International Journal of Applied Engineering Research
ISSN 0973-4562 Volume 10, Number 4 (2015) pp. 11545-11554

http://www.ripublication.com

Profile Based Concurrent Download and Data Sharing

K. Kirubhaharan,
Department of Computer Science
Sathyabama University, Chennai
Kk10993@gmail.com
K. Henry Arokiaraj
Department of Computer Science
SathyabamaUniversity, Chennai
jack_henry47@yahoo.com
Mr. PioSajin
Assistant professor
Department of Computer Science
SathyabamaUniversity, Chennai

Abstract

Cloud computing is an emerging technology widely being considered in the field of research. Among the enormousareas within the technology of cloud, the proposed system concentrates on improvising performance issues in file sharing.Whereas file sharing within the nodes under a cloud with the reduced space and timeconsuming is a challenging task. The proposed system tries to achieve this by enabling concurrent download for the clients. These clients are classified as high priority and low priority users using the classification algorithm.Based on the classification generated the rate of concurrent download is set.As we use the concurrent download concept,we implement Load balancing algorithm to assign different intermediate servers for users and to choose the best intermediate server for downloading the file. In case of any interruption in the download, instead of terminating the process, the system enables file download or upload resume concept.Introducing the technique in cloud avoids data loss and reduces time consumption in higher rates.

Keyword: Load balancing, File sharing, Concurrent download, File resume.

Introduction

Parallel database systems have long been a famous story, as the query processing algorithm 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, however, the underlying system architecture should be contributing. In the end, shared-nothing architectures have traditionally been used each processing node in the system independently processes a partition of the data set. No sharing is implemented 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. Though a shared-nothing architecture is still the way to go. It is interesting to see what happens at the level of a single processing node. The reason is that contemporary CPUs are parallel machines themselves, by placing multiple processing cores on a single chip. However, they differ from shared-nothing machines as there is nothing in the execution model, implementing all core process disjoint sets. Sharing is at multiple levels of the processing stack, for example, the memory hierarchy, or system resources. Thus, it is up to the programmer to enforce parallelism constraints at runtime. In this paper, we present and evaluate parallel implementations of the fundamental query processing algorithms tailored for execution on multicore systems. The key conception of processing in multicore systems is the thread a single execution flow supported by hardware. Multicore systems, process data by concurrently executing multiple threads, with the resulting perspective termed multithreaded processing. Another key concept is the hardware context: the logical processor that executes a single thread. Different multicore chips, also known as chip multiprocessors or CMPs, implement hardware contexts in different ways. This results in a multitude of hardware designs. Each design has its own advantage and disadvantage, some of which generically appear in any type of processing, whereas others manifest specifically in the database query processing. Regardless of the architecture, there is one main bottleneck that is aggravated when it comes to multicore chips: access to main memory. To that end, we empirically confirm and evaluate the three facets of the memory bottleneck in a multicore context, and exhibit their impact on query evaluation. After finding the problems, the natural next step is to enrich query engines with primitives that overcome these problems and specifically target multithreaded query evaluation. We present a query engine architecture that is based on the familiar concept of partition parallelism adapted for the multicore setting. This design is established on a combination of data partitioning and taskbased processing for effective parallelization, managing memory in a way that eliminates the need for per-thread memory pools while, at the same time, allows the system to scalably cater for the memory needs of the concurrently executing threads and data structures which are efficiently manipulated by multiple threads without any need for complex synchronization.

Allowing such a design in place us to implement efficient multithreaded versions of the fundamental query processing algorithms, namely selections, projections, partitioning, joint evaluation, sorting, and aggregation. For each algorithm we present several alternatives. All of them use the principles of the underlying design and strive for simplicity and hardware independence. We undertake an extensive experimental study to compare the performance of the algorithms in a variety of scenarios, both in terms of data and query properties and in terms of underlying hardware. Our results show that the choice of algorithm is neither a clear nor an easy one. The optimal

choice depends on the execution model of the hardware and the properties of the input and query at hand. Coming up with a high-performing implementation of a specific algorithm on a particular type of hardware and for a specific type of input is not our goal. Rather, our goal is to develop and evaluate generic multithreaded query processing primitives that can then be ported to and optimized for specific hardware. With a good system and algorithmic designs, it is indeed possible to have predictable and scalable performance across hardware. We believe our work serves as a starting point toward having a powerful toolkit of multithreaded query processing solutions.

Objective

The main aim of the process is to share the data with the requested users based on their priority status and profile based user authority. The users should be classified based on their priority and the file access should be provided based on the classification made. File resume should be enabled in the server and client side users for uploading and downloading the files from the cloud resource. Security issues, yet another challenge have to be handled.

Problem Definition

As we all know in the cloud, data security is not good as we expect and priority based approach is not also achieved. Chunking process is not yet done in the cloud. When the data or connection lost, we cannot resume the data packet for time saving. It has to start downloading from the beginning.

Existing System

In the Existing System, Data Sharing and Profile based User setting are not achieved so far in the cloud. There is no Load Balancing application is implemented so far. Encryption of Data alone is achieved and not the Data chunking, Data security is not as strong as we expect.

Disadvantages

Security implementation is not as good so far. Profile based User Classification is not achieved and Data Splitting is not yet done. We cannot auto resume the data packet at the time of connection problem.

Proposed System

In the previous method, data download and data sharing concepts are implemented.Data is split into number of parts and stored in a different sub server.So data is divided into different chunks and stored in a partition matrix.In thedownloading part ,as per the request from the user,data will be downloaded from

the different server.Priority based method is introduced in this process.So the high priority user gets more advantage in download.Download will start for the lowpriority user once after the completion of data for high priority.each and every data will be retrieved for read and write purpose without overlapping **.** So thenwhat we have changed isthat We have included the Process in the real-time Cloud. Data sharing and data download are achieved as said in the previous method. The Original data is Encrypted using AES algorithm. So owner upload a file to the cloud,then the data split into chunk like 8 parts and each file will be uploaded into the drop box.High priority user can download the data concurrently but low priority user will wait till entire data reaches the drop and they can only view.We have not given option to low priority user to download,because according to this concept high priority users get more advantages and benefits through this modification.If a high priority user downloading a data packet from the drop box.If any problem occurs due to the bad internet connection,they need not to fear about this issue.In this system we included resume concept for uploading as well as downloading the file.So users need not to worry about the packet loss.

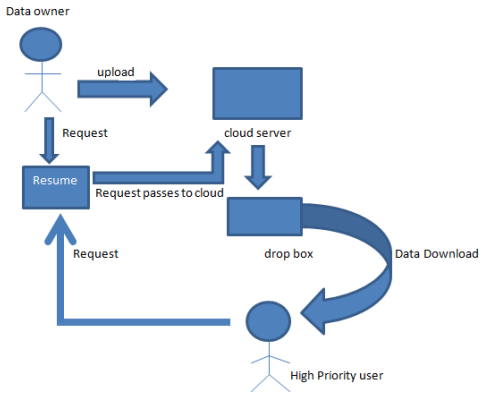
Literature Survey:

Web server ,new servers,database manager and some scientific application are becoming common.the memory allocator is often a barrier for these application and it limits the program performance and scalability.Previous allocator had suffered this problem of scalability and poor performance.Heap organization that announce false sharing.Many allocators display a impressive increase in memory consumption when face with the producer consumer pattern of object freeing and and object allocation.So increase in memory consumption can be ranged from a P factor to unbounded memory consumption ,this paper introduced high salable, a fast allocator that avoids false sharing and it is memory efficient.the combination of one global heap and preprocessor heap with a novel discipline which is done by hoard that possibly bounds memory consumption and very low synchronization cost in the familier cases.

Scheduling query execution is a complex problem in hierarchical parallel system.In there each sit consist of compilation of local time shared and space shared resource and transmit information with remote sites by message passing.we create a general approach to solve the problem,captures the complication of scheduling distributed multi dimensional resource unit for all types of parallelism within and crosswise queries and operator.So heuristic algorithm for different forms of the problems,some of that are near optimal.

To build an extensive hash table of millions of elements in real time,we prove an efficient data parallel algorithm.We study two parallel algorithms for the construction one is cuckoo approach and other one is classical sparse perfect hashing approach,that packs elements dumbly by allowing n element to store in one of the different possible location. The hybrid approach uses both algorithm.we mensurate access time, construction time and measure usage of implementations and determine real time performance on large datasets.

Architecture Diagram:



Methods

Registration

To access the network,users have to create an account and they need to login into their account.By doing that users are able to send the requested job to cloud service provider.Two keys have to be created.One is public key to upload a data and the other one is private is just to download or access the data in the cloud server.so whoever creates a login, all their details will be stored in the database of the cloud service provider.User interface frame is designed to have a communication with the cloud server through the network coding.Once if a user sends a request,it should be authenticated or getting permission from the cloud service provider to access the requested data

Cloud deployment

Cloud service providers will have more number of data in data storage. so they maintain all the user information and includes the login details.So every information related to the owner of the data and the user will be stored in the database. The user request will be redirected by the cloud server to any of the queue.The virtual machine will take care all the request in the queue. To establish a connection enables them to communicate with the client and with the other module of the cloud network. so we create the user interface frame for this purpose. In first in, first out manner all the user request job will be sent by the cloud service provider.

Classifying the user according to the priority based

In this concept we are introducing two types of profiles. One is a high priority and the other one is low priority. Based on the priority we provide the features like upload and download. For high priority user concurrent download is possible. So by creating this, we can reduce the load of the cloud server by doing this service.For the Low priority,Only view option is enabled for them.So the top priority user gets more advantage in this process.

Chunking process and encrypting the data

Before we put a large file on a cloud, we will chunk the data into a number of parts like splitting a file into eight parts. So we can reduce the load of cloud server and this is one of the benefits instead of putting a file without being chunked. We need to provide some security to the data before uploading the data to the sub cloud servers. Data is encrypted and chunked before upload to the cloud server. For the encryption we are using AES algorithm and so this is very helpful to enhance the quality of the service while user uploading or downloading the data. Everything which we do is mainly to enhance the security in the cloud.

Concurrent data transfer

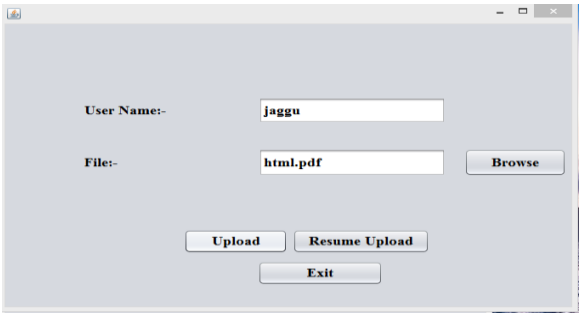
In this paper,we are implementing the concurrent data transfer processing .In that multiprocessor executes instructions simultaneously for getting a better performance.In a concurrent download,user need not to be waited till the entire data reach the dropbox.So user can download the data at the same time when the owner uploads it to the drop box.Data is split into eight number of parts so when owner uploads a part 1 of the data to the drop box,at the same time user can access the part 1 of the data.Unlike the high priority users,low priority users have to wait till the entire data packet reach the drop box.this concurrent download helps to reduce waiting and downloading time for the high priority users.

Load balancing

Users requested job is to be handled by the cloud service provider.So the cloud service provider passes the request to the sub cloud server.In the sub cloud server only each part of the data will be stored.Based on the user's request,the data owners retrieve the data through the cloud server.the load balancing in which users job request will be assigned to any of the sub cloud server which contains the required data and with the help of load balancing approach,user's request is passed to the concerned sub cloud server contains minimum load to handle.We are using four intermediate server.From that we select the best one to process the user's request.

Resuming the data packet

As we all know that data packet loss is a common issue in the cloud.When user tries to download a large data,usually connection lost or any connection oriented problems may occur.So by introducing the concept of resuming the data while download and upload in the dropbox,it avoids the data loss for the users.As we know that each data is split into no of parts while uploading as well as downloading.When the connection terminates at the time of uploading or downloading part 3 to the drop box,with the help of resume upload or download,it is possible sending the request of packet loss to the main server and we can regain the lost process.This seems to be the effective modification that we have applied for the top priority user.



Conclusion

We presented design primitives and parallel pattern-based implementations of the fundamental query processing operators and evaluated their performance in a variety of settings and execution environments. Our evaluation shows that it is indeed possible to effectively use pattern-based parallelism for efficient query processing. As a general set of guidelines, our results show

1. That synchronization should be minimized: data structures like the partition matrix reduce the need for synchronization and significantly speed up even simple operations like selections;
2. That the less uniform a distribution is, the more appropriate techniques like size-bound partitioning become;
3. That multiple passes are not detrimental so long as they are all performed in parallel: for instance, for machines with many hardware contexts, Multipass Algorithms based on a count-partitioning will likely perform well; and
4. That deeper memory hierarchy's make up for a wrong choice of algorithm: they improve the utility of the higher level caches. Our study reinforces the notion that the optimal choice of algorithm is quite sensitive to the hardware, the number of threads used, and to perturbations of the input parameters. This verifies the increased complexity of the problem and the need for elaborate analytical cost models.

Reference

[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.