

2.3 DataNode

A DataNode is a storage server that accepts read/write requests from the NameNode. DataNodes store data blocks for local or remote clients of HDFS. Each data block is saved as a separate file in the local file system of the DataNode. The DataNode also performs block creation, deletion and replication as a part of file system operations. For keeping the records up-to-date, the DataNode periodically reports all of its data block information to the NameNode. DataNode instances can talk to each other for data replication. To maintain its live status in the cluster, it periodically sends heartbeat signals to the NameNode. When the NameNode fails to receive heartbeat signals from the DataNode, it is marked as a dead node in the cluster.

3. GEO-DISTRIBUTION

Applications on cloud environment are geographically distributed, for the reasons: to access data frequently; data's are collected and stored by different organizations to shared towards a common goal; datasets are replicated across datacenters for availability. Geo-PACT is a Hadoop-based system that can efficiently process concurrent jobs on a geographically distributed dataset.

Consider n datacenters, given by DC1, DC2…DCn with input sub datasets I1, I2,…,In respectively. The total amount of input data is thus $|I|=\Sigma^{n}_{i=1}|I_i|$. The bandwidth between the datacenters DC_i and DC_j (i!= j) is given by $B_{i,j}$ and the cost of transmitting one unit of data between two datacenters is $C_{i,j.}$ We define an initial minimum partition size of input dataset to be transferred across datacenters. On this geo-distributed input dataset sequence of jobs J1, J2,…Jn have to be executed. In each job parallelization contract (PACTs) is applied. PACTS consist of input and output contract. Input contract consist of five contracts and output contract may be optional.

4 THE PACT PROGRAMMING MODEL

The PACT Programming Model is an extension of map/reduce programming model. It operates over a key/value couple data model so called Parallelization Contracts (PACTs). Following the PACT programming model, programs are implemented by providing task specific user code (the user functions, UFs) for selected PACTs and assembling them to a work flow. A PACT defines properties on the input and output data of its associated user function. Figure 2 shows the parts of a Parallelization Contracts, contains only one Input Contract and an optional Output Contract. The Input Contract of a Parallelization Contracts designates how the user function can be evaluated in parallel. Optional output Contracts allow the optimizer to infer certain properties of the output data of a user function and to create a more efficient execution strategy for a program.

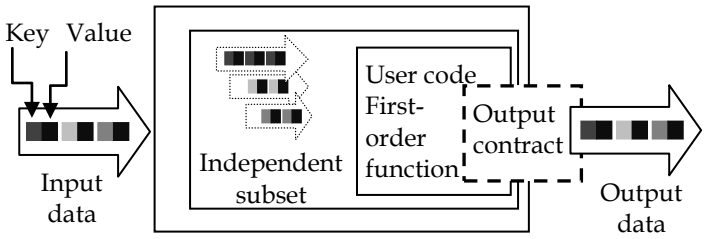

Fig 4.1 PACT Components

4.1 The Map Input Contract

The Map contract is used to process each key/value couple independently; the Map input contract states that the user code is invoked only once for each key-value couple of the input data set.

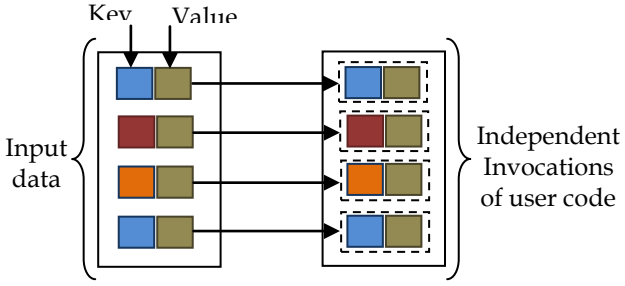

Fig 4.2 Map Input Contract

4.3 The Reduce Input Contract

PACT Reduce input contract, in which all key-value couple of a PACTs input data are grouped with an identical key. The user code is attached to the Reduce contract is invoked for each of these groups independently.

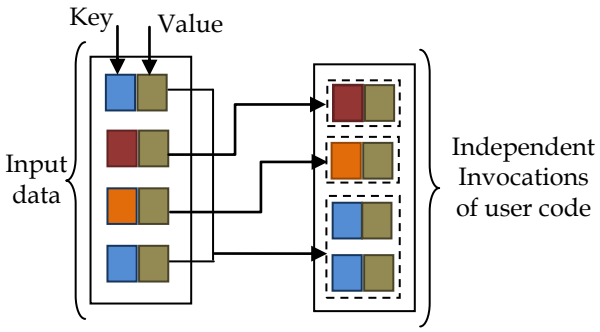

Fig 4.3 Reduce input contract

4.4 The Cross Input Contract

Multi-input contracts is the Cross input contract. The user code attached to those multi-input contracts expects to receive data from two specific data sources as input. Multi-input contracts also construct the subsets for the user code based on two different input set and builds the Cartesian product of the two inputs. All couple in the Cartesian product are then processed independently by separate calls of the user code.

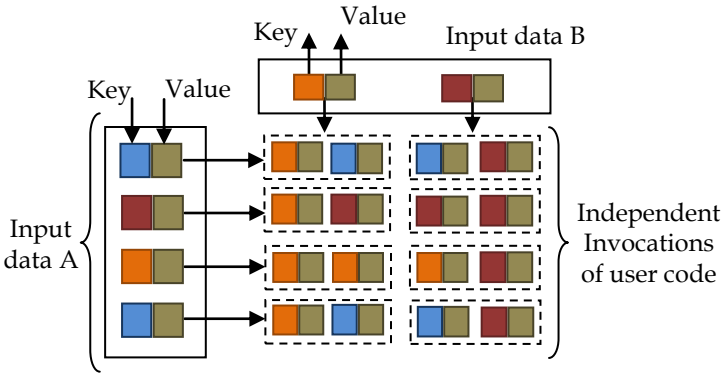

Fig 4.4 Cross input contract

4.5 The CoGroup Input Contract

CoGroup input contract, which is also a multi-input contract. Independent subsets are built by joining the groups with same keys of all inputs. The key/value couple of all inputs with the same key are assigned to the same subset.

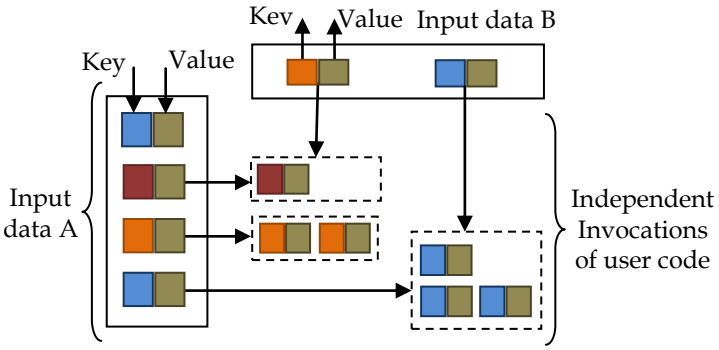

Fig 4.5 Cogroup input contract

4.6 The Match Input Contract

The Match input contract is a multiple input contract. All combinations of key-value couples with same keys are built in the input data sets. After all groups are processed independently by separate invocations of the attached user code. The Match input contract performs equi-join on the key.

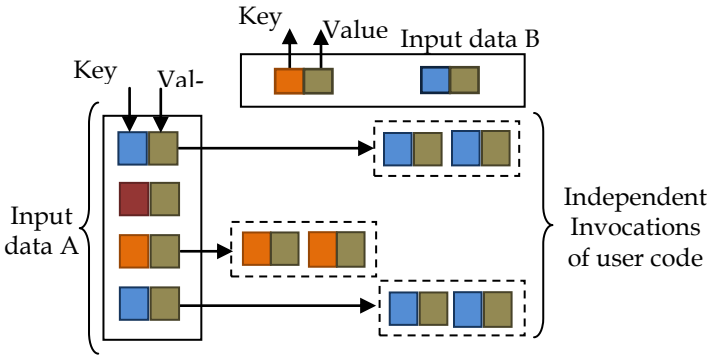

Fig 4.6 Match input contract

4.7 The PACT Output Contracts

Output Contracts are capable to declare indisputable properties of user functions to maximize the efficiency of the task execution. An example of such an output contract is the SameKey contract. When attached to a Map function then the user code will not change the key, i.e. the type and value of the key remain after the user code's invocation and are the same in the output as in the input. Those references can be used by an optimizer that to generates parallel execution plans. The mentioned SameKey contract can frequently help to avoid unnecessary repartitioning and therefore expensive data shipping. Hence, Output Contracts can significantly improve the runtime of a PACT program.

5. OPERATION

On given geo-distributed input dataset, a sequence of operations is performed. In the following we focus on parallelization contracts jobs J1 , J2 , ..., J m giving rise to a sequence of 5xm

operations. Since each parallelization contracts job consists of five major phases/operations (map, cross, match, cogroup, reduce). We define the state of data before a phase as a stage identified by 0..5m. So input data is in stage 0 and final output data received after applying MapReduce jobs 1...m is in stage 5m.

To move data from stage s to next stage s + 1 a parallelization contracts phase is applied to data partitions and the same number of (output) data partitions are created. The initial kth partition of data is denoted by Pk0, and Pks represents the output after executing parallelization contracts phases 1..s on partition Pk0. Before performing a MapReduce phase, a partition present in a datacenter may be moved to another datacenter. To make our solution tractable we only allow full partitions of data to be copied. The move may be for an initial partition or a derivative of it received after executing one or more parallelization contracts phases. Initial partition sizes can be used as parameter to trade accuracy and computation costs.

In this section executing sequence of parallelization contracts on given input datasets. One of the main execution path is multiple execution path. Execution path for executing individual parallelization contracts in each datacenter on given sub datasets and aggregating result from that datacenter is called multiple execution path. Aggregating result is not done automatically in a datacenter hence indexed aggregators are used. Moving a data partition from one datacenter to another is costly since this involves copying data across inter-datacenter links from one distributed file system to another. If a partition Psk is copied from DCi to DCj at stage s, this partition should not be copied back to DCj to DCi at s+1 stage and also not copied within same stage. Input sub-datasets are replicated across datacenters, by taking exactly one replica of each of the input sub-datasets.

5.1 Architecture

GEO-PACTs consist of single group manager in only one datacenter and job manager is deployed in every participating datacenter. Group manager finds the execution path and each job manager in datacenter manage phase of parallelization contract jobs that executed within datacenter using Hadoop cluster. Job manager contains two components which are copy manager and aggregation manager. Copy manager manage copying data from one datacenter to another and aggregation manager manage the aggregation of result copied from different datacenter. Data center configuration file describe about the datacenter which have to be participate in geographically distributed parallelization contracts jobs handled by geo-pacts. Identification is provided to each datacenter to identify when it is running parallelization contracts. Datacenters store their data in Hadoop distributed file system.

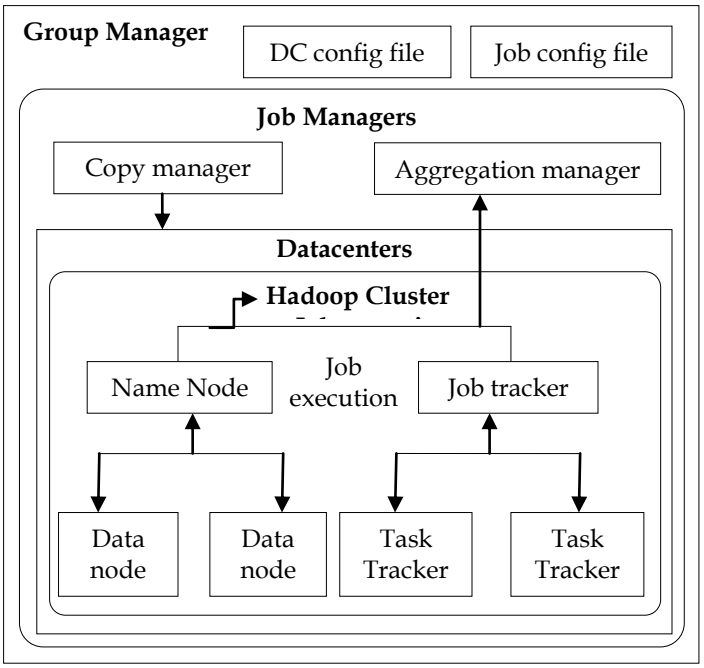

Fig 5.1 Architecture of GEO-PACT

Geographically distributed parallelization contracts jobs sequences are submitted to the Group manager through a job configuration file. Group manager breaks the job sequence into a number of tasks and this information is informed to Job manager components copy manager and aggregation manager to describe the portion of task that should be performed within corresponding datacenters. User can specify sub dataset of geographically distributed input dataset through a xml based job configuration file.

Once the group manager starts its execution in multiple execution paths it instructs each job manager about the parallelization contracts jobs that should be executed in corresponding datacenter and the respective sub-datasets those parallelization contracts jobs should be executed on. Job managers execute the jobs using the Hadoop clusters deployed in corresponding datacenters. The Group manager informs a Job manager to copy data to a corresponding datacenter or aggregate multiple sub datasets copied from two or more corresponding datacenters. A Job manager use local Copy manager and Aggregation manager to execute these tasks.

When ever a copy operation has to be performed, Copy manager in the datacenter from where the data has to be copied from and reads data from the corresponding HDFS storage and sends data to the Copy manager located in the datacenter to which data has to be copied, to through a TCP stream. The Copy manager in the receiving side stores this data in the corresponding HDFS distributed storage. GEO-PACT uses aggregators to aggregate the results obtained from parallelization contracts jobs in parts of the input. Once output data generated in one or more datacenters is copied to a single destination

datacenter, the Group manager instructs the Job manager running in the destination datacenter to initiate an aggregate operation.

6. DATACENTER ARCHITECTURE

Parallelization contract style computation systems have a minimum network bandwidth to move the data to the computation, instead of the computation must move to the data. In datacenter, compute nodes and storage nodes are attached to same network switch. Compute node contains central processing unit and network interface, and storage node contains disk to store data. Instead of co-locating storage and computation in the same box as in the traditional MapReduce storage architecture, this design co-locates storage and computation on the same Ethernet switch. The advantages of using remote disks in a datacenter are many. First, both computation node and storage node can be modified to meet the application requirements during both cluster construction and operation. Second, computation node and storage node can be decoupled from each other when they went to failure. It avoids wasted resources that would have been used to reconstruct lost storage when a computation node fails.

Third, fine-grained power management techniques can be used, whereby compute and storage nodes are enabled and disabled to meet current application requirements. Finally, because computation is now an independent resource, a mix of both high and low power processors can be implemented. The runtime environment managing application execution can change the processors being used for a specific application in order to meet administrative power and performance goals.

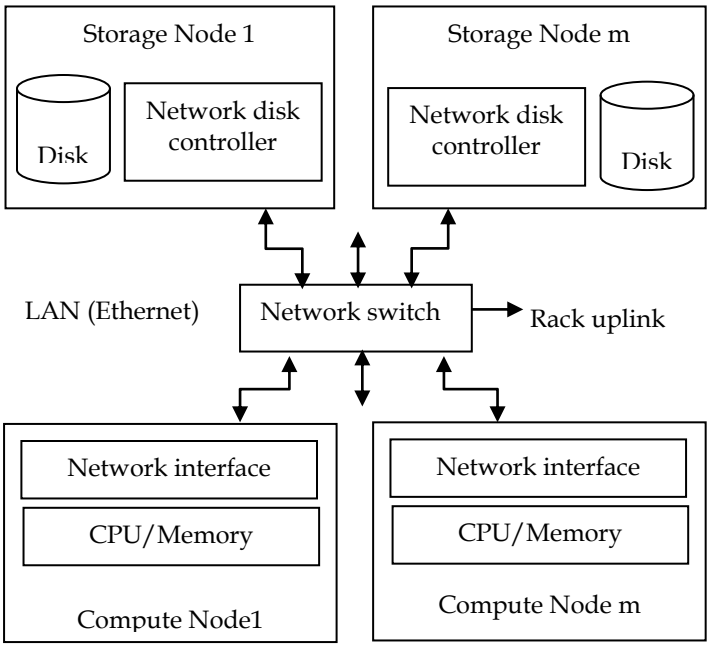

Fig 6.1 Remote Storage Architecture

7. FAILURE HANDLING

The Group manager must start in a trustworthy datacenter. Each Job manager frequently sends a heartbeat message back to the Group manager. If the Group manager does not receive a heartbeat from a corresponding Job manager for some time, a new Job manager is started in the same datacenter. If the Group manager receives a late heartbeat message from the old Job manager after starting the new Job manager, a message is sent back to the old Job manager to quit itself. Once a Job manager starts a parallelization contracts job it stores information about this job in a predefined location of the distributed file system in its datacenter so that a newly started Job manager can catch up. Failure of a Group manager will result in the failure of the GEO-PACT instance similar to the termination of the Job Tracker component of Hadoop.

8 CONCLUSIONS AND FUTURE WORK

This paper presents Geo-Pact, a extension of MapReduce framework that can efficiently execute a sequence of jobs on geographically distributed datasets. Minimizing either execution time or cost. GEO-PACTS can substantially improve time or cost of job execution compared to naive schedules of currently widely followed deployments. This framework is also applicable to single datacenters with non uniform transmission characteristics, such as datacenters divided into zones or other network architectures. Future work is to form clusters from the final result and with this analyzed data can be retrived are more efficiently.

REFERENCES

[1] Agarwal S, Dunagan J, Jain N, Saroiu S, Wolman A, and Bhogan H, "Volley: Automated Data Placement for Geo-Distributed Cloud Services," in National Spatial Data Infrastructure, 2010.

[2] Alexander Alexandrov, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, Daniel Warneke, "MapReduce and PACT - Comparing Data Parallel Programming Models".

[3] Apache Software Foundation, "Hadoop", http://hadoop.apache.org.

[4] "Big data", http://www.emc.com/campaign/bigdata/index.html

[5] ChamikaraJayalath, Julian Stephen, and Patrick Eugster, "From the Cloud to the Atmosphere: Running MapReduce across Datacenters", IEEE Transactions on Computers Special Issue On Cloud Of Clouds Volume 63, issue 1, pages 74-87, May 2013.

[6] Data from Year 2000 US Census", http://aws.amazon.com/datasets/Economics/2290.

[7] Dean J and Ghemawat S, "MapReduce Simplified Data Processing on Large clusters", in Operating Systems Design and Implementation, 2004.

[8] Dean J and Ghemawat S, "MapReduce Simplified Data Processing on Large Clusters", in Operating Systems

Design and Implementation, pages 137–150, 2004.

[9] Dejun J, Pierre D, and Chi C, "EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications", in International Conference on Service Oriented Computing, 2009.

[10] Fushimi S, Kitsuregawa M, and Tanaka H, "An Overview of The System Software of A Parallel Relational Database Machine GRACE", Very Large Data Base, pages 209–219, Morgan Kaufmann, 1986.

[11] "Hadoop the Definitive Guide", http://oreilly.com/catalog/ 9780596521981

[12] "Hadoop", http://hadoop.apache.org/index.html

[13] "HDFS", http://developer.yahoo.com/hadoop/tutorial/module2.html

[14] Introduction to Analytics of Big Data and Hadoop", Storage Networking Industry Association,http://www.snia.org/sites/default/files2/ABDS2012/Tutorials/RobPeglarIntroduction_Analytics%20_Big%20Data_Hadoop.pdf

[15] Introduction to hadoop mapreduce", http://developer.yahoo.com/hadoop/tutorial/module4

[16] Jeffrey Shafer, Scott Rixner, Alan L. Cox Rice University "Datacenter Storage Architecture for MapReduce Applications".

[17] Lloyd W, Freedman M J, Kaminsky M, and Andersen D J, "Don't Settle for Eventual: Scalable Causal Consistency for Wide area Storage with COPS", in ACM Symposium on Operating Systems Principles, 2011.

[18] Olston C, Reed B, Silberstein A, and Srivastava U, "Automatic Optimization of Parallel Dataflow Programs", Unix Users Group Annual Technical Conference, pages 267–273, Unix Users Group Association, 2008.

[19] Ousterhout J, Agrawal P, Erickson D, Kozyrakis C, Leverich J, Mazieres D, Mitra S, Narayanan A, Parulkar G, Rosenblum M, Rumble S M, Stratmann E, and Stutsman R, "The Case for RAMClouds: Scalable High-performance Storage Entirely in DRAM", ACM Special Interest Group on Operating Systems Operating Systems Review, volume 43, no 4, pages 92–105, January 2010.

[20] Pavlo A, Paulson E, RasinA, Abadi D J, DeWitt D J, Madden S, and Stonebraker M, "A Comparison of Approaches to Large-Scale Data Analysis", Special Interest Group on Management Of Data Conference, pages 165–178, ACM, 2009.

[21] Selinger P G, Astrahan M M, Chamberlin D D, Lorie D D, and Price T. G "Access Path Selection in a Relational Database Management System". In P. A Bernstein, editor, Special Interest Group on Management Of Data Conference, pages 23–34, ACM, 1979.

[22] Sovran Y, Power R, Aguilera M K, and Li J, "Transactional Storage for Geo-replicated Systems", in ACM Symposium on Operating Systems Principles, 2011.

[23] "Teradata", http://www.teradata.com

[24] vorgelegt von, Daniel Warneke, Berlin, "Massively Parallel Data Processing on Infrastructure as a Service Platforms".

[25] Yang H, Dasdan A, Hsiao R, and Parker D S, "MapReduce-Merge: Simplified Relational Data Processing on Large Clusters", in Special Interest Group on Management of Data, 2007.

[26] Zahariab M, Konwinski A, JosephA D, Katz R H, and Stoica I, "Improving MapReduce Performance in Heterogeneous Environments", in Operating Systems Design and Implementation, 2008.