PROFILE BASED CONCURRENT DATA DOWNLOAD IN CLOUD WITH DATA SHARING AND LOAD BALANCING

S. Pradeepa, E. Priyadarsini

Abstract—**Multi element system and multithread processing are now the really excellence of enterprise and personal computing. If accessed constant way. Also multithread processing might literally demean performance. We present the surface of the memory access congestion as they obvious in multithread processing and view their crash an query evolution. We access a system design based on the division parallelism, arranged base on the pooling, and data structures unfavorable to multithread processing. Based on the design, we are going to download at the load time of the data itself. Here we using the concept partition parallelism. If the data owner loads to the cloud server and its splits the into hurls using the partition algorithm. The splitted part of the data uploaded in the cloud server very first, after that the client to download the data from the server. The data stored in an encrypted format in the server.**

Index Terms—**Multicore system, multithread process, partition parallelism, Encryption.**

I. INTRODUCTION

Parallel database system have long been successful application. The highly parallelism portion are present in the data flow of query processing. By hire the partition parallelism, it has been attainable to create highly scalable parallel database system. That reveal almost complete linear speedups. To impose the partition parallelism always the system architecture is must be conducive [1] [2] [3].

The modern CPUs are parallel mechanism themselves. It will be surrounded multiple processing cores in a single chip. They differs from shared mechanism, also there is nothing in the execution model [4-9].

Parallel database systems have long been a success story, as the data flow of query processing algorithms exhibits highly parallelizable portions. By employing partition parallelism, it has been possible to build highly scalable parallel database systems that exhibit almost ideal linear speedups. To enforce partition parallelism, each processing node in the system independently processes a partition of the data set. No Sharing is enforced either at load-time, with data preprocessed and partitioned and each partition shipped to different nodes of the system; or at query-time, by dynamically splitting a data set into disjoint partitions [10] [11].

II. RELATED WORKS

A. Existing System

In this model, each processing node will partition a data set individualistic. If the server processing the data there is no connection between user and server. There is no data transaction in between them. Its is the major disadvantage of this system [12] [13] [14].

In the load time there is no data transaction in user and server. Time consuming is very high, also it taking it. Then we proposed the this model using partition parallelism [15-19].

B. Existing Usage

In the Proposed System, we are developing Two Techniques namely Data Download and Data spliting. The data download model based as priority based on the user query. The appealing of data is downloaded from different servers as the data are divided. In data sharing, the data are portioned into different hurl and stocked as threads in the division matrix. From the division matrix the data will be recovered for the read/write purpose without overlapping.

Using the concept of partition parallelism, here we are going to achieve data download at the load time of the data itself. The data owner loads the data to the cloud server and the cloud server splits the data into small chunks using the partition algorithm. When the first splitted part of the data is uploaded to the cloud server the client can download the data. These data are stored in an encrypted form in the server. C. Implementation of NFS.

In this proposed model data also is achieved during the data load time. It's reduce user participating time. Its also to reduce waiting timing. This will be longer to when the upload complete.

III. METHODS

The proposed system to implement the partition parallelism we follow below methods:

***S. Pradeepa**, department of computer science and engineering, Mailam Engineering college, Mailam, Villupuram, India,*
***E. Priyadarsini**, Mailam Engineering college, Mailam, Villupuram, India.*

A. User Registration

In this module we are going to create an User application by which the User is allowed to access the data from the Server of the Cloud Service Provider. Here first the User wants to create an account and then only they are allowed to access the Network. Once the User creates an account, they are to login into their account and request the Job from the Cloud Service Provider. All the User details will be stored in the Database of the Cloud Service Provider. All the User details will be stored in the Database of the Cloud Service Provider. In this Project, we will design the User Interface Frame to Communicate with the Cloud Server through Network Coding using the programming Languages like Java/ .Net. By sending the request to Cloud Server Provider, the User can access the requested data if they authenticated by the Cloud Service Provider [20] [21] [22].

When the user wants to access the cloud sources, first the user to register the detail in cloud server. User has to provide authentic mail id and password for the authentication of user. The sever to check the availability it will provide the token to the user. When ever the user to get the token the registration will be completed [23] [24].

B. Cloud Deployment

Cloud Service Provider will contain the large amount of data in their Data Storage. Also the Cloud Service provider will maintain the all the User information to authenticate the User when are login into their account. The User information will be stored in the Database of the Cloud Service Provider [25] [26] [27-29].

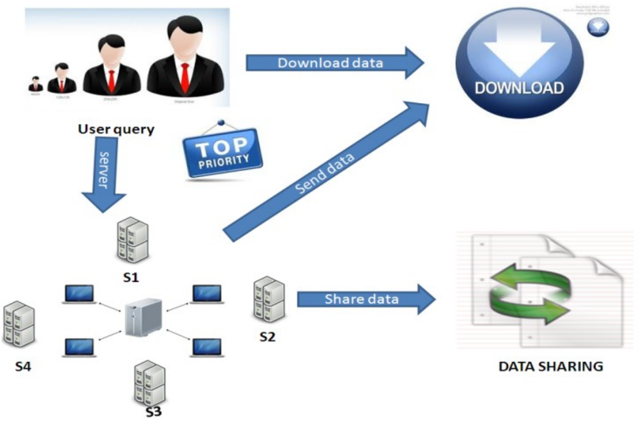
The Cloud Server will establish connection. For this Purpose we are going to create an User Interface Frame. Also the Cloud Service Provider will send the User Job request to the Resource Assign Module in Fist In First Out (FIFO) manner.

C. Data Encryption & Chunking

In this module once the data is upload by the cloud owner the data is spitted and store in the cloud server and spitted data are encrypted and they are stored in the cloud server by using encryption technique the data kept safe in the cloud sever.

D. Load Balancing & Data Delivery

In this Module, we will Process User requested Job. The User requested Job will redirect to the RA of Cloud Server. And the file will be divided into six divisions So that the Job can be efficiently processed. And we also focused how to reduce the workload of server .so that multiple users can access the server randomly and efficiently for that purpose we implement sub server to handle request .which is request to the main server so that total work load of main server is reduced [30] [31] [32].



E. Priority Based Profile Filtering

An information processing system employs an integrated profile based information filter structure to find cloud informants relevant to information desired by an individual user. The filter structure includes a two-level profile-based filter which preprocesses information in a first level to pass only relevant information, applies community filters in a second level to pass relevant information to matching communities of users, and applies in a bottom level user filters within each community to pass relevant information to matching users [33] [34].

IV. LITERATURE SURVEY

A. Hoard: A Scalable Memory Allocator for Multithreaded Applications

This paper introduces Hoard, a fast, highly scalable allocator that largely avoids false sharing and is memory efficient. Hoard is the first allocator to simultaneously solve the above problems. Hoard combines one global heap and per-processor heaps with a novel discipline that provably bounds memory consumption and has very low synchronization costs in the common case [35].

B. Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs

This paper dissects each internal phase of a typical hash join algorithm and considers different alternatives for implementing each phase, producing a family of hash join algorithms. Then, we implement these main memory algorithms on two radically different modern multi-processor systems, and carefully examine the factors that impact the performance of each method [36].

C. Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources

Scheduling query execution plans is a particularly complex problem in hierarchical parallel systems, where each site consists of a collection of local time-shared (e.g., CPU(s) or disk(s)) and space-shared (e.g., memory) resources and communicates with remote sites by message passing. We present heuristic algorithms for various forms of the problem, some of which are provably near-optimal. Preliminary experimental results confirm the effectiveness of our approach [37].

D. Real-Time Parallel Hashing on the GPU

We demonstrate an efficient data-parallel algorithm for building large hash tables of millions of elements in real-time. We consider two parallel algorithms for the construction: a classical sparse perfect hashing approach, and cuckoo hashing, which packs elements densely by allowing an element to be stored in one of multiple possible locations [38].

E. Volcano-An Extensible and Parallel Query Evaluation System

To investigate the interactions of extensibility and parallelism in database query processing, we have developed a new dataflow query execution system called Volcano. The Volcano effort provides a rich environment for research and education in database systems design, heuristics for query optimization and resource allocation [39].

V. CONCLUSIONS

We implement the partition parallelism we reduce the load timing of the when the user to access the data in cloud server. Because they contain different chunks and stored as threads format. From the partition matrix the data will be retrieved for the read/write purpose without overlapping [40].

REFERENCES

[1] R. Acker et al., “Parallel Query Processing in Databases on Multicore Architectures,” Proc. Eighth Int’l Conf. Algorithms and Architectures Parallel Processing (ICA3PP), 2008.

[2] D.A. Alcantara et al., “Real-Time Parallel Hashing on the GPU,” Proc. ACM SIGGRAPH, 2009.

[3] E.D. Berger et al., “Hoard: A Scalable Memory Allocator for Multithreaded Applications,” Proc. Ninth Int’l Conf. Architectural Support Programming Languages and Operating Systems (ASPLOS), 2000.

[4] S. Blanas et al., “Design and Evaluation of Main Memory Hash Join Algorithms for Multi-Core CPUs,” Proc. ACM SIGMOD Int’l Conf. Management of Data, 2011.

[5] R.D. Blumofe et al., “Cilk: An Efficient Multithreaded Runtime System,” Proc. Fifth ACM SIGPLAN Symp. Principles and Practice Parallel Programming (PPoPP), 1995.

[6] L. Bouganim et al., “Dynamic Load Balancing in Hierarchical Parallel Database Systems,” Proc. 22th Int’l Conf. Very Large Data Bases (VLDB), 1996.

[7] L. Bouganim et al., “Load Balancing for Parallel Query Execution on NUMA Multiprocessors,” Distributed and Parallel Databases, vol. 7, no. 1, pp. 99-121, 1999.

[8] M.-S. Chen et al., “Scheduling and Processor Allocation for Parallel Execution of Multi-Join Queries,” Proc. Eighth Int’l Conf. Data Eng. (ICDE), 1992.

[9] J. Chhugani et al., “Efficient Implementation of Sorting on Multi-Core SIMD CPU Architecture,” Proc. VLDB Endowment, vol. 1,no. 2, pp. 1313-1324, 2008.

[10] J. Cieslewicz and K.A. Ross, “Adaptive Aggregation on Chip Multiprocessors,” Proc. 33rd Int’l Conf. Very Large Data Bases (VLDB), 2007.

[11] M. Cole, Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press, 1989.

[12] D.J. DeWitt, “The Wisconsin Benchmark: Past, Present, and Future,” Benchmark Handbook for Database and Transaction Systems, Morgan Kaufmann, 1993.

[13] D.J. DeWitt and J. Gray, “Parallel Database Systems: The Future of High Performance Database Systems,” Comm. ACM, vol. 35, no. 6, pp. 85-98, 1992.

[14] D.J. DeWitt et al., “The Gamma Database Machine Project,” IEEE Trans. Knowledge Data Eng., vol. 2, no. 1, pp. 44-62, Mar. 1990.

[15] R. Fang et al., “GPUQP: Query Co-Processing Using Graphics Processors,” Proc. ACM SIGMOD Int’l Conf. Management of Data, 2007.

[16] P. Garcia and H.F. Korth, “Database Hash-Join Algorithms on Multithreaded Computer Architectures,” Proc. Third Conf. Computing Frontiers (CF), 2006.

[17] P. Garcia and H.F. Korth, “Pipelined Hash-Join on Multithreaded Architectures,” Proc. Third Int’l Workshop Data Management New Hardware (DaMoN), 2007.

[18] M.N. Garofalakis and Y.E. Ioannidis, “Multi-Dimensional Resource Scheduling for Parallel Queries,” Proc. ACM SIGMOD Int’l Conf. Management of Data, 1996.

[19] M.N. Garofalakis and Y.E. Ioannidis, “Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources,” Proc. 23rd Int’l Conf. Very Large Data Bases (VLDB), 1997.

[20] G. Graefe, “Volcano - An Extensible and Parallel Query Evaluation System,” IEEE Trans. Knowledge and Data Eng., vol. 6, no. 1, pp. 120-135, Feb. 1994.

[21] B. He et al., “Relational Query Coprocessing on Graphics Processors,” ACM Trans. Database Systems, vol. 34, no. 4, article 21, 2009.

[22] R. Johnson et al., “To Share or Not to Share?” Proc. 33rd Int’l Conf. Very Large Data Bases (VLDB), 2007.

[23] C. Kim et al., “Sort Vs. Hash Revisited: Fast Join Implementation on Modern Multi-Core CPUs,” Proc. VLDB Endowment, vol. 2, no. 2, pp. 1378-1389, 2009.

[24] M. Kitsuregawa et al., “Application of Hash to Data Base Machine and Its Architecture,” New Generation Computing, vol. 1, no. 1, pp. 63-74, 1983.

[25] K. Krikellas et al., “Modeling Multithreaded Query Execution on Chip Multiprocessors,” Proc. Int’l Workshop Accelerating Data Management Systems Using Modern Processor and Storage Architectures (ADMS ’10), 2010.

[26] K. Krikellas et al., “Scheduling Threads for Intra-Query Parallelism on Multicore Processors,” Technical Report EDI-INF-RR-1345, Univ. of Edinburgh, 2010.

[27] R. Lee et al., “MCC-DB: Minimizing Cache Conflicts in Multi-Core Processors for Databases,” Proc. VLDB Endowment, vol. 2, no. 1, pp. 373-384, 2009.

[28] B. Liu and E.A. Rundensteiner, “Revisiting Pipelined Parallelism in Multi-Join Query Processing,” Proc. 31st Int’l Conf. Very Large Data Bases (VLDB), 2005.

[29] M.-L. Lo et al., “On Optimal Processor Allocation to Support Pipelined Hash Joins,” Proc. ACM SIGMOD Int’l Conf. Management of Data, 1993.

[30] S. Manegold, P. Boncz, and M.L. Kersten, “Generic Database Cost Models for Hierarchical Memory Systems,” Proc. 28th Int’l Conf. Very Large Data Bases (VLDB), 2002.

[31] S. Manegold, M.L. Kersten, and P. Boncz, “Database Architecture Evolution: Mammals Flourished Long Before Dinosaurs Became Extinct,” Proc. VLDB Endowment, vol. 2, pp. 1648-1653, 2009.

[32] R. Pagh and F.F. Rodler, “Cuckoo Hashing,” J. Algorithms, vol. 51, pp. 122-144, 2004.

[33] L. Qiao et al., “Main-Memory Scan Sharing for Multi-Core CPUs,” Proc. VLDB Endowment, vol. 1, pp. 610-621, 2008.

[34] J. Reinders, Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. O’Reilly, 2007.

[35] K.A. Ross and J. Cieslewicz, “Optimal Splitters for Database Partitioning with Size Bounds,” Proc. Int’l Conf. Database Theory (ICDT), 2009.

[36] E.J. Shekita et al., “Multi-Join Optimization for Symmetric Multiprocessors,” Proc. 19th Int’l Conf. Very Large Data Bases (VLDB), 1993.

[37] J.S. Vitter, “Random Sampling with a Reservoir,” ACM Trans. Math. Software, vol. 11, pp. 37-57, 1985.

[38] F.M. Waas and J.M. Hellerstein, “Parallelizing Extensible Query Optimizers,” Proc. ACM SIGMOD Int’l Conf. Management of Data, 2009.

[39] D. Xu, C. Wu, and P.-C. Yew, “On Mitigating Memory Bandwidth Contention through Bandwidth-Aware Scheduling,” Proc. 19th Int’l Conf. Parallel Architectures and Compilation Techniques (PACT), 2010.

[40] J. Zhou et al., “Improving Database Performance on Simultaneous Multithreading Processors,” Proc. 31st Int’l Conf. Very Large Data Bases (VLDB), 2005

S. Pradeepa B.Sc computer science and MCA in Nehru memorial college, Bharadhidasanuniversity, Trichy. M.Tech in Computer science at ManonmaniyamSundharanar University, Thirunelveli. No. of conferences attended: 2

E. Priyadarsini Did B.E- Computer science and engineering in V.R.S College of engineering and technology, Arasur, Villupuram. No. of conferences attended: 2